



Program Studi Teknik Informatika  
&  
Program Studi Sistem Informasi

Fakultas Ilmu Komputer  
Universitas Esa Unggul  
2017



# Modul Praktikum Pemrograman Berorientasi Objek

M. Bahrul Ulum, S.kom, M.Kom



# MODUL 1

## Konsep OOP dan Instalasi Netbeans

### 1. Tujuan Pembelajaran

1. Praktikan dapat melakukan instalasi dan setting Java Development Kit.
2. Praktikan dapat menggunakan Jcreator sebagai editor pemrograman
3. Praktikan dapat menjalankan (eksekusi) program Java sederhana

### 2. Teori Singkat

## OOP Concepts

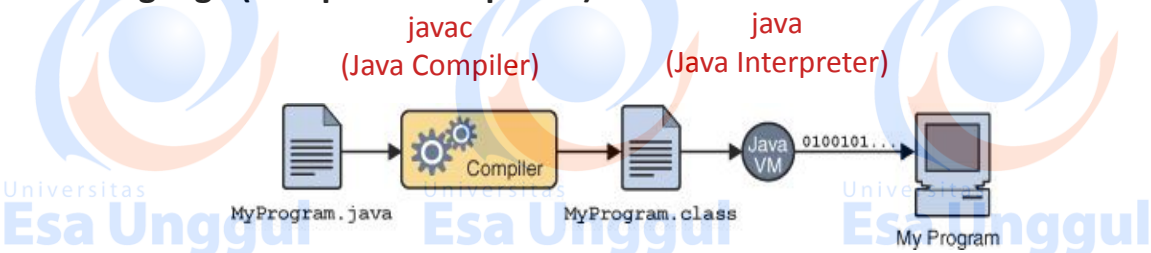
### Bahasa Pemrograman?

- ✓ Komputer bekerja seperti *switching* dan hanya mengenali 0 dan 1
- ✓ Manusia tidak (paham) berbicara dengan bahasa 0 dan 1
- ✓ Perlu bahasa pemrograman yang dapat menjadi perantara percakapan antara komputer dan manusia
- ✓ Bahasa pemrograman diubah ke dalam bahasa yang dipahami oleh komputer dengan menggunakan interpreter atau kompililer

### Compiler or Interpreter?

- ✓ Compiler = Mengkompilasi source code menjadi bentuk file yang bisa dieksekusi
- ✓ Interpreter = Mengkompilasi dan menjalankan source code secara langsung

### Java Language (Compiler+Interpreter)



### Tingkat Bahasa Pemrograman

- ✓ Bahasa Pemrograman Tingkat Rendah (Assembler)
- ✓ Bahasa Pemrograman Tingkat Sedang (C, Pascal, Fortran)
- ✓ Bahasa Pemrograman Tingkat Tinggi (Java, C++, C#)

### Paradigma Pemrograman

Sudut pandang dan style pemrograman berhubungan dengan bagaimana sebuah masalah diformulasikan dalam bahasa pemrograman

- ✓ Functional Programming = Urutan fungsi secara sekuensial (Scheme, Lisp)
- ✓ Procedural Programming = Pemecahan masalah berdasarkan prosedural kerja yg terkumpul dalam unit pemrograman bernama fungsi (C, Pascal)
- ✓ Object-Oriented Programming = Koleksi object yang saling berinteraksi . Class adalah unit pemrograman (Java, C#, C++)

### Sejarah Java

- ✓ James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991
- ✓ The language was initially called *Oak* after an oak tree that stood outside Gosling's office
- ✓ It went by the name *Green* later, and was later renamed *Java*, from a list of random words
- ✓ Gosling aimed to implement a virtual machine and a language that had a familiar C/C++ style of notation
- ✓ Sun Microsystems released the first public implementation as Java 1.0 in 1995
- ✓ On May 8, 2007, Sun finished the process, making all of Java's core code available under free software/open-source distribution terms (GNU Public License)

## Java Family Suite

- ✓ Java Standard Edition (Java SE) For desktop, client/server application
- ✓ Java Enterprise Edition (Java EE) For e-business, e-commerce web based application
- ✓ Java Micro Edition (Java ME) For small devices, like palm, handphone, etc

## Why Java?

- ✓ Simple and familiar object oriented programming
- ✓ Architecture neutral (platform independent)
- ✓ Open Source
- ✓ First rank in TIOBE Index
- ✓ De-Facto standard programming language in education

## Perangkat Pemrograman Java

- ✓ Compiler (Interpreter): Java Standard Edition (JSE)
- ✓ Code Editor:
  1. Text Editor: TextPad, Notepad++
  2. Integrated Development Environment (IDE): Netbeans, Eclipse, Jcreator

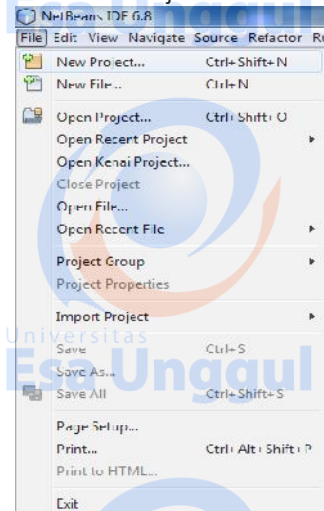
## Instalasi Java SE dan Netbeans IDE

- ✓ Instalasi Java SE dengan mengklik: jdk-7u21-windows-i586.exe  
(download dari: <http://java.sun.com/javase/downloads>)
- ✓ Instalasi Netbeans dengan mengklik: netbeans-7.3-ml-windows.exe  
(download dari: <http://netbeans.org>)
- ✓ Ikuti seluruh proses instalasi sampai selesai

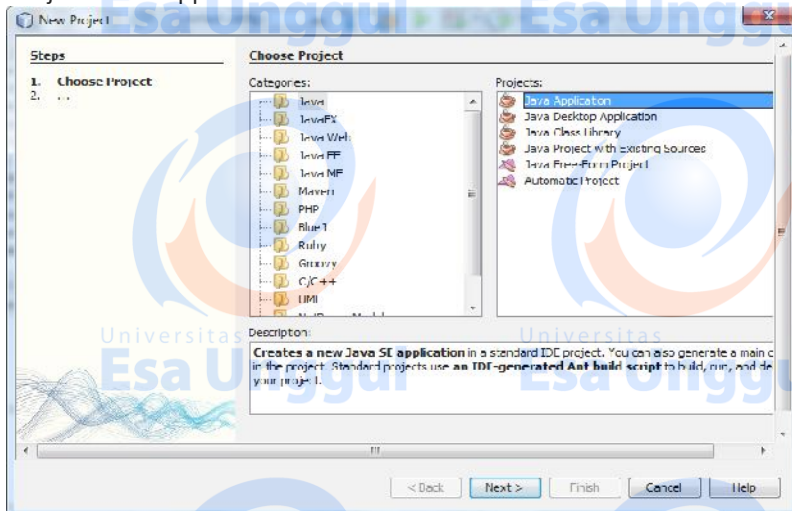
## Membuat Program dengan Netbeans

```
public class HelloJakarta{  
    public static void main(String[] args){  
        System.out.println("Halo Jakarta");  
    }  
}
```

### File – New Project

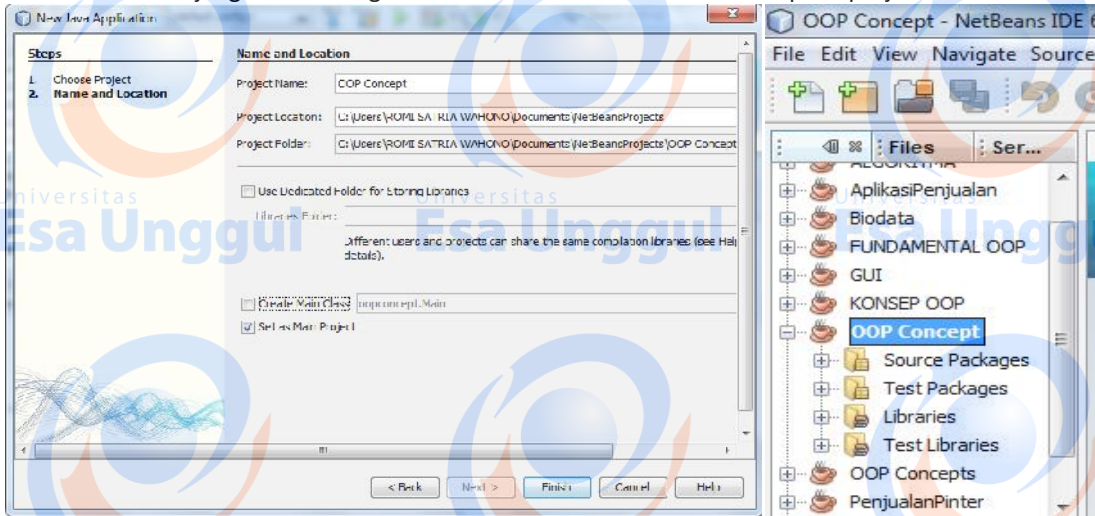


### Projects: Java Application



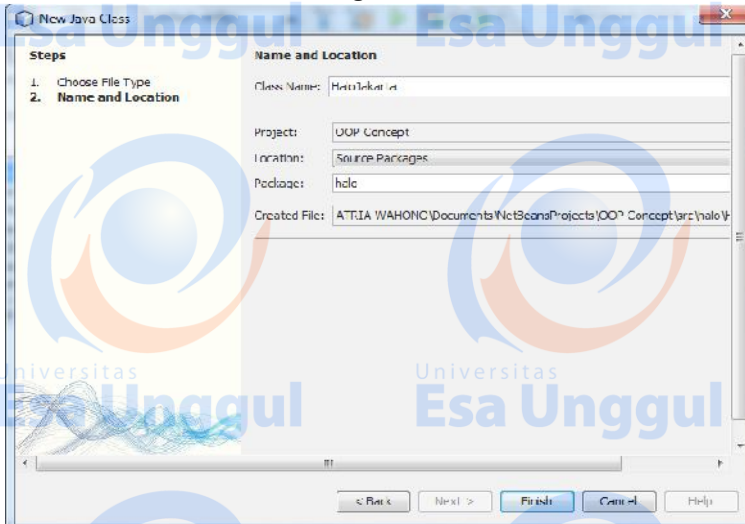
Project name: Konsep OOP  
Project location: c:\Lab\praktikum  
Create main class jangan dicentang

Klik kanan pada project OOP – New – Java Class



Class Name: HalloJakarta, Package: halo

Ketik script di bawah ini:



```
package halo;  
  
/**  
 * @author ROMI SATRIA WAHONO  
 */  
public class HalloJakarta {  
    public static void main(String[] args) {  
        System.out.println("Halo Jakarta");  
    }  
}
```

keterangan:

a. Syntax utama untuk program yang ditulis dengan JAVA adalah:

```
[modifier][class] nama_class
{
...
}
```

Ada beberapa modifier pada JAVA, yaitu public, private dan protected. Modifier public diperlukan agar program dapat dikompilasi dengan baik.

b. Bagian ini merupakan bagian utama yang ditempatkan pada bagian awal pada program JAVA. static menunjukkan tipe method. void menunjukkan bahwa method tidak mengembalikan nilai atau objek. main merupakan nama method utama program JAVA. String merupakan tipe argumen yang akan diterima sebagai parameter dari command JAVA. args merupakan array argumen yang dapat ditambahkan pada saat menggunakan command JAVA untuk menjalankan program JAVA.

c. Perintah untuk menampilkan "Halo jakarta" pada layar monitor.

### Penambahan komentar

Untuk membantu mengingat arti (tujuan) penulisan serangkaian kode yang telah ditulis, biasanya kita memberikan beberapa komentar pada program yang kita buat. Pada JAVA, untuk membuat komentar dapat dilakukan dengan cara:

- Komentar dengan menggunakan tanda //. Tanda ini digunakan untuk memberikan komentar dalam satu baris. Tanda // diletakkan pada awal baris.
- Komentar dengan menggunakan tanda /\* dan \*/. Tanda ini digunakan untuk memberikan komentar yang ditulis dalam beberapa kalimat. Tanda /\* diletakkan pada awal kalimat, dan tanda \*/ ditulis di akhir kalimat.

### 3. Tugas

1. Buatlah sebuah program yang mencetak data berikut :

Nama : Eka  
NIM : 2104.81.001  
Alamat : Jl. Raya Jendral sudirman No.20 Tangerang  
No.Telp : 0324-6575757  
Prodi : Teknik Informatika

Data bisa diganti dengan data anda



## MODUL 2

### Tipe Data, Variabel dan Operator

#### 1. Tujuan Pembelajaran:

- Praktikan dapat membuat variabel dengan benar.
- Praktikan mampu menggunakan berbagai tipe data dalam berbagai kepentingan.
- Praktikan mampu menggunakan berbagai operator dan mengimplementasikannya dalam pemrograman.

#### 2. Teori Singkat

##### Tipe Data

Di dalam pemrograman Java, kita bisa mengklasifikasikan tipe data primitif menjadi beberapa tipe data, yaitu :

- Bertipe Integer terdapat 4 (empat) Tipe Data.
- Bertipe Floating Point sebanyak 2 (dua) Tipe Data
- Satu Tipe Data berjenis Character
- Satu Tipe Data berjenis Boolean yaitu tipe untuk nilai logika.

Berikut kita bahas secara singkat dan padat mengenai keempat kategori tipe data diatas.

##### Java Integer

Tipe data integer digunakan untuk operasi data bilangan bulat dan perhitungan aritmatika. Berikut keempat tipe data yang tercakup kedalam kategori integer.

Nama Tipe Data	Keyword	Ukuran	Jangkauan Nilai
Byte-Length Integer	byte	8 bit	-128 s.d 127
Short Integer	short	16 bit	-32768 s.d 32767
Integer	Int	32 bit	-2147483648 s.d 2147483647
Long Integer	long	64 bit	-223372036854775808 s.d 223372036854775807

##### Java Floating Point

Floating-point dasarnya digunakan ketika kita mempunyai situasi dimana mendapatkan hasil atau output dalam bentuk desimal dan seluruh angka yang tidak disebutkan dalam tipe data integers. Tipe data yang termasuk kategori ini yaitu float dan double.

Nama Tipe Data	Keyword	Ukuran	Jangkauan Nilai
Single-precision Floating Point	float	32 bit	Presisi 6-7 bit -3.4E38 s.d 3.4E38
Double-precision Floating Point	double	64 bit	Presisi 14-15 bit -1.7E308 s.d 1.7E308

##### Java Character

Tipe data Character digunakan untuk mendefinisikan sebuah karakter yang merupakan simbol dalam karakter Set, seperti huruf dan angka. Keyword tipe data Character ini yaitu char, dengan ukuran 16 bit

##### Java Boolean

Tipe data boolean digunakan untuk menyebut variabel yang hanya mengandung nilai-nilai True atau False, dengan ukuran 1 bit.

Selain tipe data Primitif yang dimiliki oleh Java. Java memiliki tipe data class Object. Tipe data class Object yang sering digunakan yaitu String. String disediakan untuk menampung sejumlah character

##### Variabel

Variabel adalah suatu tempat menampung data atau konstanta dimemori yang mempunyai nilai atau data yang dapat berubah-ubah selama proses program. Dalam pemberian nama variabel, mempunyai ketentuan-ketentuan antara lain:

- Tidak boleh ada spasi (cth : gaji bersih) dan dapat menggunakan tanda garis bawah ( \_ ) sebagai penghubung (cth : gaji\_bersih).
- Tidak boleh diawali oleh angka dan menggunakan operator aritmatika.

##### Deklarasi Variabel

Deklarasi Variabel adalah proses memperkenalkan variabel kepada java dan pendeklarasian tersebut bersifat mutlak karena jika tidak diperkenalkan terlebih dulu maka java tidak menerima variabel tersebut. Deklarasi Variabel ini meliputi tipe variabel, seperti: integer atau character dan nama variabel itu sendiri. Setiap kali pendeklarasian variabel harus diakhiri oleh tanda titik koma (;).

**Bentuk penulisannya :**

```
Tipe_data nama_variabel;
```

**Contoh Deklarasi :**

```
String nama_mahasiswa;
char grade;
float rata_rata ;
int nilai1, nilai2;
```

### Menempatkan Nilai kedalam Variabel

Setelah pendeklarasian Variabel dilaksanakan, selanjutnya variabel tadi bisa anda masukan nilai kedalam variabel. Berikut cara yang mudah untuk menempatkan nilai kedalam variabel.

**Bentuk penulisannya :**

```
Nama_variabel = nilai;
```

**Contoh Penempatan Nilai kedalam Variabel :**

```
nama_mahasiswa = "Irvan Y. Ardiansyah";
grade = 'A';
rata_rata = 95.75;
nilai1 = 90; nilai2 = 95;
```

Java bisa juga memperbolehkan memberikan nilai yang sama ke beberapa nama variabel yang berbeda. Seperti contoh dibawah ini:

```
a = c = d = 7;
```

Pada contoh diatas variabel a, c, dan d masing-masing berisi nilai 7.

## Operator

Operator adalah simbol atau karakter yang biasa dilibatkan dalam program untuk melakukan sesuatu operasi atau manipulasi, seperti penjumlahan, pengurangan dan lain-lain.

### Operator Aritmatika

Operasi pada Java Contoh	Operator	Ekspresi
Perkalian	*	4 * 5
Pembagian	/	7 / 4
Sisa Pembagian	%	5 % 2
Penjumlahan	+	7 + 3
Pengurangan	-	6 - 4

### Operator Pemberi Nilai

Operasi pada Java	Operator Pemberi Nilai	Contoh Ekspresi	Penggunaan Operator Pemberi Nilai
Perkalian	*=	A = A * 5	A *= 5
Pembagian	/=	A = A / 5	A /= 5
Sisa Pembagian	%=	A = A % 2	A %= 2
Penjumlahan	+=	A = A + 1	A += 1
Pengurangan	-=	A = A - 4	A -= 4

### Operator Penambah dan Pengurang

Operator	Keterangan
++	Penambahan
--	Pengurangan

## Operator Pembandingan

Operator	Keterangan
==	Sama Dengan ( bukan pemberi nilai )
!=	Tidak Sama dengan
>	Lebih Dari
<	Kurang Dari
>=	Lebih Dari sama dengan
<=	Kurang Dari sama dengan

## Operator Logika

Operator	Keterangan
&&	Operator Logika AND
	Operator Logika OR
!	Operator Logika NOT

### Operator Logika AND

Operator logika AND digunakan untuk menghubungkan dua atau lebih ekspresi relasi, akan dianggap BENAR, bila semua ekspresi relasi yang dihubungkan bernilai BENAR.

### Operator Logika OR

Operator logika OR digunakan untuk menghubungkan dua atau lebih ekspresi relasi, akan dianggap BENAR, bila salah satu ekspresi relasi yang dihubungkan bernilai BENAR dan bila semua ekspresi relasi yang dihubungkan bernilai SALAH, maka akan bernilai SALAH.

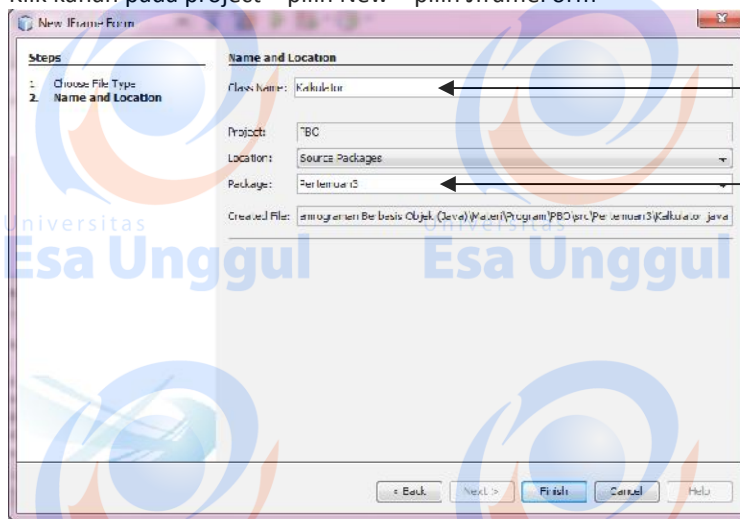
### Operator Logika NOT

Operator logika NOT akan memberikan nilai kebalikkan dari ekspresi yang disebutkan. Jika nilai yang disebutkan bernilai BENAR maka akan menghasilkan nilai SALAH, begitu pula sebaliknya.



# Java GUI: Kalkulator Sederhana

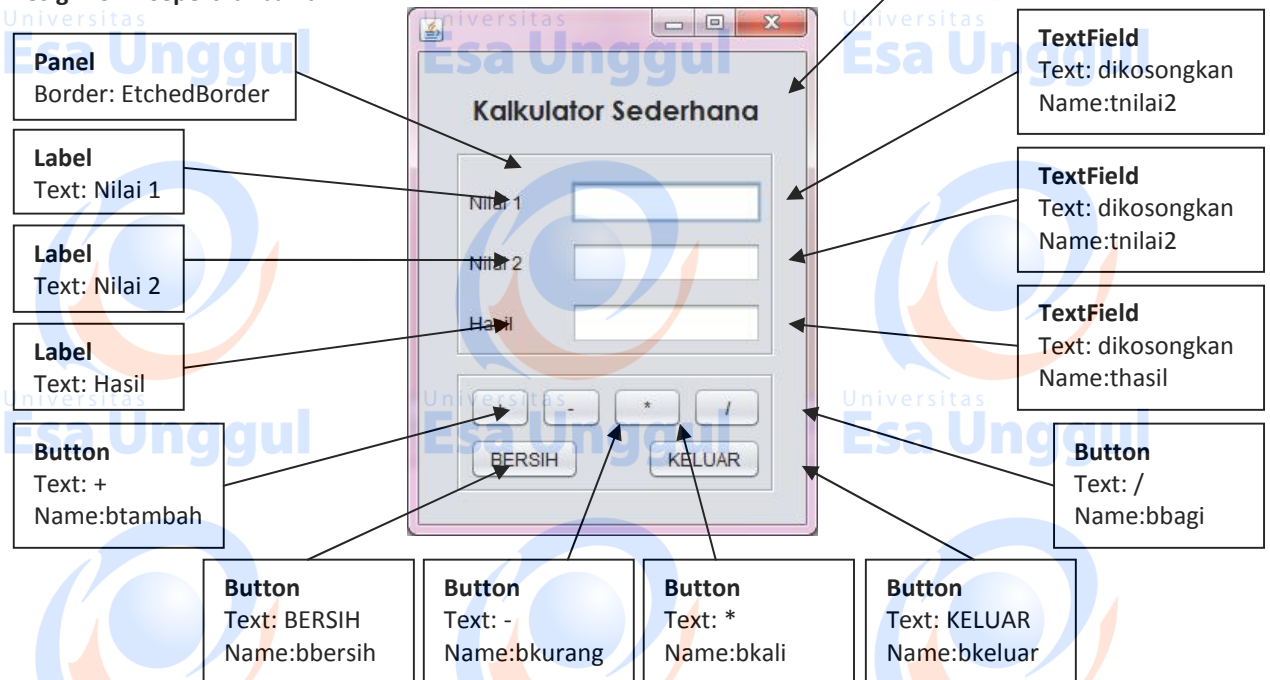
Klik kanan pada project – pilih New – pilih JFrameForm



Kalkulator

Pertemuan3

Design form seperti di bawah ini:



Listing:

```
private void btambahActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int a=Integer.parseInt(tnilai1.getText());
    int b=Integer.parseInt(tnilai2.getText());
    int hasil=a+b;
    thasil.setText(Integer.toString(hasil));
}
```

```
private void bkurangActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int a=Integer.parseInt(tnilai1.getText());
    int b=Integer.parseInt(tnilai2.getText());
    int hasil=a-b;
    thasil.setText(Integer.toString(hasil));
}
```

```

}
private void bkaliActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int a=Integer.parseInt(tnilai1.getText());
    int b=Integer.parseInt(tnilai2.getText());
    int hasil=a*b;
    thasil.setText(Integer.toString(hasil));
}

private void bbagiActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int a=Integer.parseInt(tnilai1.getText());
    int b=Integer.parseInt(tnilai2.getText());
    int hasil=a/b;
    thasil.setText(Integer.toString(hasil));
}

private void bbersihActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    tnilai1.setText("");
    tnilai2.setText("");
    thasil.setText("");
}

private void bkeluarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int a=JOptionPane.showConfirmDialog(null,"Yakin Mau Keluar","info",JOptionPane.YES_NO_OPTION);
    if(a==JOptionPane.YES_OPTION)
        dispose();
}

```

## 2.1. Variabel

Variabel adalah nama dari suatu lokasi di memori yang digunakan untuk menyimpan data sementara. Variabel diberinama tertentu yang menunjukkan domain yang diwakilinya. Dalam memberikan nama variabel, ada beberapa ketentuan yang harus diperhatikan yaitu:

- Panjang karakter nama variabel tidak dibatasi.
- Nama variabel diawali dengan huruf, tanda garis bawah (*underscore*) atau tanda dolar (\$). Selanjutnya dapat diikuti oleh karakter lain, selain operator (\*, -, +).
- Bersifat case sensitive (membedakan antara huruf kapital dan huruf kecil).
- Tidak diperbolehkan menggunakan kata-kata kunci yang digunakan pada java, seperti: if, for, while, dll.

Data yang tersimpan dalam variabel memiliki tipe tertentu. Sebelum digunakan dalam aplikasi, suatu variabel harus dideklarasikan terlebih dahulu.

<b>Syntax: [tipe_data] [nama_variabel]</b>
--------------------------------------------

Beberapa tipe data yang dapat digunakan akan dibahas pada subbagian berikut.

## 2.2. Tipe Data

### 2.2.1 Karakter

Karakter tunggal, diberikan dengan tipe data char. Data yang memiliki tipe data ini ditulis dengan diapit tanda petik tunggal, seperti: 'A', 'S', '?', dll. Char berbeda dengan String. String adalah kumpulan dari beberapa karakter. Data yang memiliki tipe data ini ditulis dengan diapit tanda petik ganda.

Contoh:

```
String nama, golDarah;  
nama = "arwan";  
golDarah = '0';
```

### 2.2.2 Integer

Tipe data integer merupakan bilangan bulat (positif, nol, atau negatif).

Contoh:

```
int x1, x2, Jumlah;  
Jumlah = x1 + x2;
```

Selain int, bilangan integer juga dapat memiliki tipe data byte, short atau long yang masing-masing dibedakan oleh panjang memori yang ditempatinya.

- Byte menempati lokasi sebesar 1 byte.
- Short menempati lokasi memori sebesar 2 byte
- Int menempati lokasi memori sebesar 4 byte
- Long menempati lokasi memori sebesar 8 byte

### 2.2.3 Floating Point

Untuk merepresentasikan data pecahan (bukan integer) dapat digunakan dua macam tipe data, yaitu float atau double. Tipe data float memiliki panjang lokasi penyimpanan sebesar 4 byte sedangkan double sepanjang 8 byte.

Contoh:

```
int x1, x2;  
float Rata2;  
double PanjangJalur;  
Rata2 = (x1 + x2)/2;  
PanjangJalur = 1.5E3;
```

### 2.2.4 Boolean

Tipe data boolean hanya memiliki dua kemungkinan nilai yaitu benar atau salah.

Contoh:

```
boolean Selesai;  
Selesai = true;
```

## 2.3. Operator

### 2.3.1 Operator Aritmetik

Operator-operator aritmetik di Java seperti pada umumnya terdiri dari: penjumlahan (+), pengurangan (-), pembagian (/), perkalian (\*), dan modulo (%). Kode program pada Gambar 2.1 berikut menunjukkan operasi aritmetik untuk nilai A=100 dan B=30.

Apabila program tersebut dieksekusi, maka hasilnya sepertiterlihat pada **Gambar 2.2**.

```

1  /* Operator aritmetik
2  */
3  package operator_aritmetik;
4
5  /**
6   * @author Cicie
7   */
8  public class Main {
9
10     public static void main(String[] args) {
11         int A=100, B=30;           // nilai variabel A dan B
12
13         int jumlah = A+B;         // operasi penjumlahan
14         int kurang = A-B;        // operasi pengurangan
15         int kali = A*B;          // operasi perkalian
16         float bagi = (float)A/B; // operasi pembagian
17         int modulo = A%B;        // modulo
18
19         // tampilkan hasil
20         System.out.println("Penjumlahan: "+A+" + "+B+" = "+jumlah);
21         System.out.println("Pengurangan: "+A+" - "+B+" = "+kurang);
22         System.out.println("Perkalian: "+A+" * "+B+" = "+kali);
23         System.out.println("Pembagian: "+A+" / "+B+" = "+bagi);
24         System.out.println("Modulo: "+A+" mod "+B+" = "+modulo);
25     }
26 }

```

**Gambar 2.1** Contoh program dengan menggunakan operatoraritmetik.

```

Output - operator_aritmetik (run)
init:
deps-jar:
Compiling 1 source file to D:\NetBeansProjects\operator_aritmetik\build\classes
compile:
run:
Penjumlahan: 100 + 30 = 130
Pengurangan: 100 - 30 = 70
Perkalian: 100 * 30 = 3000
Pembagian: 100 / 30 = 3.3333333
Modulo: 100 mod 30 = 10
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Gambar 2.2** Output program operator\_aritmetik.

Tunjukkanlah hasil output yang dihasilkan apabila pernyataanpada baris ke-16 diganti dengan:

```
float bagi = A/B;
```

Di samping operator-operator dasar tersebut, ada beberapa cara singkat untuk menuliskan operator aritmetika, seperti:

- $x = x + 1$ , dapat disingkat menjadi  $x++$
- $x = x - 1$ , dapat disingkat menjadi  $x--$
- $x = x + 3$ , dapat disingkat menjadi  $x+=3$
- $x = x - 5$ , dapat disingkat menjadi  $x-=3$

Coba tulislah kode program di bawah ini dalam project bernama , pernyataan\_increment. Selanjutnya, eksekusi project tersebut, amatilah hasilnya.

```

int A=10, B=10, C=0, D=0; // nilai awal
int step=3; // kenaikan
A++; // A = A+1
B--; // B = B-1
C += step; // C = C+step

```



```

D -= step; // D = D-step
// tampilkan hasil
System.out.println("A = A+1 -> "+A);
System.out.println("B = B-1 -> "+B);
System.out.println("C = C+step -> "+C);
System.out.println("D = D-step -> "+D);

```

### 2.3.2 Operator Relasi

Operator relasi digunakan dalam ekspresi boolean yang akan menghasilkan nilai boolean guna menentukan langkah eksekusi blok pernyataan tertentu. Beberapa operator logika dapat dilihat pada **Tabel 2.1**.

Kode program pada **Gambar 2.3** berikut menunjukkan operasi relasi untuk nilai A=100 dan B=30.

**Tabel 2.1 Operator Relasi**

Symbol	Keterangan
>	Lebih Dari
<	Kurang Dari
>=	Lebih Dari atau Sama Dengan
<=	Kurang Dari atau Sama Dengan
==	Sama Dengan
!=	Tidak Sama Dengan

```

1  /* Operator relasi
2  */
3
4  package operator_relasi;
5
6  /**
7   * @author Cicie
8   */
9  public class Main {
10
11     public static void main(String[] args) {
12         int A=100, B=30; // nilai variabel A dan B
13
14         boolean lb = A > B; // lebih besar
15         boolean lk = A < B; // lebih kecil
16         boolean lbs = A >= B; // lebih dari atau sama
17         boolean lks = A <= B; // kurang dari atau sama
18         boolean sm = A == B; // sama
19         boolean tsm = A != B; // tidak sama
20
21         // Tampilan hasil
22         System.out.println(A+" > "+B+" => "+lb);
23         System.out.println(A+" < "+B+" => "+lk);
24         System.out.println(A+" >= "+B+" => "+lbs);
25         System.out.println(A+" <= "+B+" => "+lks);
26         System.out.println(A+" == "+B+" => "+sm);
27         System.out.println(A+" != "+B+" => "+tsm);
28     }
29 }

```

**Gambar 2.3** Contoh program dengan menggunakan operator relasi.

Apabila program tersebut dieksekusi, maka hasilnya seperti terlihat pada **Gambar 2.4**.



```

:Output - operator_relasi (run)
init:
deps-jar:
Created dir: D:\NetBeansProjects\operator_relasi\build\classes
Compiling 1 source file to D:\NetBeansProjects\operator_relasi\build\classes
compile:
run:
100 > 30 => true
100 < 30 => false
100 >= 30 => true
100 <= 30 => false
100 = 30 => false
100 != 30 => true
BUILD SUCCESSFUL (total time: 0 seconds)

```

Gambar 2.4 Output program operator\_relasi.

### 2.3.3 Operator Logika

Operator logika digunakan juga dalam ekspresi boolean yang akan menghasilkan nilai boolean guna menentukan langkah eksekusi blok pernyataan tertentu. Beberapa operator logika seperti AND (&&), OR (||) atau NOT (!).

Kode program pada Gambar 2.5 menunjukkan operasi logika untuk A=true dan B=false.

```

1  /* Operator logika
2  */
3
4  package operator_logika;
5
6  /**
7   * @author Cicie
8   */
9  public class Main {
10
11     public static void main(String[] args) {
12         boolean A=true, B=false; // nilai boolean A dan B
13
14         boolean o_and = A && B; // A and B
15         boolean o_or = A || B; // A or B
16         boolean o_notA = !A; // not A
17         boolean o_notB = !B; // not B
18
19         // Tampilan hasil
20         System.out.println(A+" AND "+B+" => "+o_and);
21         System.out.println(A+" OR "+B+" => "+o_or);
22         System.out.println("NOT "+A+" => "+o_notA);
23         System.out.println("NOT "+B+" => "+o_notB);
24     }
25 }
26

```

Gambar 2.5 Contoh program dengan menggunakan operator logika.

Apabila program tersebut dieksekusi, maka hasilnya seperti terlihat pada Gambar 2.6.

```

:Output - operator_logika (run)
init:
deps-jar:
Created dir: D:\NetBeansProjects\operator_logika\build\classes
Compiling 1 source file to D:\NetBeansProjects\operator_logika\build\classes
compile:
run:
true AND false => false
true OR false => true
NOT true => false
NOT false => true
BUILD SUCCESSFUL (total time: 0 seconds)

```

Gambar 2.6 Output program operator\_logika.

### 3. Latihan

Lakukan ujicoba pada program diatas sebagai bentuk latihan kemudian lakukan analisa

### 4. Tugas

- a. Buatlah program kalkulator serhana yang mengimplementasikan beberapa operator yang sudah dijelaskan.
- b. Buatlah sebuah program untuk yang menampilkan hasil operasi dasar logika AND OR dan NOT
- c. Buatlah sebuah program yang melibatkan proses casting nilai pada sebuah variabel.



## MODUL 3 Kondisi

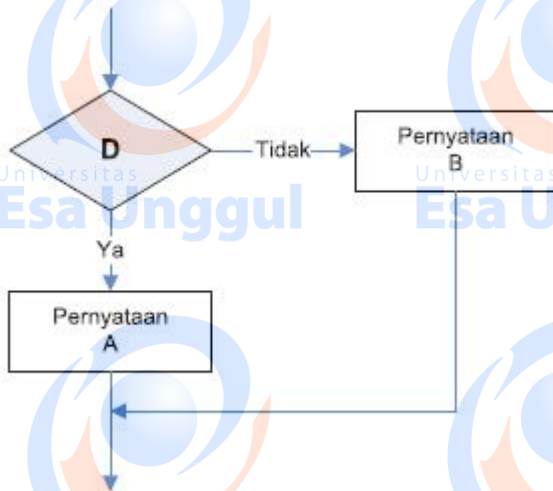
### 1. Tujuan Pembelajaran:

- Praktikan mengenal beberapa perintah untuk seleksi kondisi
- Praktikan mampu menggunakan berbagai conditional statement dalam berbagai kebutuhan.

### 2. Teori Singkat

#### Operator Kondisi Bahasa Java

Dalam pemrograman seringkali dibutuhkan eksekusi blok pernyataan jika dipenuhi kondisi tertentu. Kondisi yang diberikan dinyatakan dengan ekspresi boolean. Pada **Gambar 3.1** terlihat bahwa blok pernyataan A akan dieksekusi jika kondisi D bernilai benar, sebaliknya blok pernyataan B akan dieksekusi jika kondisi D bernilai salah.



**Gambar 3.1** Diagram blok seleksi kondisi

#### Pengkondisian dengan if

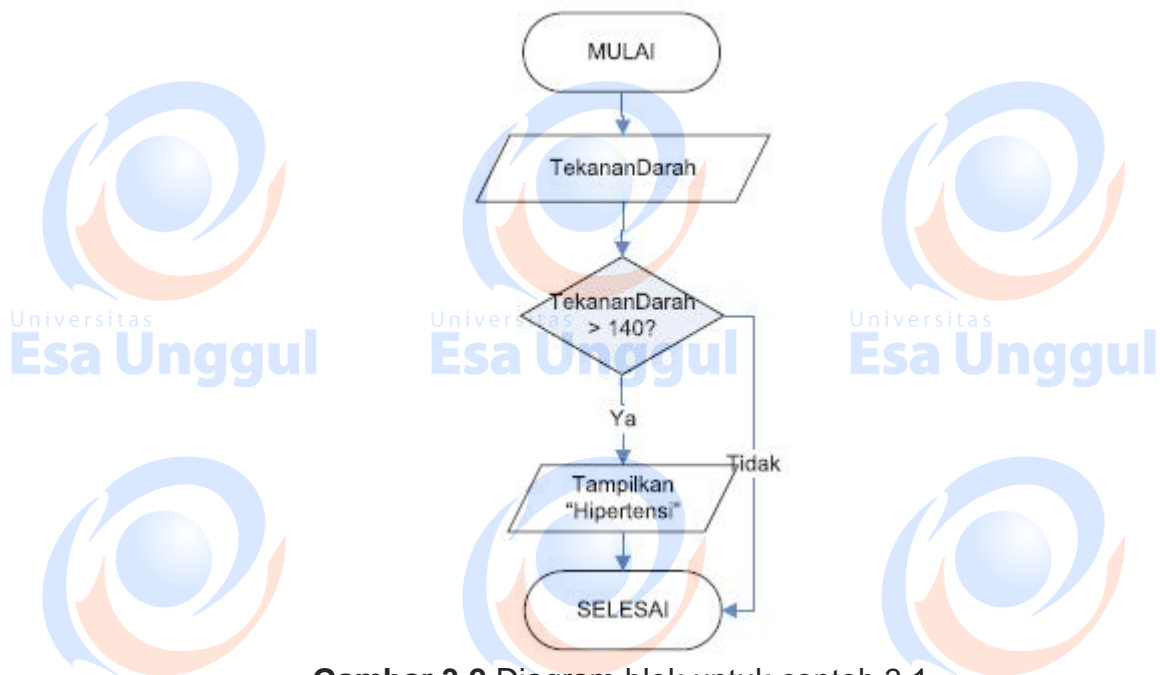
Pengkondisian dengan if digunakan untuk mengeksekusi blok pernyataan, dimana yang dibutuhkan hanyalah kondisi benarsaja.

Sintaks:

```
if (<ekspresi_boolean>)  
{  
<blok pernyataan>  
}
```

Contoh 3.1:

Lihat *flowchart* pada **Gambar 3.2** berikut. *Flowchart* ini digunakan untuk memberikan informasi kepada pengguna tentang status tekanan darahnya. Seseorang dikatakan hipertensi jika tekanan darah sistolik lebih dari 140 mmHg.



Gambar 3.2 Diagram blok untuk contoh 3.1

## Pernyataan IF

Pernyataan if mempunyai pengertian, “Jika kondisi bernilai benar, maka perintah akan dikerjakan dan jika tidak memenuhi syarat maka akan diabaikan”. Penulisan kondisi harus didalam tanda kurung dan berupa ekspresi relasi dan penulisan pernyataan dapat berupa sebuah pernyataan tunggal, pernyataan majemuk atau pernyataan kosong. Jika pemakaian if diikuti dengan pernyataan majemuk, bentuk penulisannya sebagai berikut:

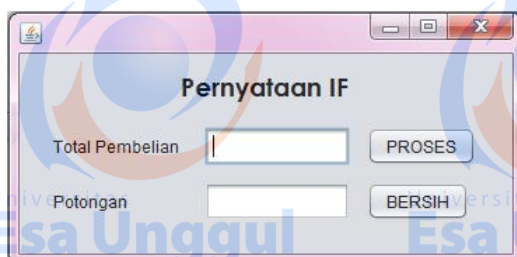
### 1 Pernyataan

```
if (kondisi)
    pernyataan;
```

### 2 Pernyataan/lebih

```
if (kondisi)
{
    pernyataan;
    .....
}
```

Diapit kurung kurawal jika lebih dari 1



### Contoh:

Menentukan besarnya potongan dari pembelian barang yang diberikan seorang pembeli, dengan kriteria:

- Tidak ada potongan jika total pembelian kurang dari Rp. 50.000,-
- Jika total pembelian lebih dari atau sama dengan Rp. 50.000,- potongan yang diterima sebesar 20% dari total pembelian

### Listing:

```
private void bprosesActionPerformed(java.awt.event.ActionEvent evt) {
    int total=Integer.parseInt(tttotal.getText());
    if(total>=50000){
        tpotongan.setText(Double.toString(0.2*total));
    }
}
private void bbersihActionPerformed(java.awt.event.ActionEvent evt) {
    tttotal.setText("");
}
```

```

tpotongan.setText("");
}

```

Kode program untuk permasalahan pada contoh 3.1 tersebut dapat dilihat pada **Gambar 3.3**.

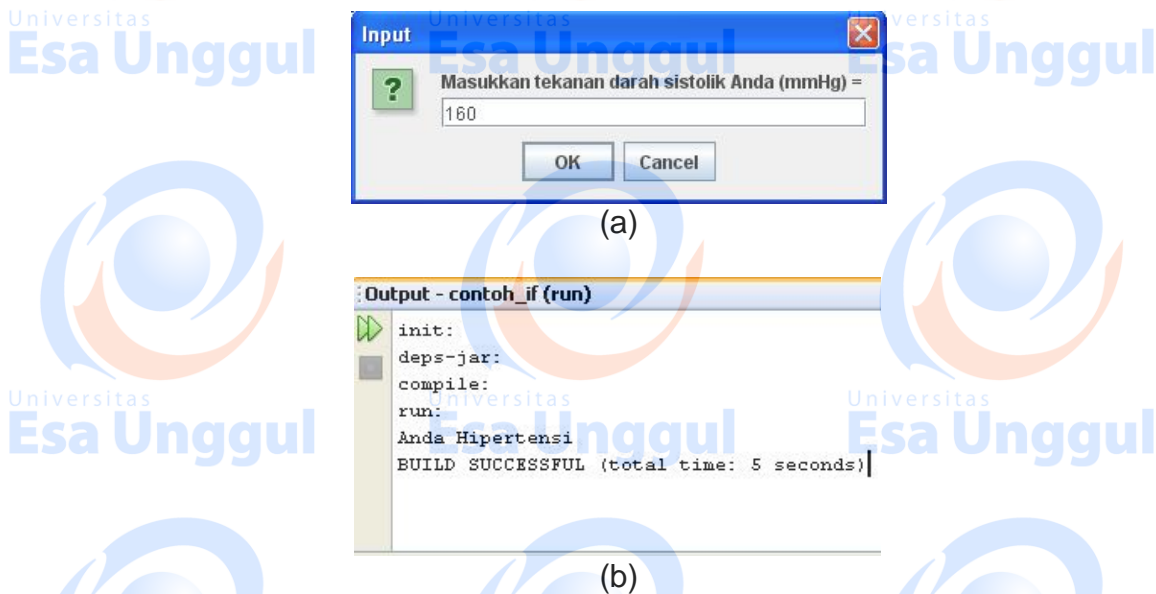
```

1  /* Contoh program if
2  */
3  package contoh_if;
4
5  import javax.swing.*;
6  public class Main {
7      public static void main(String[] args) {
8          String data_ID = JOptionPane.showInputDialog(" Masukkan tekanan darah "
9              + "sistolik Anda (mmHg)");
10         int TekananDarah = Integer.parseInt(data_ID);
11         if (TekananDarah > 140) {
12             System.out.println("Anda Hipertensi");
13         }
14     }
15 }

```

**Gambar 3.3** Kode program untuk contoh 3.1.

Apabila program tersebut dieksekusi, maka hasilnya sepertipada **Gambar 3.4**.



**Gambar 3.4** Output yang dihasilkan oleh program pada contoh 3.1.

### Pengkondisian dengan if-else

Pengkondisian dengan if-else digunakan untuk mengeksekusi blok pernyataan A jika suatu kondisi bernilai benar, dan sebaliknya akan mengeksekusi blok pernyataan B jika suatu kondisi bernilai salah.

Sintaks:

```

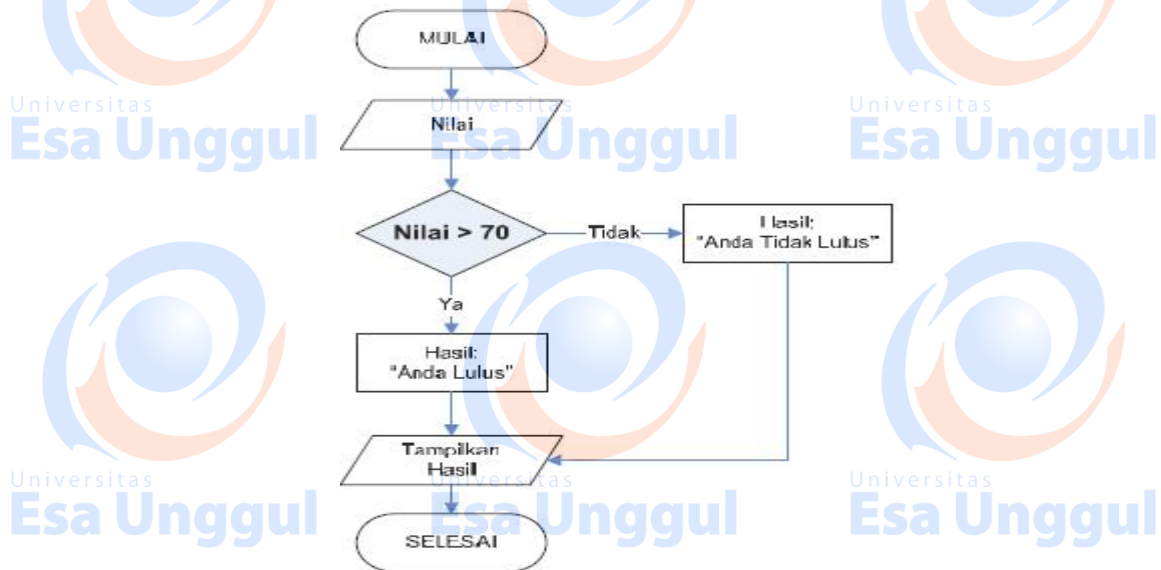
if (<ekspresi_boolean>
{
<blok pernyataan A>
}
else
{
<blok pernyataan B>
}

```



Contoh 3.2:

Lihat *flowchart* pada **Gambar 3.5**. *Flowchart* ini digunakan untuk menentukan kelulusan berdasarkan nilai yang diberikan. Seseorang dikatakan lulus jika nilai yang diperoleh lebih dari 70, sebaliknya dia dikatakan tidak lulus.



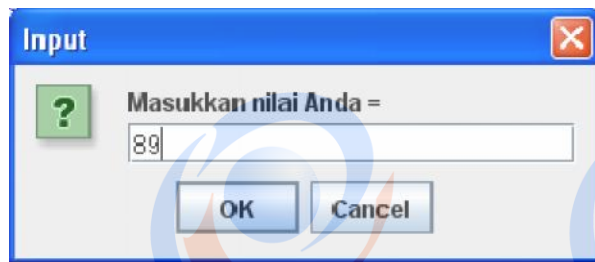
**Gambar 3.5** Diagram blok untuk contoh 3.2

Kode program untuk permasalahan pada contoh 3.2 tersebut dapat dilihat pada **Gambar 3.6**.

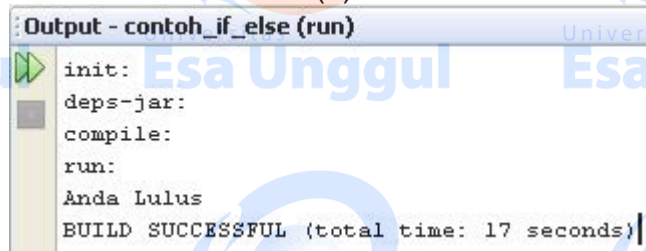
```
1  /* contoh pemakaian if-else
2  */
3
4  package contoh_if_else;
5
6  import javax.swing.*;
7
8  public class Main {
9
10     public static void main(String[] args) {
11         String Hasil = "";
12         String data_nilai = JOptionPane.showInputDialog("Masukkan nilai Anda = ");
13         float nilai = Float.parseFloat(data_nilai);
14         if (nilai > 70.0f)
15         {
16             Hasil="Anda Lulus";
17         }
18         else
19         {
20             Hasil="Anda Tidak Lulus";
21         }
22         System.out.println(Hasil);
23     }
24 }
25
```

**Gambar 3.6** Kode program untuk contoh 3.2.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 3.7**.



(a)



(b)

**Gambar 3.7** Output yang dihasilkan oleh program pada contoh 3.2

## Pernyataan IF-ELSE

Pernyataan if mempunyai pengertian, "Jika kondisi bernilai benar, maka perintah-1 akan dikerjakan dan jika tidak memenuhi syarat maka akan mengerjakan perintah-2".

### 1 Pernyataan

```
if (kondisi)
pernyataan-1;
else
pernyataan-1;
```

### 2 Pernyataan/lebih

```
if (kondisi)
{
perintah-1;
...
}
else
{
perintah-2;
...
}
```

Diapit kurung kurawal jika lebih dari 1

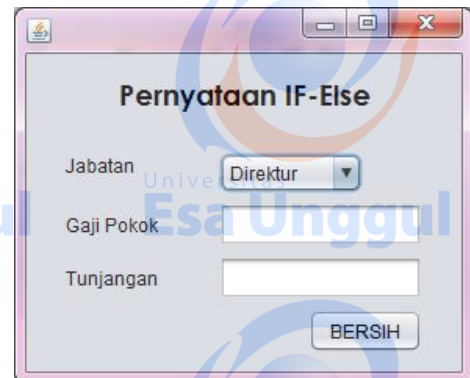
### Contoh:

Menentukan gaji pokok dan tunjangan dgn kriteria:

- Direktur, Gaji pokok Rp.10.000.000 dan Tunjangan Rp.1.000.000
- Manager, Gaji pokok Rp.7.000.000 dan Tunjangan Rp.750.000
- Supervisor, Gaji pokok Rp.5.000.000 dan Tunjangan Rp.500.000

### Listing:

```
private void cbjabatanActionPerformed(java.awt.event.ActionEvent evt) {
String jabatan=(String)cbjabatan.getSelectedItem();
if (jabatan.equals("Direktur")){
tgaji.setText("10.000.000");
ttunjangan.setText("1.000.000");
}
else if (jabatan.equals("Manager")){
tgaji.setText("7.000.000");
ttunjangan.setText("750.000");
}
else{
tgaji.setText("5.000.000");
ttunjangan.setText("500.000");
}
}
private void bbersihActionPerformed(java.awt.event.ActionEvent evt) {
tgaji.setText("");
}
```



```

    ttunjangan.setText("");
}

```

## Pernyataan IF-ELSE

Nested if merupakan pernyataan if berada didalam pernyataan if yang lainnya. Bentuk penulisan pernyataan Nested if adalah:

```

if(syarat)
    if(syarat)
        ... perintah;
    else
        ... perintah;
else
    if(syarat)
        ... perintah;
    else
        ... perintah;

```



### Contoh:

Menentukan harga berdasarkan merk dan satuan, dgn kriteria:

- Harga Aqua satuan gelas Rp.500
  - Harga Aqua satuan botol Rp.3.000
  - Harga VIT satuan gelas Rp.450
  - Harga VIT satuan botol Rp.2.800
- Pilih merk lalu pilih satuan maka harga tampil.

### Listing:

Membuat prosedur untuk menentukan harga, letakkan prosedur di bawah listing ini:

```

public NestedIf() {
    initComponents();
}

```

### Prosedur harga:

```

private void harga(){
    String merk=(String)lmerk.getSelectedValue();
    if (merk.equals("AQUA"))
        if (rgelas.isSelected()==true)
            tharga.setText("500");
        else
            tharga.setText("3000");
    else
        if (rgelas.isSelected()==true)
            tharga.setText("450");
        else
            tharga.setText("2800");
}

private void rgelasActionPerformed(java.awt.event.ActionEvent evt) {
    rbotol.setSelected(false);
    harga();
}

private void rbotolActionPerformed(java.awt.event.ActionEvent evt) {
    rgelas.setSelected(false);
    harga();
}

private void bbersihActionPerformed(java.awt.event.ActionEvent evt) {
    rgelas.setSelected(false);
    rbotol.setSelected(false);
    tharga.setText("");
}

```

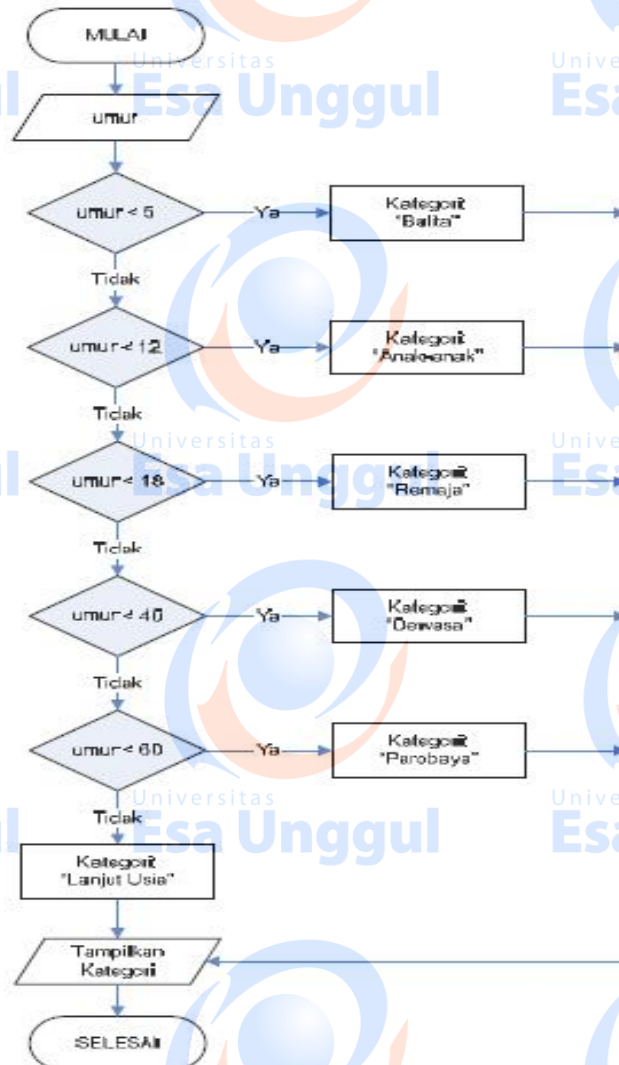
Contoh 3.3

Diketahui pembagian umur sebagaimana terlihat pada Tabel 3.1.

Tabel 3.1 Pembagian umur

Umur (Tahun)	Kategori
Umur <5	Balita
5 umur < 12	Anak-anak
12 umur < 18	Remaja
18 umur < 40	Dewasa
40 umur < 60	Parobaya
umur 60	Lanjut usia

Gambar 3.8 menunjukkan flowchart untuk permasalahan tersebut.



Gambar 3.8 Diagram blok untuk contoh 3.3.

Kode program untuk permasalahan pada contoh 3.3 tersebut dapat dilihat pada Gambar 3.9.

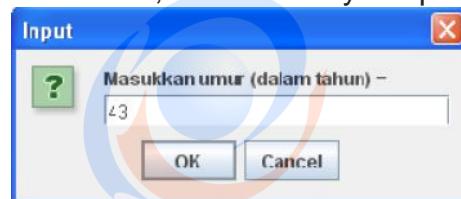
```

1
2 import javax.swing.*;
3 public class Contoh_if_else2 {
4     public static void main(String[] args) {
5         String Kategori;
6         String data_umur = JOptionPane.showInputDialog("Masukkan umur anda (dalam tahun)= ");
7         float umur = Float.parseFloat(data_umur);
8         if (umur < 5.0F){
9             Kategori="Balita";
10        }
11        else if (umur < 12.0F){
12            Kategori="Anak-anak";
13        }
14        else if (umur < 18.0F){
15            Kategori="Remaja";
16        }
17        else if (umur < 40.0F){
18            Kategori="Dewasa";
19        }
20        else if (umur < 60.0F){
21            Kategori="Paruh Baya";
22        }
23        else {
24            Kategori="Lanjut Usia";
25        }
26        System.out.println("Umur "+umur+" tahun termasuk dalam kategori "+Kategori);
27    }
28 }
29 }

```

**Gambar 3.9** Kode program untuk contoh 3.3.

Apabila program tersebut dieksekusi, maka hasilnya sepertipada **Gambar 3.10**.



(a)

```

Output - contoh_if_else2 (run)
init:
deps-jar:
compile:
run:
Umur 43.0 tahun termasuk kategori Parobaya
BUILD SUCCESSFUL (total time: 11 seconds)

```

(b)

**Gambar 3.10** Output yang dihasilkan oleh program pada contoh 3.3 .

### Pengkondisian dengan switch

Pengkondisian dengan switch digunakan apabila ada beberapapilihan dimana setiap pilihan akan mengeksekusi blokpernyataan yang berbeda.

Sintaks:

```

switch (<ekspresi_integer>)
{
case <nilai_variabel>: <blok pernyataan>
Break;
case <nilai_variabel>: <blok pernyataan>
Break;
case <nilai_variabel>: <blok pernyataan>
Break;
...
default: <blok pernyataan>
Break;

```

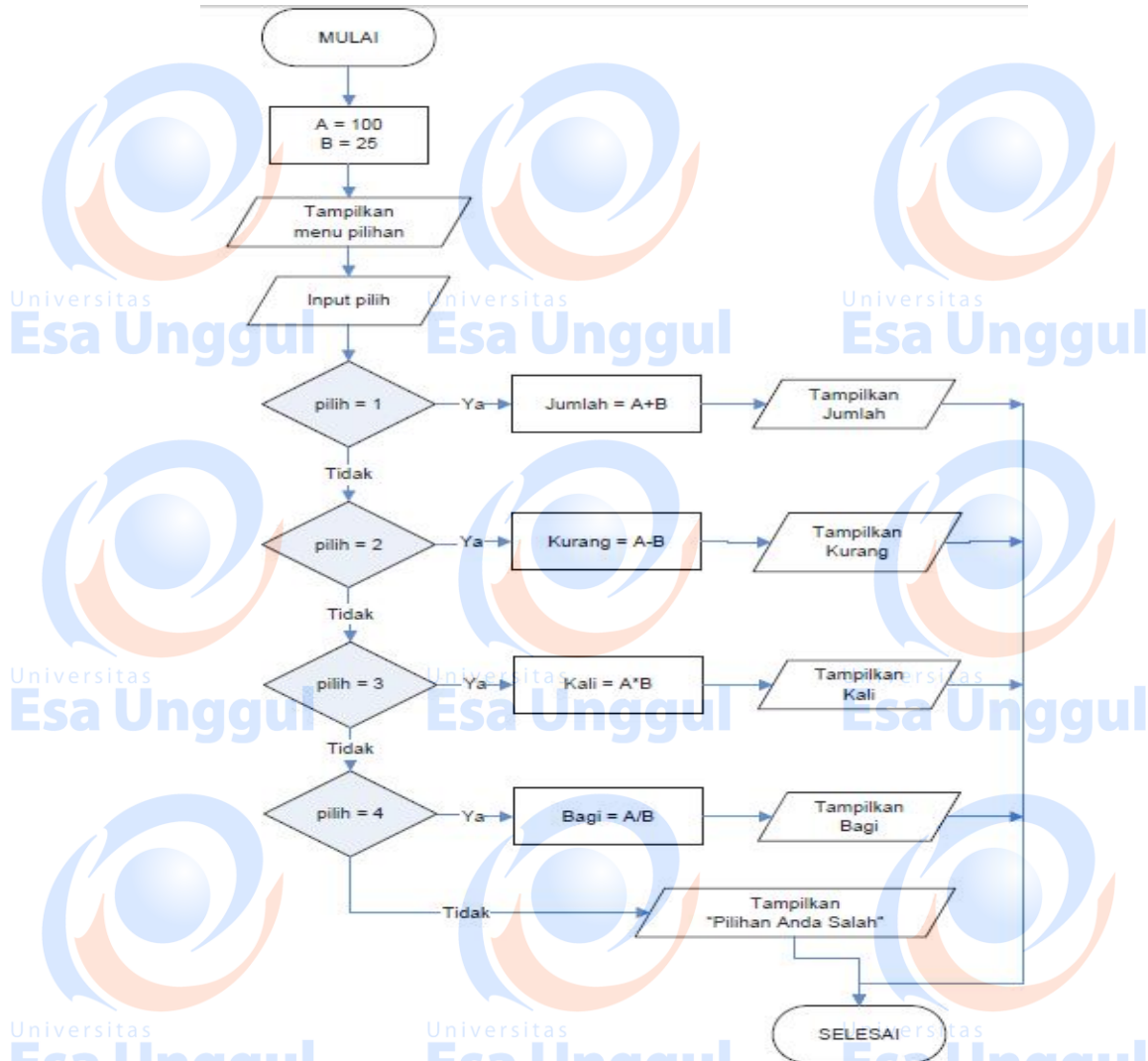


```
}  
}
```

*Ekspresi\_integer* yang dapat digunakan adalah byte, short, int, dan char. Char dapat digunakan karena char sebenarnya merupakan unicode yang bernilai antara 0 – 65535. Sedangkan tipe long tidak dapat digunakan untuk ekspresi *integer* ini.

### Contoh 3.4

Berikut akan dilakukan operasi aritmetik untuk dua bilangan integer A dan B, dengan A=100 dan B=25. Ada 4 operasi yang dapat dipilih, yaitu penjumlahan, pengurangan, perkalian dan pembagian. **Gambar 3.11** menunjukkan *flowchart* untuk permasalahan tersebut.



**Gambar 3.11** Diagram blok untuk contoh 3.4.

Kode program untuk permasalahan pada contoh 3.4 tersebut dapat dilihat pada **Gambar 3.12**.

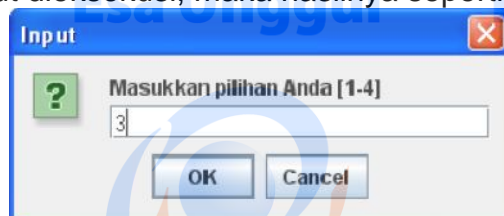
```

1 import javax.swing.*;
2 public class Contoh_switch {
3     public static void main(String[] args) {
4         int A = 100, B = 25;
5         System.out.println("Menu Pilihan :");
6         System.out.println("1. Penjumlahan");
7         System.out.println("2. Pengurangan");
8         System.out.println("3. perkalian");
9         System.out.println("4. pembagian");
10        String data_pilihan = JOptionPane.showInputDialog("naukan pilihan anda [1-4] :");
11        int pilihan = Integer.parseInt(data_pilihan);
12        switch (pilihan) {
13            case 1: {
14                int jumlah = A + B;
15                System.out.println("Penjumlahan " + A + " + " + B + " = " + jumlah);
16                break;
17            }
18            case 2: {
19                int kurang = A - B;
20                System.out.println("Pengurangan " + A + " - " + B + " = " + kurang);
21                break;
22            }
23            case 3: {
24                int kali = A * B;
25                System.out.println("Perkalian " + A + " * " + B + " = " + kali);
26                break;
27            }
28            case 4: {
29                float bagi = A / B;
30                System.out.println("Pembagian " + A + " : " + B + " = " + bagi);
31                break;
32            }
33            default:
34                System.out.println("Pilihan anda salah");
35                break;
36        }
37    }
38 }

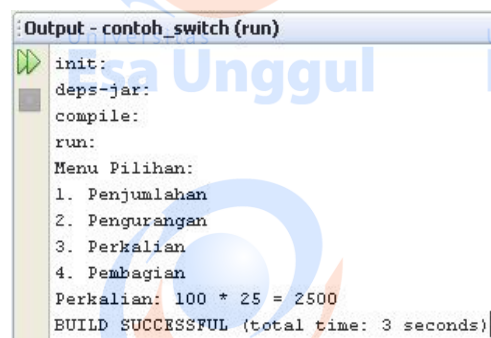
```

**Gambar 3.12** Kode program untuk contoh 3.4.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 3.13**.



(a)



(b)

**Gambar 3.13** Output yang dihasilkan oleh program pada contoh 3.4.

## Pernyataan SWITCH-CASE

Bentuk dari switch - case merupakan pernyataan yang dirancang khusus untuk menangani pengambilan keputusan yang melibatkan sejumlah atau banyak alternatif. Pernyataan switch - case ini memiliki kegunaan sama seperti if - else bertingkat, tetapi penggunaannya hanya untuk memeriksa data yang bertipe primitif integer saja. Bentuk penulisan perintah ini sebagai berikut:

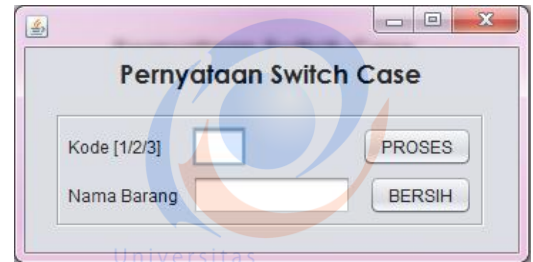
```
switch (ekspresi integer)
{
    case konstanta-1 :
        ... perintah;
        ... perintah;
        break;
    case konstanta-2 :
        ... perintah;
        ... perintah;
        break;
    .....
    .....
    default :
        ... perintah;
        ... perintah;
```

Setiap cabang akan dijalankan jika syarat nilai konstanta tersebut dipenuhi dan default akan dijalankan jika semua cabang di atasnya tidak terpenuhi. Pernyataan break menunjukkan bahwa perintah siap keluar dari switch. Jika pernyataan ini tidak ada, maka program akan diteruskan ke cabang - cabang yang lainnya.

Contoh:

Menentukan nama barang berdasarkan kode barang, dgn kriteria:

- kode 1, Buku
- kode 2, Pulpen
- kode 3, Pensil



Listing:

```
private void bprosesActionPerformed(java.awt.event.ActionEvent evt) {
    int kode=Integer.parseInt(tkode.getText());
    switch(kode){
        case 1:
            tnama.setText("Buku");
            break;
        case 2:
            tnama.setText("Pulpen");
            break;
        default:
            tnama.setText("Pensil");
            break;
    }
}
```

```
private void bbersihActionPerformed(java.awt.event.ActionEvent evt) {
    tkode.setText("");
    tnama.setText("");
    tkode.requestFocus();
}
```

### 3. Tugas

- Buatlah sebuah program konversi nilai angka ke nilai huruf dengan range berikut :
  - Nilai  $\geq 85$  and nilai  $\leq 100 = A$
  - Nilai  $\geq 70$  and nilai  $\leq 84 = B$
  - Nilai  $\geq 60$  and nilai  $< 70 = C$
  - Nilai  $\geq 50$  and nilai  $< 60 = D$  dan Nilai  $< 50 = E$
- Buatlah sebuah program untuk mengecek apakah bilangan yang diinput adalah bilangan positif, negatif atau nol
- Buatlah sebuah program untuk mengecek apakah bilangan yang dimasukkan adalah bilangan genap atau ganjil
- Buatlah sebuah program untuk konversi angka dengan menggunakan konsep struktur kontrol switch - case

## MODUL 4 Perulangan

### 1. Tujuan Pembelajaran:

- Praktikan mengenal beberapa perintah untuk melakukan perulangan
- Praktikan mampu menggunakan berbagai bentuk perulangan dalam berbagai kebutuhan.

### 2. Teori Singkat Perulangan

Adakalanya suatu blok pernyataan harus dieksekusi berulang kali tergantung pada kondisi tertentu. Untuk keperluan tersebut, seperti halnya bahasa pemrograman yang lainnya, JAVA menyediakan beberapa statement perulangan. Dengan menggunakan statement tersebut, suatu blok pernyataan tidak perlu ditulis berulang kali, namun cukup dengan memberikan ekspresi boolean terhadap suatu kondisi.

#### Perintah: for

Bentuk for digunakan untuk melakukan perulangan, dimana banyaknya perulangan telah diketahui sebelumnya. Pernyataan dengan for akan memiliki counter yang akan bergerak (naik atau turun) secara otomatis.

#### Sintaks:

```
for (<inisialisasi> ; <ekspresi_boolean>; <increment>)  
{  
<blok_pernyataan>  
}
```

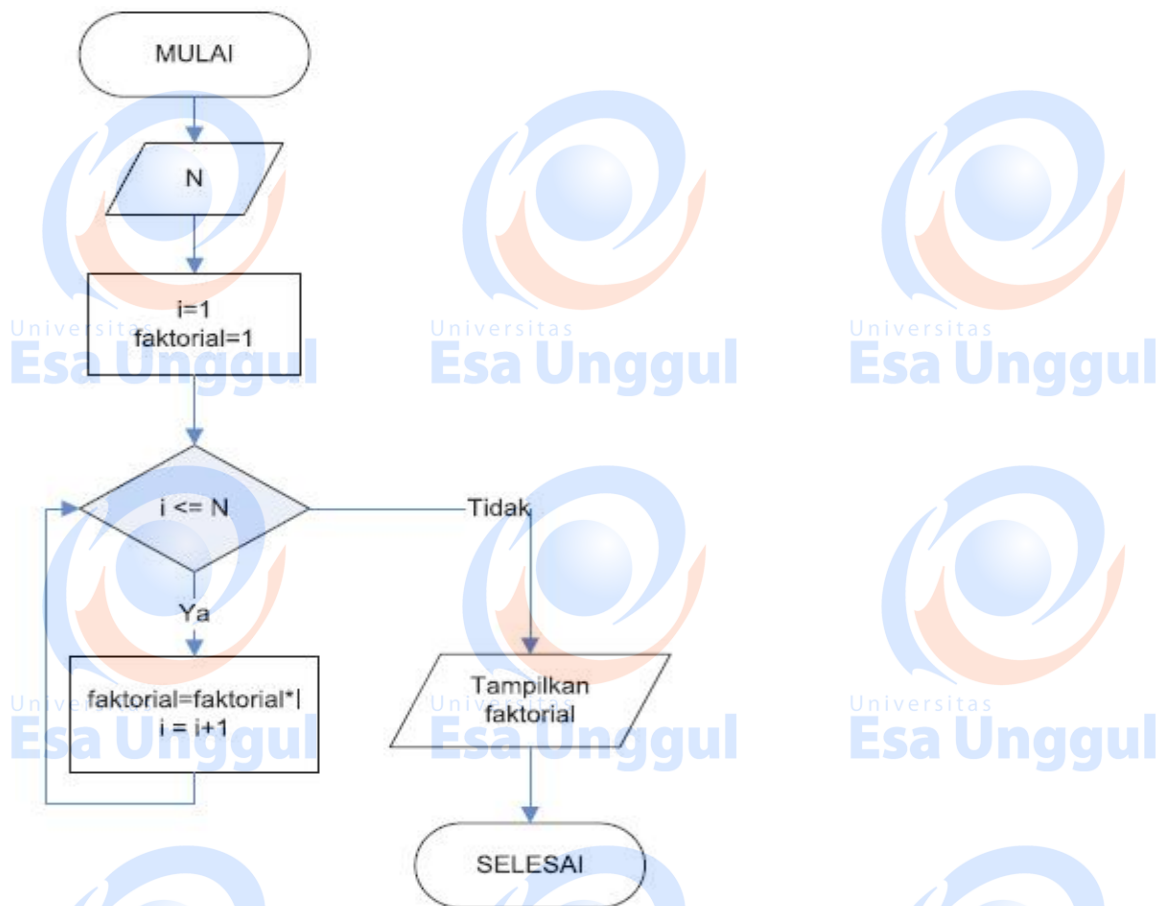
Misal, **for (int i=1; i<=10; i++)**, berarti proses perulangan akan berjalan mulai dari 1 sampai 10, dimana counter i akan naik secara otomatis (i++) hingga i mencapai angka 10.

#### Contoh 4.1:

Akan dihitung nilai N! (N faktorial). Besarnya N ditentukan melalui input data.

**Gambar 4.1** menunjukkan *flowchart* untuk permasalahan tersebut.





Gambar 4.1 Diagram blok untuk contoh 4.1.

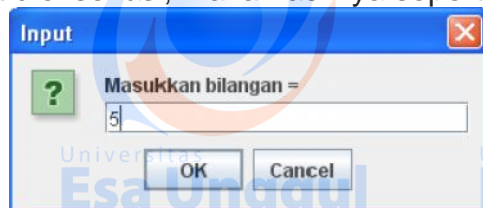
Kode program untuk permasalahan pada contoh 4.1 tersebut dapat dilihat pada Gambar 4.2.

```

1  /* contoh for
2  */
3
4  package contoh_for;
5
6  import javax.swing.*;
7
8  public class Main {
9
10     public static void main(String[] args) {
11         String data_N = JOptionPane.showInputDialog("Masukkan bilangan = ");
12         int N = Integer.parseInt(data_N);
13         int faktorial=1;
14         for (int i=1; i<=N; i++)
15         {
16             faktorial = faktorial*i;
17         }
18         System.out.println("Nilai "+N+"! = "+faktorial);
19     }
20 }
  
```

Gambar 4.2 Kode program untuk contoh 4.1.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada Gambar 4.3.



(a)



```

Output - contoh_for (run)
>>
src:
deps-jar:
cc:
run:
Nilai 5! = 120
BUILD SUCCESSFUL (total time: 24 seconds)

```

(b)

**Gambar 4.3** Output yang dihasilkan oleh program pada contoh 4.1.

**Perintah: while**

Pernyataan while digunakan untuk melakukan perulangan terhadap sejumlah pernyataan dalam blok selama kondisi (dinyatakan dengan ekspresi boolean) bernilai benar.

Sintaks:

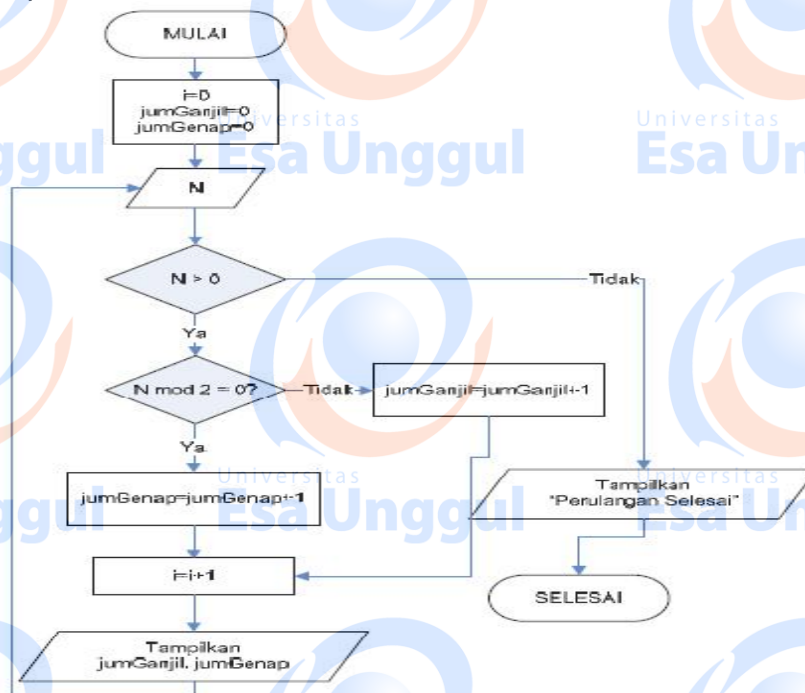
```

while (<ekspresi_boolean>)
{
<blok_pernyataan>
}

```

Contoh 4.2:

Berikut akan dihitung banyaknya bilangan ganjil dan bilangan genap yang dimasukkan melalui dialog box. Proses penentuan banyaknya bilangan akan terus diulang hingga pengguna memasukkan bilangan nol (0). Gambar menunjukkan flowchart untuk permasalahan tersebut.



**Gambar 4.4** Diagram blok untuk contoh 4.2.

Kode program untuk permasalahan pada contoh 4.2 tersebut dapat dilihat pada **Gambar 4.5**. Apabila program tersebut dieksekusi, apabila diberikan input 6, 4, 5, 2, dan 0, maka hasilnya seperti pada **Gambar 4.6**.

```

1  import javax.swing.*;
2  public class Contoh_while{
3      public static void main(String[] args) {
4          int i=0, jumGenap=0, jumGanjil=0;
5          boolean ulang=true;
6          while(ulang){
7              String data_n=JOptionPane.showInputDialog("masukan bilangan");
8              int N = Integer.parseInt(data_n);
9              if(N>0)
10             {
11                 if(N%2==0){
12                     jumGenap=jumGenap+1;
13                 }
14             }
15             else
16             {
17                 jumGanjil=jumGanjil+1;
18             }
19             i=i+1;
20             System.out.println("Banyakna bilangan genap sampai iterasi ke "+i+" = "+jumGenap);
21             System.out.println("Banyakna bilangan ganjil sampai iterasi ke "+i+" = "+jumGanjil);
22         }
23         else
24         {
25             ulang=false;
26         }
27     }
28     System.out.println("Perulangan selesai");
29 }
30 }

```

Gambar 4.5 Kode program untuk contoh 4.2.

```

Output - contoh_while (run)
init:
deps-jar:
compile:
run:
Banyaknya bilangan ganjil sampai iterasi ke-1 = 0
Banyaknya bilangan genap sampai iterasi ke-1 = 1
Banyaknya bilangan ganjil sampai iterasi ke-2 = 0
Banyaknya bilangan genap sampai iterasi ke-2 = 2
Banyaknya bilangan ganjil sampai iterasi ke-3 = 1
Banyaknya bilangan genap sampai iterasi ke-3 = 2
Banyaknya bilangan ganjil sampai iterasi ke-4 = 1
Banyaknya bilangan genap sampai iterasi ke-4 = 3
Perulangan selesai
BUILD SUCCESSFUL (total time: 54 seconds)

```

Gambar 4.6 Output yang dihasilkan oleh program pada contoh 4.2.

### Perintah: do-while

Pada pernyataan do-while, blok pernyataan dieksekusi hingga suatu kondisi yang dinyatakan oleh ekspresi logika pada while bernilai salah. Sedikit berbeda dengan pernyataan while yang baru akan mengeksekusi blok pernyataan setelah diketahui kondisi benar, pada do-while ini blok akan dieksekusi setidaknya satu kali sebelum kondisi bernilai salah.

Sintaks:

```

do
{
<blok pernyataan>
}
while (<ekspresi boolean>);

```

Contoh 4.3:

Pada contoh 4.2 akan coba dibuat dengan menggunakan dowhile. Kode program untuk permasalahan tersebut dapat dilihat pada **Gambar 4.7**.

```
1E import javax.swing.*;
2 public class Contoh_do_while {
3E public static void main(String[] args) {
4     int i=0, jumGenap=0, jumGanjil=0;
5     boolean ulang=true;
6     do{
7         String data_n=JOptionPane.showInputDialog("masukan bilangan");
8         int N = Integer.parseInt(data_n);
9         if(N>0)
10        {
11            if(N%2==0){
12                jumGenap=jumGenap+1;
13            }
14
15            else
16            {
17                jumGanjil=jumGanjil+1;
18            }
19            i=i+1;
20            System.out.println("Banyaknya bilangan genap sampai iterasi ke "+i+" = "+jumGenap);
21            System.out.println("Banyaknya bilangan ganjil sampai iterasi ke "+i+" = "+jumGanjil);
22        }
23        else
24        {
25            ulang=false;
26        }
27    }while(ulang);
28    System.out.println("Perulangan selesai");
29 - }
30 }
```

**Gambar 4.7** Kode program untuk contoh 4.3.

Apabila program tersebut dieksekusi, dengan input bilangan 5,8, 3, 10, dan 10, maka hasilnya seperti pada **Gambar 4.8**.

```
: Output - contoh_do_while (run)
init:
deps-jar:
compile:
run:
Banyaknya bilangan ganjil sampai iterasi ke-1 = 1
Banyaknya bilangan genap sampai iterasi ke-1 = 0
Banyaknya bilangan ganjil sampai iterasi ke-2 = 1
Banyaknya bilangan genap sampai iterasi ke-2 = 1
Banyaknya bilangan ganjil sampai iterasi ke-3 = 2
Banyaknya bilangan genap sampai iterasi ke-3 = 1
Banyaknya bilangan ganjil sampai iterasi ke-4 = 2
Banyaknya bilangan genap sampai iterasi ke-4 = 2
Perulangan selesai
BUILD SUCCESSFUL (total time: 33 seconds)
```

**Gambar 4.8** Output yang dihasilkan oleh program pada contoh4.3.

### 3. Latihan

Lakukan ujicoba pada contoh program diatas dan lakukan analisa

### 4. Tugas

- a. Buatlah sebuah program untuk menampilkan output berikut:  
30 29 28 27 26.....16 1 2 3 4 5 6 7 8 9.....15
- b. Buatlah sebuah program untuk menampilkan sederetan angka genap dan ganjil beserta jumlahnya  
Contoh :

$$1\ 3\ 5\ 7\ 9 = 25$$

$$2\ 4\ 6\ 8\ 10 = 30$$

- c. Buatlah sebuah program untuk menampilkan output berikut dengan menggunakan konsep perulangan do-while:

1  
2  
3  
4  
5  
6  
7  
8  
10 11 12 13 14 15 16 17 18 19 20



## MODUL 5 Array dan String

### 1. Tujuan Pembelajaran:

1. Praktikan mengerti dan memahami penggunaan array dan string
2. Praktikan mampu menggunakan beberapa operator dan method yang menyertai penerapan array dan string.

### 2. Teori Singkat

#### Array

Array merupakan tipe data yang digunakan apabila data diberikan berupa kelompok data yang disajikan secara berurutan. Setiap elemen data memiliki tipe data yang sama. Array menggunakan sekelompok lokasi memori yang berurutan dengan nama dan tipe data yang berbeda.

Sintaks:

```
tipe_data nama_variabel_array[]
```

Besarnya alokasi memori yang akan digunakan perlu diberikan sebelum variabel bertipe array tersebut digunakan. Apabila alokasi memori tidak diberikan secara eksplisit, maka Java akan memberikannya secara implisit. Kemudian, setiap elemen data disebut dengan *nama\_variabel\_array* yang diikuti dengan indeks penomorannya.

Sebagai contoh, apabila akan dibuat variabel penjualan bertipe array dengan alokasi memori sebesar 11, dan akan memberikannya nilai 125 untuk penjualan ke-11, maka dapat dituliskan:

Syntax :

```
int penjualan[];  
penjualan = new int[12];  
penjualan[11] = 125;
```

Untuk mengetahui jumlah elemen dari sekelompok data yang bertipe array, dapat digunakan perintah `length`.

Sintaks:

```
nama_variabel_array.length
```

Contoh:

Program yang ditulis pada **Gambar 5.1** menunjukkan aplikasi variabel penjualan yang bertipe array dimana setiap elemen bertipe integer.



```

1  /* Contoh penggunaan array satu dimensi
2  */
3
4  package contoh_array_1d;
5
6  public class Main {
7
8      public static void main(String[] args) {
9
10         int penjualan[] = {100, 120, 112, 132, 105, 122,
11                            108, 121, 122, 130, 115, 125};
12         int N = penjualan.length;
13
14         for (int i=0; i<N; i++)
15         {
16             System.out.println("Penjualan ke-"+i+" = "+penjualan[i]);
17         }
18     }
19 }
20

```

**Gambar 5.1** Contoh program untuk array satu dimensi.

Apabila program tersebut dieksekusi, maka hasilnya sepertipada **Gambar 5.2**.

```

Output - contoh_array_1d (run)
init:
deps-jar:
Compiling 1 source file to D:\NetBeans\Projects\contoh_array_1d\build\classes
compile:
run:
Penjualan ke-0 = 100
Penjualan ke-1 = 120
Penjualan ke-2 = 112
Penjualan ke-3 = 132
Penjualan ke-4 = 105
Penjualan ke-5 = 122
Penjualan ke-6 = 100
Penjualan ke-7 = 121
Penjualan ke-8 = 122
Penjualan ke-9 = 130
Penjualan ke-10 = 115
Penjualan ke-11 = 125
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Gambar 5.2** Output yang dihasilkan oleh program pada

contoh 5.1.

Model array sebagaimana yang dijelaskan di awal merupakan array satu dimensi. Sangat dimungkinkan suatu variabel membutuhkan array dua dimensi, misalkan pada aplikasi matriks. Sintaks untuk array dua dimensi diberikan sebagai berikut.

Sintaks:

```

tipe_data nama_variabel_array[][]

```

Sebagai contoh, apabila akan dibuat variabel penjualan bertipe array dua dimensi, dengan alokasi memori sebesar 12 untuk dimensi pertama dan 31 untuk dimensi kedua, serta akan memberikan nilai 125 untuk penjualan pada dimensi (11, 23), maka dapat dituliskan:

```

int penjualan[][];
penjualan = new int[12][31];
penjualan[11][23] = 125;

```

Contoh 5.2

Diketahui daftar nilai matakuliah PBO untuk 5 mahasiswa sebagaimana terlihat pada **Tabel 5.1**.

**Tabel 5.1** Daftar nilai PBO mahasiswa contoh 5.1.

Mahasiswa Ke	Nilai		
	Rata 2 Tugas	UTS	UAS
1	50	60	70
2	70	75	87
3	89	90	90
4	65	76	89
5	65	70	80

Selanjutnya akan dicari nilai akhir untuk setiap mahasiswa yang merupakan nilai rata-rata dari ketiga komponen penilaian (rata2 tugas, UTS, dan UAS). Selanjutnya, pemberian nilai dengan huruf diberikan sebagai berikut (**Tabel 5.2**):

**Tabel 5.2** Konversi nilai ke huruf contoh 5.1.

Nilai	Huruf
Nilai > 80	A
70 < Nilai < 80	B
60 < Nilai < 70	C
50 < Nilai < 60	D
Nilai < 50	E

Program yang ditulis pada **Gambar 5.3** menunjukkan aplikasi variabel penjualan yang bertipe array dimana setiap elemen bertipe integer.

```

1 public class Contoh_array_2d {
2     public static void main(String[] args) {
3         double nilai[][]={{50,60,70},{70,75,87},{89,90,90},{65,76,89},{65,70,80}};
4         int N = nilai.length;
5         char huruf;
6         for (int i=0;i<N;i++){
7             int M = nilai[i].length;
8             double totNilai=0;
9             for(int j=0;j<M;j++){
10                totNilai+= nilai[i][j];
11            }
12            double rata2 = totNilai/M;
13            if (rata2 > 80.0d)
14            {
15                huruf='A';
16            }
17            else if (rata2 > 70.0d)
18            {
19                huruf='B';
20            }
21            else if (rata2 > 60.0d)
22            {
23                huruf='C';
24            }
25            else if (rata2 > 50.0d)
26            {
27                huruf='D';
28            }
29            else
30            {
31                huruf='E';
32            }
33
34            System.out.println("Nilai mahasiswa ke-"+i+" = "+rata2+" (" +huruf+" )");
35        }
36    }
37 }

```

**Gambar 5.3** Contoh program untuk array dua dimensi.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 5.4**.

```

Output - contoh_array_2d (run)
init:
deps-jar:
Compiling 1 source file to D:\NetBeansProjects\contoh_array_2d\build\classes
compile:
run:
Nilai mahasiswa ke-0 = 60.0(D)
Nilai mahasiswa ke-1 = 77.33333333333333(B)
Nilai mahasiswa ke-2 = 89.66666666666667(A)
Nilai mahasiswa ke-3 = 76.66666666666667(B)
Nilai mahasiswa ke-4 = 71.66666666666667(B)
BUILD SUCCESSFUL (total time: 0 seconds)

```

Gambar 5.4 Output yang dihasilkan oleh program pada contoh 5.2.

### String

String merupakan deretan karakter. Pada java, string merupakan objek dari kelas String. Penulisan string dilakukan dengan mengapit data menggunakan tanda petik (ganda).

Contoh 5.3:

```
String judul = "Pemrograman Berorientasi Objek";
```

String dapat dioperasikan dengan menggunakan beberapa operator atau method untuk beberapa kepentingan yang berbeda.

#### a. Mengetahui panjang string

Untuk mengetahui panjang string dapat digunakan method `length()`.

Contoh 5.4:

Gambar 5.5 menunjukkan kode program untuk mencari panjang string.

```

1  /* Contoh mencari panjang String
2  */
3
4  package panjangstring;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          String A = "Pemrograman Berorientasi Objek";
10         String B = "Jurusan Teknik Informatika";
11         String C, D, E;
12         C = "Fakultas Teknologi Industri";
13         D = "Universitas Islam Indonesia";
14         E = "Yogyakarta";
15
16         int nA = A.length();
17         int nB = B.length();
18         int nC = C.length();
19
20         System.out.println(A+"--> panjang string = "+nA);
21         System.out.println(B+"--> panjang string = "+nB);
22         System.out.println(C+"--> panjang string = "+C.length());
23         System.out.println(D+"--> panjang string = "+D.length());
24         System.out.println(E+"--> panjang string = "+E.length());
25     }
26 }

```

Gambar 5.5 Contoh program untuk contoh 5.4.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 5.6**.

```

Output - panjangString (run)
init:
deps-jar:
compile:
run:
Pemrograman Berorientasi Objek--> panjang string = 30
Jurusan Teknik Informatika--> panjang string = 26
Fakultas Teknologi Industri--> panjang string = 27
Universitas Islam Indonesia--> panjang string = 27
Yogyakarta--> panjang string = 10
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Gambar 5.6** Output yang dihasilkan oleh program pada contoh 5.4.

**b. Mengetahui kesamaan antara dua string**

Untuk mengetahui kesamaan antara dua string dapat digunakan operator `==` atau method `equal(String)` atau method `equal.IgnoreCase(String)`.

Contoh 5.5:

**Gambar 5.7** menunjukkan kode program untuk mengetahui kesamaan antara dua string. Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 5.8**.

```

1  /* Contoh menguji kesamaan string
2  */
3
4  package kesamaanstring;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          String A = "Laboratorium Komputasi dan Sistem Cerdas";
10         String B = "LABORATORIUM KOMPUTASI DAN SISTEM CERDAS";
11         String C, D, E;
12         C = "Laboratorium Komputasi Sistem Cerdas";
13         D = "LABORATORIUM KOMPUTASI SISTEM CERDAS";
14         E = "Laboratorium Komputasi dan Sistem Cerdas";
15
16         boolean tes1, tes2, tes3;
17         tes1 = A.equals(B);
18         tes2 = (A==C);
19         tes3 = A.equals(E);
20
21         System.out.println("[ "+A+" ]==[ "+B+" ] --> "+tes1);
22         System.out.println("[ "+A+" ]==[ "+C+" ] --> "+tes2);
23         System.out.println("[ "+A+" ]==[ "+E+" ] --> "+tes3);
24         System.out.println("[ "+B+" ]==[ "+D+" ] --> "+B.equals(D));
25         System.out.println("[ "+C+" ]==[ "+E+" ] --> "+(C==E));
26     }
27 }

```

**Gambar 5.7** Contoh program untuk contoh 5.5.

```

Output - kesamaanString (run)
init:
deps-jar:
Compiling 1 source file to D:\NetBeansProjects\kesamaanString\build\classes
compile:
run:
[Laboratorium Komputas: dan Sistem Cerdas]==[LABORATORIUM KOMPUTASI DAN SISTEM CERDAS] --> false
[Laboratorium Komputas: dan Sistem Cerdas]==[Laboratorium Komputasi Sistem Cerdas] --> false
[Laboratorium Komputas: dan Sistem Cerdas]==[Laboratorium Komputasi dan Sistem Cerdas] --> true
[LABORATORIUM KOMPUTAS: DAN SISTEM CERDAS]==[LABORATORIUM KOMPUTASI SISTEM CERDAS] --> false
[Laboratorium Komputas: Sistem Cerdas]==[Laboratorium Komputasi dan Sistem Cerdas] --> false
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Gambar 5.8** Output yang dihasilkan oleh program pada contoh 5.5.

**c. Melakukan perbandingan karakter-karakter pada string**



Untuk melakukan perbandingan karakter-karakter secara berurutan dari awal string dapat digunakan method `compareTo()`.

Contoh 5.6:

**Gambar 5.9** menunjukkan kode program untuk membandingkan karakter-karakter pada string.

```
1  /* Contoh perbandingan string
2  */
3
4  package perbandinganstring;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          String[] nama = {"Megawati Sukarno Putri",
10             "Susilo Bambang Yudoyono",
11             "Jusuf Kalla",
12             "Prabowo Subiyanto",
13             "Boediono",
14             "Wiranto"};
15
16          String sementara;
17          int N = nama.length;
18
19          //Menampilkan nama awal sebelum diurutkan
20          System.out.println("Daftar nama sebelum diurutkan");
21          for (int i=0; i<N; i++)
22          {
23              System.out.println((i+1)+" "+nama[i]);
24          }
25
26          //Pengurutan nama
27          for (int i=0; i<=N-2; i++)
28          {
29              for (int j=i+1; j<=N-1; j++)
30              {
31                  if (nama[i].compareTo(nama[j])>0)
32                  {
33                      sementara = nama[i];
34                      nama[i] = nama[j];
35                      nama[j] = sementara;
36                  }
37              }
38          }
39
40          //Menampilkan setelah diurutkan
41          System.out.println("Daftar nama setelah diurutkan");
42          for (int i=0; i<N; i++)
43          {
44              System.out.println((i+1)+" "+nama[i]);
45          }
46      }
47  }
```

Pembandingan karakter antar string

**Gambar 5.9** Contoh program untuk contoh 5.6.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 5.10**.

```
Output - perbandinganString (run)
init:
deps-jar: Universitas
compile: Universitas
run: Universitas
Daftar nama sebelum diurutkan
1. Megawati Sukarno Putri
2. Susilo Bambang Yudoyono
3. Jusuf Kalla
4. Prabowo Subiyanto
5. Boediono
6. Wiranto
Daftar nama setelah diurutkan
1. Boediono
2. Jusuf Kalla
3. Megawati Sukarno Putri
4. Prabowo Subiyanto
5. Susilo Bambang Yudoyono
6. Wiranto
BUILD SUCCESSFUL (total time: 0 seconds)
```



**Gambar 5.10** Output yang dihasilkan oleh program pada contoh 5.6.

**d. Mendapatkan karakter pada posisi tertentu**

Untuk mendapatkan karakter pada posisi tertentu dapat digunakan method `charAt(int Posisi)`. Posisi pertama bernilai integer 0, hingga posisi terakhir bernilai N-1 (dengan N adalah panjang string).

Contoh 5.7:

**Gambar 5.11** menunjukkan kode program untuk mendapatkan karakter pada posisi tertentu dalam string. Program akan menampilkan karakter pada posisi tertentu yang diberikan secara random dan diulang sebanyak sepuluh kali.

```
1 /* Contoh mendapatkan karakter pada posisi tertentu di string
2 */
3
4 package karakterpadaposisi;
5 import java.util.Random;
6
7 public class Main {
8
9     public static void main(String[] args) {
10         String A = "Laboratorium Komputasi dan Sistem Cerdas";
11         Random rand = new Random();
12         for (int i=1; i<=10; i++)
13         {
14             int ke = rand.nextInt(A.length());
15             System.out.println("Karakter ke-"+ke+" = "+A.charAt(ke));
16         }
17     }
18 }
```

**Gambar 5.11** Contoh program untuk contoh 5.7.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 5.12**.

```
Output - karakterPadaPosisi (run)
init:
deps-jar:
Compiling 1 source file to D:\NetBeansProjects\karakterPadaPosisi\build\classes
compile:
run:
Karakter ke-3 = o
Karakter ke-20 = s
Karakter ke-17 = u
Karakter ke-27 = S
Karakter ke-16 = p
Karakter ke-6 = t
Karakter ke-38 = a
Karakter ke-38 = a
Karakter ke-37 = d
Karakter ke-11 = m
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Gambar 5.12** Output yang dihasilkan oleh program pada contoh 5.7.

**e. Mendapatkan posisi karakter pada string**

Untuk mendapatkan posisi karakter pada string dapat dilakukan dengan dua cara sebagai berikut.

- Apabila posisi awal pencarian tidak ditentukan, maka untuk mendapatkan posisi karakter pada string dapat digunakan method `indexOf(char Karakter)`. Apabila karakter tidak ditemukan, maka akan diberikan nilai -1.
- Apabila posisi awal pencarian telah ditentukan, maka untuk mendapatkan posisi karakter pada string dapat digunakan method `indexOf(char Karakter, int Awal)`. Nilai Awal merupakan bilangan integer yang menunjukkan posisi awal pencarian. Apabila karakter tidak ditemukan, maka akan diberikan nilai -1.

Contoh 5.8:

**Gambar 5.13** menunjukkan kode program untuk mendapatkan posisi suatu karakter dalam string.

```

1  /* Contoh mendapatkan posisi karakter di string
2  */
3
4  package posisikarakter;
5
6  public class Main (
7
8      public static void main(String[] args) {
9          String A = "Laboratorium Komputasi dan Sistem Cerdas";
10
11         System.out.println("Karakter s terletak pada posisi ke:" + A.indexOf('s'));
12         System.out.println("Karakter a terletak pada posisi ke:" + A.indexOf('a'));
13         System.out.println("Karakter v terletak pada posisi ke:" + A.indexOf('v'));
14         System.out.println("Setelah posisi ke-10, karakter s terletak pada " +
15             "posisi ke:" + A.indexOf('s', 10));
16         System.out.println("Setelah posisi ke-10, karakter a terletak pada " +
17             "posisi ke:" + A.indexOf('a', 10));
18     }
19 }

```

Mendapatkan posisi karakter

**Gambar 5.13** Contoh program untuk contoh 5.8.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 5.14**.

```

:Output - posisikarakter (run)
init: |
deps-jar.
Compiling 1 source file to D:\NetBeansProjects\posisikarakter\build\classes
compile:
run:
Karakter s terletak pada posisi ke:20
Karakter a terletak pada posisi ke:1
Karakter v terletak pada posisi ke:-1
Setelah posisi ke-10, karakter s terletak pada posisi ke:20
Setelah posisi ke-10, karakter a terletak pada posisi ke:19
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Gambar 5.14** Output yang dihasilkan oleh program pada contoh 5.8.

**f. Melakukan konversi huruf besar (kapital) ke huruf kecil**

Untuk melakukan konversi huruf besar ke huruf kecil dapat digunakan method `toLowerCase()`.

**g. Melakukan konversi huruf kecil ke huruf besar (kapital)**

Untuk melakukan konversi huruf kecil ke huruf besar dapat digunakan method `toUpperCase()`.

Contoh 5.9:

**Gambar 5.15** menunjukkan kode program untuk melakukan konversi huruf besar ke huruf kecil atau sebaliknya dalam string.

Apabila program tersebut dieksekusi, maka hasilnya seperti pada **Gambar 5.16**.

```

1  /* Contoh konversi karakter di string
2  */
3
4  package konversikarakter;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          String A = "Laboratorium Komputasi dan Sistem Cerdas";
10         String Besar = A.toUpperCase();
11         String Kecil = A.toLowerCase();
12
13         System.out.println("Kalimat awal: "+A);
14         System.out.println("Diubah ke huruf besar: "+Besar);
15         System.out.println("Diubah ke huruf kecil: "+Kecil);
16     }
17 }

```

Gambar 5.15 Contoh program untuk contoh 5.9.

```

Output - konversiKarakter (run)
init:|
deps-jar:
Compiling 1 source file to D:\NetBeansProjects\konversiKarakter\build\classes
compile:
run:
Kalimat awal: Laboratorium Komputasi dan Sistem Cerdas
Diubah ke huruf besar: LABORATORIUM KOMPUTASI DAN SISTEM CERDAS
Diubah ke huruf kecil: laboratorium komputasi dan sistem cerdas
BUILD SUCCESSFUL (total time: 0 seconds)

```

Gambar 5.16 Output yang dihasilkan oleh program pada contoh 5.9.

#### h. Melakukan penyambungan antar dua string

Untuk melakukan penyambungan antar dua string dapat digunakan operator +.

Contoh 5.10:

Gambar 5.17 menunjukkan kode program untuk menyambung string.

```

1  /* Contoh penggabungan string
2  */
3
4  package gabungstring;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          String A, B, C;
10         A = "Laboratorium";
11         B = "Komputasi ";
12         C = "Sistem Cerdas";
13
14         String Lab = A + B + " dan " + C;
15         System.out.println(Lab);
16     }
17 }

```

Gambar 5.17 Contoh program untuk contoh 5.10.

Apabila program tersebut dieksekusi, maka hasilnya sepertipada Gambar 5.18.

```

Output - gabungString (run)
init:|
deps-jar:
Compiling 1 source file to D:\NetBeansProjects\gabungString\build\classes
compile:
run:
LaboratoriumKomputasi dan Sistem Cerdas
BUILD SUCCESSFUL (total time: 0 seconds)

```

Gambar 5.18 Output yang dihasilkan oleh program pada contoh 5.10.



### i. Membuat string baru

Untuk membuat string baru dari string yang sudah pernah adadapat dilakukan dengan dua cara sebagai berikut.

- Untuk membuat string baru dari string yang telah ada mulaidari karakter awal string dapat digunakan method *substring(int awal)*.
- Untuk membuat string baru dari string yang telah ada mulaidari karakter awal hingga karakter akhir string dapatdigunakan method *substring(int awal, int akhir)*.

Contoh 5.11:

**Gambar 5.19** menunjukkan kode program untuk membuatstring baru dari string yang telah ada.

```
1  /* Contoh pemenggalan string
2  */
3
4  package stringbaru;
5
6  public class Main (
7
8      public static void main(String[] args) {
9      String A = "Laboratorium Komputasi dan Sistem Cerdas";
10
11      String penggal1 = A.substring(10);
12      String penggal2 = A.substring(20);
13      String penggal3 = A.substring(10,30);
14      String penggal4 = A.substring(20,25);
15
16      System.out.println("Kalimat asli: "+A);
17      System.out.println("Pemenggalan mulai karakter ke-10: "+penggal1);
18      System.out.println("Pemenggalan mulai karakter ke-20: "+penggal2);
19      System.out.println("Pemenggalan mulai karakter ke-10 sampai ke-30: "+
20      penggal3);
21      System.out.println("Pemenggalan mulai karakter ke-20 sampai ke-25: "+
22      penggal4);
23
24  }
```

Membuat string baru dari string yang telah ada

**Gambar 5.19** Contoh program untuk contoh 5.11.

Apabila program tersebut dieksekusi, maka hasilnya sepertipada **Gambar 5.20**.

```
Output - stringBaru (run)
init:|
deps-jar:
Compiling 1 source file to D:\NetBeansProjects\stringBaru\build\classes
compile:
run:
Kalimat asli: Laboratorium Komputasi dan Sistem Cerdas
Pemenggalan mulai karakter ke-10: um Komputasi dan Sistem Cerdas
Pemenggalan mulai karakter ke-20: si dan Sistem Cerdas
Pemenggalan mulai karakter ke-10 sampai ke-30: um Komputasi dan Sis
Pemenggalan mulai karakter ke-20 sampai ke-25: si da
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Gambar 5.20** Output yang dihasilkan oleh program padacontoh 5.11.

### j. Melakukan modifikasi string

Untuk melakukan modifikasi string dapat dilakukan dengandua cara sebagai berikut.

- Untuk me-*replace* karakter pada string dengan karakter barudapat digunakan method *replace(char karakterLama, charkarakterBaru)*.
- Untuk menghilangkan spasi di awal dan si akhir string dapatdigunakan method *trim()*.

Contoh 5.12:

**Gambar 5.21** menunjukkan kode program untuk memodifikasistring yang telah ada.

```

1  /* Contoh modifikasi string
2  */
3
4  package modifikasistring;
5
6  public class Main (
7
8      public static void main(String[] args) (
9          String A = "  Laboratorium Komputasi dan Sistem Cerdas  ";
10
11         String B = A.replace("a", "A");
12         String C = A.replace("i", "u");
13         String D = A.trim();
14
15         System.out.println("Kalimat asli: "+A);
16         System.out.println("Replace a dengan A: "+B);
17         System.out.println("Replace i dengan u: "+C);
18         System.out.println("Hilangkan spasi awal & akhir: "+D);
19     )
20 )

```

Gambar 5.21 Contoh program untuk contoh 5.12.

Apabila program tersebut dieksekusi, maka hasilnya sepertipada Gambar 5.22.

```

Output - modifikasiString (run)
init:|
deps-jar:
compile:
run:
Kalimat asli:   Laboratorium Komputasi dan Sistem Cerdas
Replace a dengan A:   Laboratorium Komputasi dan Sistem Cerdas
Replace i dengan u:   Laboratorium Komputasi dan Sistem Cerdas
Hilangkan spasi awal & akhir: Laboratorium Komputasi dan Sistem Cerdas
BUILD SUCCESSFUL (total time: 0 seconds)

```

Gambar 5.22 Output yang dihasilkan oleh program pada contoh 5.12.

### k. Membuat array karakter dari string

Untuk membuat array karakter dari string yang telah ada dapat dilakukan dengan dua cara sebagai berikut.

- Menggunakan method `toCharArray()` yang diperoleh dari class `String`. Implementasi dari method ini membutuhkan adanya pendeklarasian variabel array `char` untuk menampung larik karakter yang dihasilkan.
- Menggunakan method `getChars(int Awal, int Akhir, char[] arrayChar, int posAwal)`, dengan `Awal` merupakan posisi awal karakter pada string, `Akhir` merupakan posisi akhir karakter pada string, `arrayChar` merupakan variabel untuk menyimpan larik karakter yang dihasilkan, dan `posAwal` merupakan indeks awal untuk menyimpan karakter pertama.

Contoh 5.13:

Gambar 5.23 menunjukkan kode program untuk membuat array karakter dari string yang telah ada.



```

1  /* Contoh konversi string ke array
2  */
3  package konversistringarray;
4
5  public class Main {
6
7      public static void main(String[] args) {
8          String A = "Lab. KSC";
9          char[] karakter0 = A.toCharArray();
10         char[] karakter1 = new char[7];
11         A.getChars(2, 6, karakter1, 1);
12
13         System.out.println("Kalimat asli: "+A);
14
15         //Penulisan per karakter
16         System.out.println("Per karakter:");
17         for (int i=0; i<A.length(); i++)
18         {
19             System.out.println("karakter ke-"+(i+1)+" : "+karakter0[i]);
20         }
21
22         //Penulisan per karakter dari karakter ke-2 sampai 6
23         System.out.println("Karakter ke-2 sampai ke-6:");
24         for (int i=1; i<karakter1.length; i++)
25         {
26             System.out.println("karakter ke-"+i+" : "+karakter1[i]);
27         }
28     }
29 }

```

Gambar 5.23 Contoh program untuk contoh 5.13.

Apabila program tersebut dieksekusi, maka hasilnya sepertipada Gambar 5.24.

```

Output - konversiStringArray (run)
init:
deps-jar:
compile:
run:
Kalimat asli: Lab. KSC
Per karakter:
Karakter ke-1: L
Karakter ke-2: a
Karakter ke-3: b
Karakter ke-4: .
Karakter ke-5: K
Karakter ke-6: S
Karakter ke-7: C
Karakter ke-2 sampai ke-6:
Karakter ke-1: b
Karakter ke-2: .
Karakter ke-3: K
Karakter ke-4: S
Karakter ke-5: C
BUILD SUCCESSFUL (total time: 0 seconds)

```

Gambar 5.24 Output yang dihasilkan oleh program pada contoh 5.14.

### 3. Latihan

Silahkan lakukan ujicoba pada kode program diatas dan lakukan evaluasi

### 4. Tugas

- Buatlah program antrian FIFO
- Buatlah program antrian LIFO
- Buatlah program searching dan sorting pada sebuah elemen Array

## MODUL 6 dan 7 Kelas, Objek dan Method

### 1. Tujuan Pembelajaran:

- a. Praktikan mampu memahami konsep dasar pemrograman Java.
- b. Praktikan dapat membedakan arti dan pemakaian kelas, objek, referensiobjek, method, constructor, dan beberapa kata kunci dalam pemrograman Java.

### 2. Teori Singkat

#### Class, Object, Method, Attribute



Attribute:  
Topi, Baju, Jaket,  
Tas Punggung,  
Tangan, Kaki, Mata

Behavior:  
Cara Jalan ke Depan  
Cara Jalan Mundur  
Cara Belok ke Kiri  
Cara Memanjat



Attribute (State):  
Ban, Stir, Pedal Rem, Pedal Gas,  
Warna, Tahun Produksi

Behavior:  
Cara Menghidupkan Mesin  
Cara Manjalkan Mobil  
Cara Memundurkan Mobil

Attribute → Variable(Member)  
Behavior → Method(Fungsi)

#### Perbedaan Class dan Object

- ✓ Class: konsep dan deskripsi dari sesuatu  
Class mendeklarasikan method yang dapat digunakan (dipanggil) oleh object
- ✓ Object: instance dari class, bentuk (contoh) nyata dari class  
Object memiliki sifat independen dan dapat digunakan untuk memanggil method
- ✓ Contoh Class dan Object:  
Class: mobil  
Object: mobilnya pak Joko, mobilku, mobil berwarna merah
- ✓ Class seperti cetakan kue, dimana kue yang dihasilkan dari cetakan kue itu adalah object
- ✓ Warna kue bisa bermacam-macam meskipun berasal dari cetakan yang sama (object memiliki sifat independen)

Person
name : string
age : integer

Class with Attributes

(Person)
Joe Smith
24

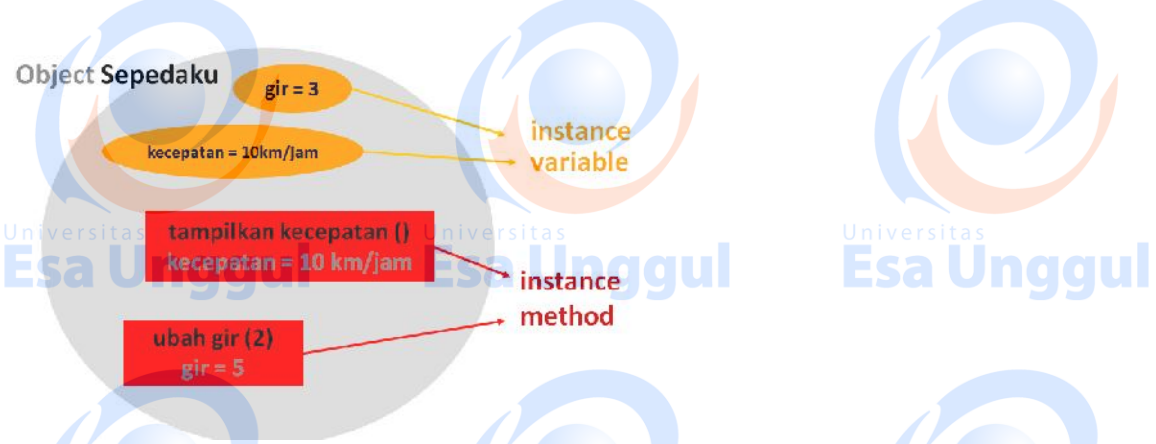
(Person)
Mary Sharp
52

Objects with Values

#### Class = Method + Variable



## Object = Method + Variable yang Memiliki Nilai



### Attribute

- ✓ Variable yang mengitari class, dengan nilai datanya bisa ditentukan di object
- ✓ Variable digunakan untuk menyimpan nilai yang nantinya akan digunakan pada program
- ✓ Variable memiliki jenis (tipe), nama dan nilai
- ✓ Name, age, dan weight adalah atribute (variabel) dari class Person

<b>Person</b>
name : string
age : integer

Class with Attributes

<b>(Person)</b>
Joe Smith
24

Objects with Values

<b>(Person)</b>
Mary Sharp
52

## Membuat Class, Object dan Memanggil Attribute

Mobil.java

```
public class Mobil {
    String warna;
    int tahunProduksi;
}
```

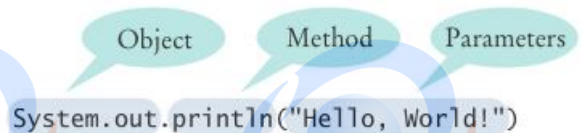
MobilBeraksi.java

```
public class MobilBeraksi{
    public static void main(String[] args){
        // Membuat object
        Mobil mobilku = new Mobil();

        /* memanggil atribut dan memberi nilai */
        mobilku.warna = "Hitam";
        mobilku.tahunProduksi = 2006;
        System.out.println("Warna: " + mobilku.warna);
        System.out.println("Tahun: " + mobilku.tahunProduksi);
    }
}
```

### Method

- ✓ Method adalah urutan instruksi yang mengakses data dari object
- ✓ Method melakukan:
  1. Manipulasi data
  2. Perhitungan matematika
  3. Memonitor kejadian dari suatu event



## Membuat dan Memanggil Method

Mobil2.java

```
public class Mobil2{
    String warna;
    int tahunProduksi;
    void printMobil(){
        System.out.println("Warna: " + warna);
        System.out.println("Tahun: " + tahunProduksi);
    }
}
```

Mobil2Beraksi.java

```
public class Mobil2Beraksi{
    public static void main(String[] args){
        Mobil2 mobilku = new Mobil2();
        mobilku.warna = "Hitam";
        mobilku.tahunProduksi = 2006;
        mobilku.printMobil();
    }
}
```

## Latihan

- ✓ Buat class **Handphone**, masukkan dalam package **hp**
- ✓ Class **Handphone** berisi empat method di bawah:
  1. hidupkan()
  2. lakukanPanggilan()
  3. kirimSMS()
  4. matikan()
- ✓ Isi masing-masing method dengan tampilan status menggunakan **System.out.println()**
- ✓ Buat class **HandphoneBeraksi**, dan panggil method-method diatas dalam class tersebut

## Latihan: Hasil Tampilan

Handphone hidup ...

Kring, kring, kring ... panggilan dilakukan

Dung, dung ... sms berhasil terkirim

Handphone mati ...

## Apa Itu Kelas?

Kelas merupakan inti dari pemrograman Java karena Java adalah bahasa pemrograman yang mendukung dan mengimplementasikan konsep pemrograman berorientasi objek sepenuhnya. Setiap program Java merupakan kelas, sehingga setiap konsep atau kasus pemrograman yang akan diimplementasikan dengan Java harus dibungkus ke dalam sebuah kelas.

Kelas dapat didefinisikan sebagai cetak biru (*blueprint*) atau *prototype*/kerangka yang mendefinisikan variabel-variabel (data) dan *method-method* (perilaku) umum dari sebuah objek tertentu. Sebagai contoh, kita ambil objek *Mahasiswa*. *Mahasiswa* memiliki data seperti *nim*, *nama*, *alamat*, *IPK*, *jenis kelamin*, *jurusan*, dan sebagainya. Selain data atau ciri-ciri fisik tersebut, *mahasiswa* juga memiliki perilaku-perilaku spesifik yang dapat membedakan antara mahasiswa yang satu dengan yang lainnya, seperti *cara presentasi*, *cara belajar*, *cara mengerjakan tugas* dan sebagainya.



Dalam dunia pemrograman, sebenarnya kelas tidak jauh berbeda dengan tipe data sederhana seperti *integer*, *char*, *boolean*, dan sebagainya. Perbedaannya, tipe data sederhana digunakan untuk mendeklarasikan variabel 'normal', sedangkan kelas digunakan untuk mendeklarasikan sebuah variabel yang berupa objek. Variabel yang berupa objek ini sering disebut dengan referensi objek (*object reference*).

### Mendefinisikan Kelas

Dalam Java, kelas didefinisikan dengan menggunakan katakunci *class*. Berikut ini bentuk umum yang digunakan untuk mendefinisikan sebuah kelas.

```
class NamaKelas {
    tipe data1;
    tipe data2;
    ...
    tipe dataN;

    tipe method1(daftar-parameter) {
        //kode untuk method1
    }

    tipe method2(daftar-parameter) {
        //kode untuk method2
    }
    ...
    tipe methodN(daftar-parameter) {
        //kode untuk methodN
    }
}
```

Data atau variabel yang didefinisikan di dalam sebuah kelas sering disebut dengan *instance variable* yang selanjutnya akan diakses melalui *method-method* yang ada. Data dan *method* yang tergabung dalam suatu kelas sering dinamakan sebagai *class members*.

### Contoh Kelas Sederhana

Pada bagian ini akan diberikan pembahasan terkait dengan pembuatan sebuah kelas sederhana. Di sini kita akan membuat kelas *Karyawan*, yang memiliki data-data: *ID*, *nama*, *divisi*, dan *gaji*. Untuk saat ini, belum ditambahkan *method* ke dalam kelas tersebut. *Method* akan dibahas terpisah pada bagian selanjutnya dalam bab ini.

Perhatikan kode berikut ini

#### Program 6.1 Contoh pembuatan kelas.

```
Public class Karyawan
{
    String ID,nama,divisi;
    Double gaji;
}
```



Melalui kode di atas, berarti kita telah mendefinisikan sebuah tipe data baru dengan nama *Karyawan*. Penting untuk diingat bahwa pendefinisian kelas hanya akan membuat sebuah pola atau *template*, bukan membuat objek. Objek aktual dari kelas tersebut harus dibuat sendiri melalui kode berikut:

```
//membuat objek karyawan dengan nama Aurel  
Karyawan Aurel = new Karyawan();
```

Di sini, *Karyawan* adalah kelas dan *Aurel* adalah objek atau *instance* dari kelas *Karyawan*. Melalui objek *Aurel*, kita dapat mengakses dan memanipulasi data-data yang terdapat pada kelas *Karyawan*, dengan cara menggunakan operator titik (*.*), seperti yang tampak pada kode di bawah ini.

```
Aurel.ID = "K001";  
Aurel.divisi = "Aurel DIan";  
Aurel.nama = "Marketing";  
Aurel.gaji= "2500000";
```

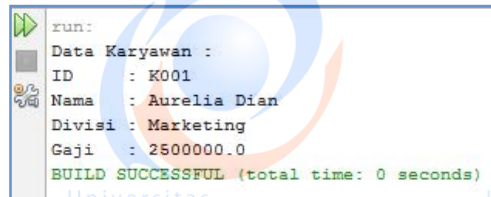
kode tersebut digunakan untuk mengisi nilai ke dalam data *ID*, *nama*, *divisi* dan *gaji* yang dimiliki oleh objek *Aurel* masing-masing dengan nilai “K001”, “Aurelia Dian”, “Marketing” dan 2500000 Untuk lebih jelasnya, coba perhatikan contoh kode program lengkap berikut ini yang akan menunjukkan penggunaan kelas *Karyawan* di atas.

**Program 6.2** Contoh instansiasi objek dan pengaksesan data pada objek.

```
public class ImplementasiKelasKaryawan  
{  
    public static void main(String[] args)  
    {  
        //membuat objek karyawan dengan nama Aurel  
        Karyawan Aurel = new Karyawan();  
  
        //mengisi nilai ke dalam data-data Objek Karyawan  
        Aurel.ID = "K001";  
        Aurel.divisi = "Aurel DIan";  
        Aurel.nama = "Marketing";  
        Aurel.gaji = "2500000";  
  
        //mencetak data-data object karyawan  
        System.out.println("Data Karyawan");  
        System.out.println("ID      : " + Aurel.ID);  
        System.out.println("Nama   : " + Aurel.nama);  
        System.out.println("Divisi : " + Aurel.divisi);  
        System.out.println("Gaji   : " + Aurel.gaji);  
    }  
}
```

Kode program di atas menunjukkan contoh implementasi dari kelas *Karyawan* yang telah dibuat sebelumnya, yaitu dimulai dengan membuat objek *Karyawan*, mengisi nilai ke dalam data-data yang dimiliki oleh objek *Karyawan* yang telah diinstansiasi, kemudian mencetak data-data objek *Karyawan* tersebut.

Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:



```
run:
Data Karyawan :
ID      : K001
Nama    : Aurelia Dian
Divisi  : Marketing
Gaji    : 2500000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 6.1 Hasil eksekusi program6.2

Perlu diketahui bahwa setiap objek atau *instance* dari suatu kelas akan memiliki salinan data sendiri-sendiri. Ini artinya, antara objek yang satu dengan yang lainnya, nilai data-datanya bisa berbeda. Perhatikan kode program di bawah ini.

**Program 6.3** Contoh instansiasilebih dari satu objek dari satu kelas yang sama

```
public class ImplementasiKelasKaryawan {
    public static void main(String[] args) {
        //membuat objek karyawan dengan nama Aurel
        Karyawan Aurel = new Karyawan();

        //membuat objek karyawan dengan nama Dhennis
        Karyawan Dhennis = new Karyawan();

        //mengisi nilai kedalam data-data Objek Karyawan2
        Aurel.ID = "K001";
        Aurel.nama = "Marketing";
        Aurel.divisi = "Aurel Dian";
        Aurel.gaji = "2500000";

        //mengisi nilai kedalam data-data Objek Karyawan2
        Dhennis.ID = "K002";
        Dhennis.nama = "Keuangan";
        Dhennis.divisi = "Muhamad Dhennis";
        Dhennis.gaji = "2250000";

        //mencetak data-data object karyawan ke-1
        System.out.println("ID      : " + Aurel.ID);
        System.out.println("Nama    : " + Aurel.nama);
        System.out.println("Divisi  : " + Aurel.divisi);
        System.out.println("Gaji    : " + Aurel.gaji);

        //mencetak data-data object karyawan ke-2
        System.out.println("ID      : " + Dhennis.ID);
        System.out.println("Nama    : " + Dhennis.nama);
        System.out.println("Divisi  : " + Dhennis.divisi);
        System.out.println("Gaji    : " + Dhennis.gaji);
    }
}
```

Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:

```
run:|
Data Karyawan Pertama:
ID      : K001
Nama    : Aurelia Dian
Divisi  : Marketing
Gaji    : 2500000.0

Data Karyawan Kedua:
ID      : K002
Nama    : Muhammad Dhennis
Divisi  : Keuangan
Gaji    : 2250000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 6.2 Hasil eksekusi program 6.3

### Instansiasi Objek

Untuk membuat objek baru dari tipe kelas yang telah didefinisikan, secara eksplisit kita perlu melakukan dua tahap. Pertama, kita perlu mendeklarasikan variabel yang digunakan sebagai referensi ke objek dari kelas bersangkutan. Kedua, kita perlu menginstansiasi kelas dengan menggunakan operator *new* dan memasukkan *instance*-nya ke dalam variabel referensi yang baru saja dideklarasikan. Operator *new* secara dinamis akan mengalokasikan ruang memori untuk menyimpan suatu objek tertentu dan mengembalikan nilai berupa referensi ke objek bersangkutan.

Berikut ini kode yang mengilustrasikan dua tahapan proses yang dimaksud di atas. Untuk mempermudah pembahasan, di sini masih menggunakan kelas *Karyawan* yang telah dibuat sebelumnya.

```
//mendeklarasikan variabel Aurel bertipe
Karyawan
Karyawan Aurel;
//instansiasi & memasukkan referensi ke
variabel Aurel
Aurel = new Karyawan();
```

Pada praktiknya, dua tahap di atas biasanya ditulis dalam satu baris, seperti berikut:

```
Karyawan Aurel = new Karyawan();
```

### Mengisi Nilai pada Referensi Objek

Terdapat satu buah catatan penting yang perlu diperhatikan pada saat kita memasukkan nilai pada sebuah variabel referensi. Sebelumnya, perhatikan terlebih dahulu kode berikut:

```
Karyawan karyawan001, karyawan002;
Karyawan001 = new Karyawan();
Karyawan002 = Karyawan001;
```

Baris pertama digunakan untuk mendeklarasikan variabel referensi ke objek *Karyawan* dengan nama *karyawan001* dan *karyawan002*. Baris kedua berfungsi untuk membuat objek *Karyawan* dan menyimpan referensinya ke dalam variabel *karyawan001*. Dan baris ketiga digunakan memasukkan *karyawan001* ke dalam

*karyawan002*. Ini artinya, variabel *karyawan002* berperan sebagai referensi ke objek yang sedang ditunjuk oleh *karyawan001*. Dengan demikian, variabel *karyawan001* dan *karyawan002* masing-masing menunjuk ke objek *Karyawan* yang sama. Maka dari itu, setiap perubahan yang terjadi pada objek yang bersangkutan melalui *karyawan001* akan berpengaruh juga pada objek yang ditunjuk oleh *karyawan002*, karena keduanya sebenarnya adalah objek yang sama, begitu juga sebaliknya.

#### Program 6.4 Contoh pengaruh referensi objek (1)

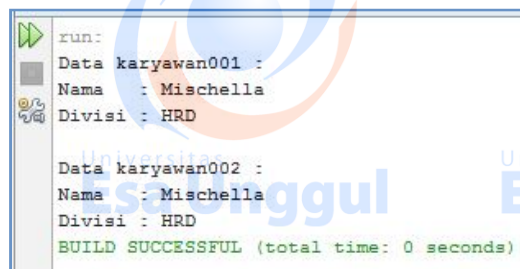
```
public class PengaruhReferensiObjek1 {
    public static void main(String[] args) {
        /*Instansiasi 2 objek referensi yang mengacu pada 1 objek
        karyawan */
        Karyawan Karyawan001 = new Karyawan();
        Karyawan Karyawan002 = Karyawan001;

        //mengisi data Objek Karyawan melalui objek referensi 1
        Karyawan001.nama = "Mischella";
        Karyawan001.divisi = "HRD";

        //mencetak data object karyawan yang di acu 2 objek referensi
        System.out.println("Data Karyawan001");
        System.out.println("Nama      : " + Karyawan001.nama);
        System.out.println("Divisi   : " + Karyawan001.divisi);
        System.out.println(" ");
        System.out.println("Data Karyawan002");
        System.out.println("Nama      : " + Karyawan002.nama);
        System.out.println("Divisi   : " + Karyawan002.divisi);
    }
}
```

Tampak bahwa kita hanya memasukkan nilai untuk objek *karyawan001*, namun pada saat kita mencetak data pada *karyawan002*, hasil yang diberikan adalah sama seperti yang dihasilkan oleh *karyawan001*. Ini disebabkan karena *karyawan001* dan *karyawan002* sebenarnya menunjuk ke objek *Karyawan* yang sama.

Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:



```
run:
Data karyawan001 :
Nama      : Mischella
Divisi   : HRD

Data karyawan002 :
Nama      : Mischella
Divisi   : HRD
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 6.3 hasil eksekusi program 6.4

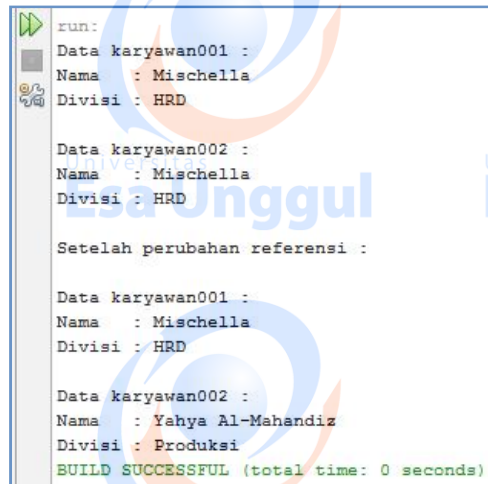
Sebagai catatan penting, meskipun keduanya menunjuk ke objek yang sama, akan tetapi tidak terdapat hubungan antara variabel *karyawan001* dan *karyawan002*. Artinya, kita bisa saja melepas salah satu variabel tersebut untuk menunjuk ke objek lain, tanpa berpengaruh terhadap variabel lainnya. Sebagai contoh, perhatikan kode program berikut:



### Program 6.5 Contoh pengaruh referensi objek (2)

```
public class PengaruhReferensiObjek2 {  
    public static void main(String[] args) {  
        /*Instansiasi 2 objek referensi yang mengacu pada 1 objek karyawan  
*/  
        Karyawan Karyawan001 = new Karyawan();  
        Karyawan Karyawan002 = Karyawan001;  
        //mengisi data Objek Karyawan melalui objek referensi 1  
        Karyawan001.nama = "Mischella";  
        Karyawan001.divisi = "HRD";  
        //mencetak data object karyawan yang di acu 2 objek referensi  
        System.out.println("Data Karyawan001");  
        System.out.println("Nama      : " + Karyawan001.nama);  
        System.out.println("Divisi   : " + Karyawan001.divisi);  
        System.out.println(" ");  
        System.out.println("Data Karyawan002");  
        System.out.println("Nama      : " + Karyawan002.nama);  
        System.out.println("Divisi   : " + Karyawan002.divisi);  
        //memindahkan objek referensi ke-2 untuk mengacu objek baru  
        Karyawan002 = new Karyawan();  
        Karyawan002.nama="Yahya Al-Mahandis";  
        Karyawan002.divisi="Produksi";  
        //mencetak data objek setelah perubahan referensi  
        System.out.println(" ");  
        System.out.println("Setelah Perubahan Referensi");  
        System.out.println(" ");  
        System.out.println("Data Karyawan001");  
        System.out.println("Nama      : " + Karyawan001.nama);  
        System.out.println("Divisi   : " + Karyawan001.divisi);  
        System.out.println(" ");  
        System.out.println("Data Karyawan002");  
        System.out.println("Nama      : " + Karyawan002.nama);  
        System.out.println("Divisi   : " + Karyawan002.divisi);  
    }  
}
```

Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:



```
run:
Data karyawan001 :
Nama : Mischella
Divisi : HRD

Data karyawan002 :
Nama : Mischella
Divisi : HRD

Setelah perubahan referensi :

Data karyawan001 :
Nama : Mischella
Divisi : HRD

Data karyawan002 :
Nama : Yahya Al-Mahandiz
Divisi : Produksi

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

Gambar 6.4 Hasil eksekusi program 6.5

Dari hasil tersebut dapat terlihat bahwa pada saat variabel *karyawan002* dipindahkan untuk menunjuk ke objek lain (terjadi perubahan referensi objek), data-data *karyawan002* yang dicetak akan berbeda dengan data-data dari *karyawan001*.

### Method

Dalam Java, kelas berisi kumpulan data dan *method*, yang selanjutnya akan saling bekerja sama dalam melakukan tugas-tugas spesifik tertentu sesuai dengan perilaku objek yang dimodelkan.

Berikut ini bentuk umum dari pembuatan *method* di dalam kelas.

```
tipe namaMethod(daftar-parameter) {
//kode yang akan dituliskan
}
```

Pada bentuk umum di atas, tipe adalah tipe data yang akan dikembalikan oleh *method*. Sebagai catatan, dalam Java *method* terbagi menjadi dua: *void* dan *non-void*. *Method void* adalah *method* yang tidak mengembalikan nilai, sedangkan *method non-void* adalah *method* yang mengembalikan nilai. Jika *method* yang kita buat ditujukan untuk mengembalikan suatu nilai tertentu, maka di dalam *method* tersebut harus terdapat statemen *return*, yang diikuti dengan nilai yang akan dikembalikan, seperti berikut:

```
return nilaiKembalian;
```

*nilaiKembalian* dapat berupa konstanta maupun variabel, yang digunakan untuk menandakan nilai yang akan dikembalikan oleh *method*.

### Mendefinisikan Method

Setelah mengetahui bentuk umum yang digunakan, di sini kita akan langsung mendefinisikan sebuah contoh *method* ke dalam kelas yang sebelumnya kita buat, yaitu kelas *LapanganFutsal*. Kita akan menambahkan *method cetakPengelola()*

untuk mencetak nama *pengelola* dari objek *LapanganFutsal*. *Method* ini merupakan *method* yang tidak mengembalikan nilai. maka dari itu, kita harus mengganti tipe kembalian dengan kata kunci *void*. Berikut ini kode program yang dimaksud.

**Program 6.6** Penambahan *method void* pada deklarasi kelas *Karyawan*

```
public class Karyawan {
    String ID, nama, divisi;
    Double gaji;
    void cetakData() {
        System.out.println("Data Karyawan ");
        System.out.println("ID      : " + ID);
        System.out.println("Nama   : " + nama);
        System.out.println("Divisi  : " + divisi);
        System.out.println("Gaji   : " + gaji);
    }
}
```

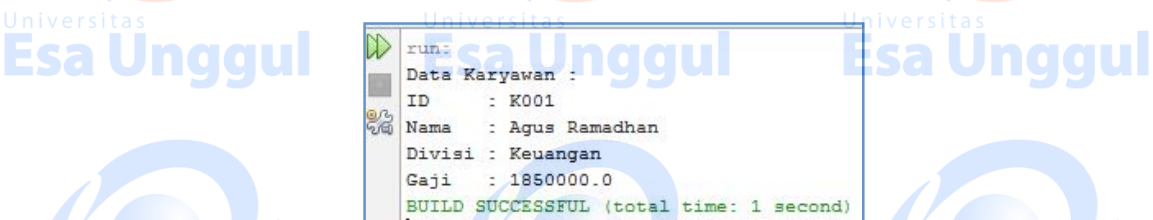
**Program 6.7** Contoh pemanggilan *method void*

```
public class ImplementasiMethodVoid {
    public static void main(String[] args) {
        //instansiasi objek karyawan
        Karyawan Karyawan001 = new Karyawan();

        //mengisi data pada objek karyawan
        Karyawan001.ID = "K001";
        Karyawan001.nama = "Agus Ramadhan";
        Karyawan001.divisi = "Keuangan";
        Karyawan001.gaji = 1850000;

        //memanggil method cetakData();
        Karyawan001.cetakData();
    }
}
```

Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:



**Gambar 6.5** Hasil eksekusi program 6.7

Pemanggilan *method* dalam suatu kelas dilakukan dengan menuliskan objek pemiliknya, dan diikuti oleh operator titik (.) beserta nama *method* yang akan dieksekusi. Berikut ini contoh kode yang digunakan untuk memanggil *method cetakData()* yang telah dideklarasikan di dalam kelas *Karyawan*.

```
//memanggil method cetakData
Karyawan001.cetakData();
```

Hasil yang diberikan oleh *method cetakData()* akan menyesuaikan dengan data-data yang tersimpan pada objek di mana *method cetakData()* tersebut dideklarasikan

**Pengembalian Nilai di dalam Method**

Sekarang, kita akan melanjutkan pembahasan dengan memperlihatkan cara pendefinisian *method* yang dapat mengembalikan nilai. Di sini, kita akan memodifikasi program sebelumnya dengan menambahkan *method* `hitungSumbanganZakat()` yang akan mengembalikan nilai dengan tipe *double*. Berikut ini kode program yang dimaksud.

Berikut ini kode program yang dimaksud.

**Program 6.8** Contoh implementasi *method non-void*

```
public class Karyawan {

    String ID, nama, divisi;
    double gaji;

    void cetakData() {
        System.out.println("Data Karyawan : ");
        System.out.println("ID      : " + ID);
        System.out.println("Nama   : " + nama);
        System.out.println("Divisi  : " + divisi);
        System.out.println("Gaji   : " + gaji);
    }

    double hitungSumbanganZakat() {
        double zakat = gaji * 0.025;
        return zakat;
    }
}
```

**Program 6.9** Pemanggilan *method non-void*

```
public class ImplementasiMethodNonVoid {

    public static void main(String[] args) {
        //instantiasi objek Karyawan
        Karyawan karyawan001 = new Karyawan();

        //mengisi data pada objek Karyawan
        karyawan001.ID = "K001";
        karyawan001.nama = "Agus Ramadhan";
        karyawan001.divisi = "Keuangan";
        karyawan001.gaji = 1850000;

        //memanggil method cetakData()
        karyawan001.cetakData();

        //memanggil method hitungSumbanganZakat()
        System.out.println("Sumbangan      Zakat      :      "      +
            karyawan001.hitungSumbanganZakat());
    }
}
```

Pada program di atas, *method* `hitungSumbanganZakat()` mengembalikan nilai bertipe *double* sehingga hasilnya pun dapat dilewatkan langsung melalui `println()`. Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:



```
run:
Data Karyawan :
ID      : K001
Nama    : Agus Ramadhan
Divisi  : Keuangan
Gaji    : 1850000.0
Sumbangan Zakat : 46250.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 6.6 Hasil eksekusi program 6.6

### Method Berparameter

Pada kenyataannya sebagian besar *method* yang ditulis dalam program memiliki satu atau beberapa parameter. Dengan adanya parameter, sebuah *method* dapat bersifat dinamis dan general. Artinya, *method* tersebut dapat mengembalikan nilai yang beragam sesuai dengan nilai parameter yang dilewatkannya. Coba perhatikan terlebih dahulu pendefinisian *method* tanpa parameter berikut ini.

```
double volumeTabung() {
return (3.14 * 7 * 7 * 10);
}
```

Secara sintaks, *method* di atas benar dan akan mengembalikan nilai 1538.5. Namun, dilihat dari sisi desain program, *method* di atas dikatakan tidak fleksibel atau tidak bersifat general. Pasalnya, *method* tersebut hanya dapat digunakan untuk tabung yang memiliki jari-jari alas 7 dan tinggi 10. Bagaimana jika terdapat tabung dengan jari-jari alas 10 dan tinggi 12? *Method* di atas tidak dapat kita gunakan untuk menghitung volume dari tabung dengan ukuran tersebut. Untuk mengatasi hal semacam ini, maka Java mengizinkan kita untuk menempatkan parameter di dalam sebuah *method* agar *method* di atas dapat digunakan untuk segala ukuran tabung. Berikut ini kode yang seharusnya dituliskan untuk mendefinisikan *method* di atas.

```
double volumeTabung(int jejari, int tinggi)
{
return (3.14 * jejari * jejari * tinggi);
}
```

Melalui cara seperti ini, kita dapat melewatkan nilai berapa pun ke dalam *method* tersebut. Sebagai contoh, perhatikan kode berikut:

```
int volume1, volume2, volume3;
//menghasilkan nilai 1538.6
volume1 = volumeTabung(7, 10)
//menghasilkan nilai 3768
volume2 = volumeTabung(10, 12)
//menghasilkan nilai 4019.2
volume3 = volumeTabung(8, 20)
```

Seperti kita lihat, *volume1* merupakan variabel penampung untuk perhitungan volume tabung dengan jari-jari alas 7 dan tinggi 10, *volume2* dengan jari-jari alas 10 dan tinggi 12, dan *volume3* dengan jari-jari alas 8 dan tinggi 20; padahal *method* yang didefinisikan hanya satu (bukan tiga buah *method*).

Sekarang kita akan mengimplementasikan konsep di atas ke dalam kelas *Karyawan*. Di sini data-data yang dimiliki oleh objek *Karyawan* akan kita isikan melalui sebuah *method*. Berikut ini kode program yang dapat digunakan untuk melakukan hal tersebut.

**Program 6.10** Implementasi *method* dengan parameter pada kelas *Karyawan*

```
public class Karyawan {  
  
    String ID, nama, divisi;  
    double gaji;  
  
    void isiData(String kode, String Nama, String  
Div, double Gaji) {  
        ID = kode;  
        nama = Nama;  
        divisi = Div;  
        gaji = Gaji;  
    }  
  
    void cetakData() {  
        System.out.println("Data Karyawan : ");  
        System.out.println("ID      : " + ID);  
        System.out.println("Nama   : " + nama);  
        System.out.println("Divisi : " + divisi);  
        System.out.println("Gaji   : " + gaji);  
    }  
  
    double hitungSumbanganZakat() {  
        double zakat = gaji * 0.025;  
        return zakat;  
    }  
}
```

**Program 6.11** Pemanggilan *method* berparameter pada kelas *Karyawan*

```
public class ImplementasiMethodBerparameter {  
  
    public static void main(String[] args) {  
        //instansi objek Karyawan  
        Karyawan karyawan001 = new Karyawan();  
        /*mengisi data pada objek Karyawan  
melalui method isiData()*/  
        karyawan001.isiData("k001", "Rommy", "Marketing", 2350000);  
  
        //memanggil method cetakData();  
        karyawan001.cetakData();  
    }  
}
```

Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:

```

run:
Data Karyawan :
ID      : k001
Nama    : Rommy
Divisi  : Marketing
Gaji    : 2350000.0
BUILD SUCCESSFUL (total time: 1 second)

```

**Gambar 6.7** Hasil eksekusi program 6.9

Pada kode di atas kita mendefinisikan *method* yang tidak mengembalikan nilai dan memiliki empat (4) buah parameter, yaitu *kode*, *Nama*, *Div*, dan *Gaji*. Parameter ini selanjutnya dapat diisi dengan nilai yang dikehendaki pada saat proses pemanggilan *method* tersebut. Dalam contoh di atas, kita melewati nilai “k001”, “Rommy”, “Marketing”, dan 2350000 sebagai argumen dari *method* *isiData()* untuk mengisi data *ID*, *nama*, *divisi*, dan *gaji* dari objek *Karyawan*.

### **Constructor**

*Constructor* adalah *method* khusus yang didefinisikan di dalam kelas dan akan dipanggil secara otomatis tiap kali terjadi instansiasi objek. *Constructor* itu sendiri berfungsi untuk melakukan inialisasi nilai terhadap data-data yang terdapat pada kelas yang bersangkutan. Jika kita tidak mendefinisikan *constructor* pada kelas yang kita buat, maka secara otomatis Java akan membuatnya untuk kita. *Constructor* semacam ini dinamakan dengan *defaultconstructor*. *Defaultconstructor* akan menginisialisasi semua data yang ada dengan nilai nol. Sama halnya seperti *method*, *constructor* juga dapat memiliki parameter dan juga dapat di-overload.

Berikut ini contoh program yang menunjukkan pembuatan *constructor* untuk kelas *Karyawan* dan nilai awal dari data-data yang terdapat pada kelas tersebut akan kita isi dengan nilai awal. *Constructor* ini kita tambahkan sebagai pengganti *method* *isiData()* yang telah dibuat sebelumnya.

### **Program 6.12** Deklarasi kelas *Karyawan* dengan *constructor*

```

public class Karyawan {
    String ID, nama, divisi;
    double gaji;

    //constructor kelas karyawan
    Karyawan() {
        ID = "k001";
        nama = "Budi";
        divisi = "Produksi";
        gaji = "1750000";
    }

    void cetakData() {
        System.out.println("Data Karyawan   :");
        System.out.println("ID      : " + ID);
        System.out.println("Nama    : " + nama);
        System.out.println("Divisi  : " + divisi);
        System.out.println("Gaji    : " + gaji);
    }

    double hitungSumbanganZakat() {
        double zakat = gaji * 0.025;
        return zakat;
    }
}

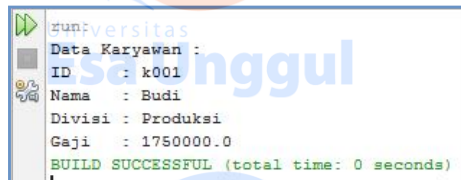
```

```
}  
}
```

**Program 6.13**Instansiasi objek *Karyawan* dengan menggunakan *constructor*

```
public class ImplementasiConstructor {  
    public static void main(String[] args) {  
        Karyawan karyawan001 = new Karyawan();  
  
        karyawan001.cetakData();  
    }  
}
```

Pada kode Program 6.12, kita mengisikan nilai awal untuk data *ID*, *nama*, *divisi*, dan *gaji* masing-masing dengan nilai “k001”, “Budi”, “Produksi”, dan 1750000. Hal ini akan berlaku untuk setiap pembentukan objek *Karyawan*. Selanjutnya, pada saat Program 6.13dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:



**Gambar 6.8** Hasil eksekusi program 6.13

Proses inisialisasi dengan cara di atas masih dapat dikatakan kurang dinamis. Pasalnya, setiap kali pembentukan objek *Karyawan*, nilai yang dijadikan nilai awal selalu “k001”, “Budi”, “Produksi”, dan 1750000. Untuk menjadikan *constructor* lebih bersifat dinamis, maka kita dapat membubuhkan parameter ke dalamnya. Hal ini mirip dengan implementasi *method isiData()* pada pembaharuan *method* berparameter yang sebelumnya sudah dibahas. Implementasi *constructor* berparameter ditunjukkan pada contoh program berikut ini.

**Program 6.14**Deklarasi kelas *Karyawan* dengan *constructor* dinamis

```
public class Karyawan {  
    String ID, nama, divisi;  
    double gaji;  
    //constructor kelas Karyawan  
    Karyawan(String kode, String Nama, String Div, double Gaji) {  
        ID = kode;  
        nama = Nama;  
        divisi = Div;  
        gaji = Gaji;  
    }  
  
    void cetakData() {  
        System.out.println("Data Karyawan :");  
        System.out.println("ID      : " + ID);  
        System.out.println("Nama   : " + nama);  
        System.out.println("Divisi : " + divisi);  
        System.out.println("Gaji   : " + gaji);  
    }  
}
```



```

double hitungSumbanganZakat() {
    double zakat = gaji * 0.025;
    return zakat;
}
}

```

Kali ini, *constructor* yang kita buat ditambah dengan parameter. Hal ini menyebabkan nilai awal dapat bersifat dinamis tergantung dari nilai-nilai yang dilewatkan pada saat pembentukan objek.

**Program 6.15** Instansiasi objek *Karyawan* dengan menggunakan *constructor* dinamis

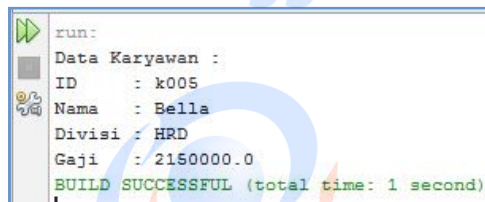
```

public class ImplementasiConstructorDinamis {

    public static void main(String[] args) {
        Karyawan karyawan001 = new Karyawan("k005",
        "Bella", "HRD", 2150000);
        karyawan001.cetakData();
    }
}

```

Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:



**Gambar 6.9** Hasil eksekusi program 6.15

### Kata Kunci *This*

Pada kasus-kasus tertentu, terkadang *method* perlu mengacu keobjek yang memilikinya. Untuk mengizinkan hal ini, Javamenyediakan kata kunci *this*. *this* sebenarnya merupakanreferensi ke objek yang sedang aktif. *this* digunakan di dalam*method* untuk mewakili nama kelas bersangkutan. Sebagaicontoh, perhatikan kode berikut ini.

```

//constructor kelas Karyawan
Karyawan(String kode, String Nama, String
Div, double Gaji) {
    ID = kode;
    nama = Nama;
    divisi = Div;
    gaji = Gaji;
}

```

Sebenarnya kode di atas juga dapat ditulis dengan menggunakan referensi *this*, seperti berikut:

```

//constructor kelas Karyawan
Karyawan(String kode, String Nama, String Div, double Gaji) {
    this.ID = kode;
    this.nama = Nama;
    this.divisi = Div;
}

```

```
    this.gaji = Gaji;
}
```

Untuk kasus di atas, kehadiran kata kunci *this* memang justru akan sedikit merepotkan dan bias dikatakan tidak perlu. Meskipun demikian, dalam konteks lain, penggunaan kata kunci *this* ini justru diharuskan.

Seperti yang telah kita ketahui bersama, Java tidak mengizinkan kita untuk mendeklarasikan variabel dengan nama sama di dalam satu blok tertentu. Di sisi lain, Java memperbolehkan kita untuk mendeklarasikan nama parameter yang sama dengan nama data yang terdapat di dalam kelas dengan resiko variabel yang digunakan sebagai parameter akan menimpa nama data yang terdapat di dalam kelas bersangkutan. Untuk menghindari proses penimpaan variabel ini, maka kita harus menggunakan kata kunci *this*, seperti yang tampak pada kode berikut:

```
//constructor kelas Karyawan
Karyawan(String ID, String nama, String
divisi, double gaji) {
    this.ID = ID;
    this.nama = nama;
    this.divisi = divisi;
    this.gaji = gaji;
}
```

Di sini, *this.ID*, *this.nama*, *this.divisi*, dan *this.gaji* menunjuk ke data *ID*, *nama*, *divisi*, dan *gaji* yang terdapat dalam kelas *Karyawan*; bukan parameter yang terdapat pada *constructor*.

### Melakukan *Overload* terhadap *Method*

Kita dapat mendefinisikan lebih dari satu *method* dengan nama yang sama dalam sebuah kelas, dengan syarat parameter yang terdapat pada *method-method* tersebut berbeda. Parameter dalam suatu *method* dikatakan berbeda dari *method* yang lainnya apabila:

- Jumlahnya berbeda, meskipun tipe datanya sama
- Tipe datanya berbeda, meskipun jumlahnya sama
- Jumlah dan tipe datanya berbeda

Prose pendefinisian *method* dengan nama sama ini disebut dengan *overload*. Perhatikan dua buah contoh *method* berikut ini.

```
int tambah(int a, int b) {
    return a + b;
}
double tambah(double a, double b) {
    return a + b;
}
```

Kode di atas legal untuk didefinisikan di dalam sebuah kelas. Pada contoh tersebut, kita mendefinisikan dua buah *method* dengan nama yang sama, tapi parameternya berbeda.

Berikut ini contoh program yang akan menunjukkan pendefinisian proses *overloadmethod* di dalam sebuah kelas.

**Program 6.16** Kelas *Pembagian* dengan *method* yang di-*overload*

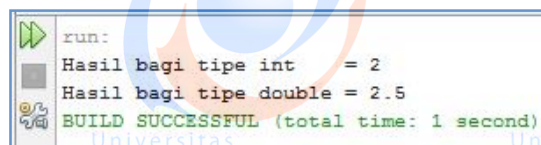
```
public class Pembagian {
    //method dengan dua parameter bertipe
    int
        int bagi(int a, int b) {
            return a / b;
        }
    //method dengan dua parameter bertipe
    double
        double bagi(double a, double b) {
            return a / b;
        }
}
```

**Program 6.17** Pemanggilan *method* yang di-*overload* pada kelas *Pembagian*

```
public class ContohOverloadMethod {
    public static void main(String[] args) {
        Pembagian b = new Pembagian();
        int x = b.bagi(10, 4);
        double y = b.bagi(10.0, 4.0);

        System.out.println("Hasil bagi tipe int
        = " + x);
        System.out.println("Hasil bagi tipe
        double = " + y);
    }
}
```

Saat dijalankan, Program 6.17 tersebut di atas akan memberikan hasil sebagai berikut:



**Gambar 6.10** hasil eksekusi program 6.17

**Overload pada Constructor**

Selain pada *method*, proses *overload* juga dapat diaplikasikan ke dalam *constructor* suatu kelas. Hal ini disebabkan karena sebenarnya *constructor* juga adalah sebuah *method* yang mengembalikan tipe kelas (dirinya sendiri). Pada kenyataannya, suatu kelas pada umumnya justru memiliki lebih dari satu *constructor*. Untuk mendemonstrasikan bagaimana Java melakukan hal itu, perhatikan contoh program di bawah ini.

**Program 6.18** Contoh *overload* terhadap *constructor* kelas *Karyawan*

```
public class Karyawan {
```

```

String ID, nama, divisi;
double gaji;

//constructor pertama
Karyawan(String ID, String nama) {
    this.ID = ID;
    this.nama = nama;
}
//constructor kedua
Karyawan(String ID, String nama, String divisi, double
gaji) {
    this.ID = ID;
    this.nama = nama;
    this.divisi = divisi;
    this.gaji = gaji;
}

void cetakData() {
    System.out.println("Data Karyawan :");
    System.out.println("ID : " + ID);
    System.out.println("Nama : " + nama);
    System.out.println("Divisi : " + divisi);
    System.out.println("Gaji : " + gaji);
}

double hitungSumbanganZakat() {
    double zakat = gaji * 0.025;
    return zakat;
}
}

```

**Program 6.19** Instansiasi objek *Karyawan* menggunakan *constructor* yang telah di-*overload*

```

public class ImplemetasiOverloadConstructor {

    public static void main(String[] args) {
        //instantiasi menggunakan constructor pertama
        Karyawan karyawan01 = new Karyawan("k006", "Zaky");
        //instantiasi menggunakan constructor kedua
        Karyawan karyawan02 = new Karyawan("k007", "Deva",
"Keuangan", 2250000);

        karyawan01.cetakData();
        System.out.println("");
        karyawan02.cetakData();
    }
}

```

Saat dijalankan, Program 6.19 tersebut di atas akan memberikan hasil sebagai berikut:





Gambar 6.11 Hasil eksekusi program 6.19

### Menentukan Tingkat Akses Data dan Method

Seperti yang telah dibahas pada bab-bab sebelumnya, dalam pembungkusan (*encapsulation*), sebenarnya kita menggabungkan data dan kode (*method*) menjadi satu. Pada situasi semacam ini, kita dapat menentukan tingkat akses dari data dan *method* tersebut menggunakan kata kunci *private*, *public*, maupun *protected*.

Dalam pemrograman berorientasi objek, data pada umumnya didesain untuk bersifat *private* atau hanya dapat diakses oleh kelas yang memilikinya saja. Pengaksesan data dari bagian luar hanya dapat dilakukan melalui *method-method* yang ada. Ini artinya, *method* berperan sebagai antarmuka (*interface*). Maka dari itu, *method* pada umumnya bersifat *public*. Meskipun demikian, apabila kita ingin mendefinisikan *method* yang hanya dapat diakses oleh kelas itu sendiri, maka kita pun harus mendeklarasikannya dengan kata kunci *private*. Dan jika kita menginginkan data dan *method* yang kita definisikan di dalam kelas hanya dapat diakses oleh kelas-kelas turunan dari kelas yang bersangkutan, maka data dan *method* tersebut harus dideklarasikan dengan kata kunci *protected*.

Secara umum, pembungkusan data dan *method* dari suatu kelas dapat dituliskan seperti berikut:

```

class NamaKelas {
    tingkat-akses data1;
    tingkat-akses data2;
    ...
    tingkat-akses dataN;
    tingkat-akses method1(daftar-parameter) {
        //kode untuk method1
    }
    tingkat-akses methodN(daftar-parameter) {
        //kode untuk methodN
    }
    ...
}

```

```

tingkat-akses methodN(daftar-parameter) {
//kode untuk methodN
}
}

```

dalam hal ini tingkat-akses adalah salah satu dari kata kunci *private*, *public*, maupun *protected*. Apabila kita tidak mencantumkan salah satu kata kunci tersebut pada saat mendeklarasikan data maupun *method*, maka secara default Java akan menganggapnya sebagai data atau *method* dengan sifat *public*.

Berikut ini contoh program yang akan menunjukkan pengaruh dari tingkat akses *public* dan *private* di dalam suatu kelas. Tingkat akses *protected* akan dijelaskan lebih lanjut pada bab lain di modul ini, yaitu yang membahas mengenai pewarisan sifat terhadap kelas turunan (*inheritance*).

#### Program 6.20 Deklarasi kelas Mahasiswa

```

public class Mahasiswa {
    int nim;
    public String nama;
    private double ipk;

    public void setIPK(double nilai) {
        ipk = nilai;
    }

    public double getIPK() {
        return ipk;
    }
}

```

**Program 6.21** Contoh pengaruh tingkat akses data dan *method* pada kelas *Mahasiswa*

```

public class PengaruhHakAkses {
    public static void main(String[] args) {
        Mahasiswa Yahya = new Mahasiswa();

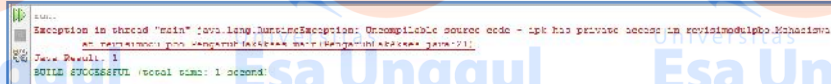
        //BENAR, karena nim secara default bersifat public
        Yahya.nim = 1009;
        //BENAR, karena nama bersifat public
        Yahya.nama = "Yahya Al-Mahandiz";
        Yahya.ipk = 3.77; //SALAH, karena ipk bersifat private

        //BENAR, karena method setC() bersifat public
        Yahya.setIPK(3.77);
        System.out.println("NIM : " + Yahya.nim);
        System.out.println("Nama : " + Yahya.nama);
        System.out.println("IPK : " + Yahya.getIPK());
    }
}

```

Pada kode program di atas, terdapat tanda *error* (yang bergaris bawah) yang menandakan bahwa variabel *ipk* tidak bisa diakses secara langsung karena didefinisikan menggunakan hak akses *private*.

Jika dipaksakan untuk dijalankan, program tersebut di atas akan menampilkan pesan kesalahan seperti ditunjukkan pada Gambar 6.12.



Gambar 6.12 Hasil eksekusi program 6.21

### Kata Kunci *static*

Secara normal, suatu anggota kelas (baik data maupun *method*) hanya dapat diakses melalui objek referensinya. Meskipun demikian, Java mengizinkan kita untuk mengakses suatu anggota kelas tanpa harus membuat objek atau *instance*-nya terlebih dahulu. Untuk membuat anggota kelas semacam ini, kita harus menjadikan data atau *method* tersebut dengan sifat statis, yaitu dengan membubuhkan kata kunci *static* pada awal deklarasi. Berikut ini contoh pendeklarasian sebuah data dan *method* statis dalam suatu kelas.

```
class ContohDeklarasiStatis {  
    //mendeklarasikan data statis  
    static int dataStatis1;  
    ....  
    //mendeklarasikan method statis  
    static methodStatis1() {...}  
    ....  
}
```

Melalui cara ini, variabel *dataStatis1* dan *methodStatis1()* dapat dipanggil secara langsung tanpa harus membuat objek atau *instance* dari kelas *ContohDeklarasiStatis* terlebih dahulu.

Untuk lebih memahami cara kerja dari data dan *method* statis, coba perhatikan contoh program berikut ini.

### Program 6.22 Contoh implementasi data dan *method* statis

```
public class ContohStatis {  
  
    static int dataA = 10;  
    static int dataB = 7;  
    int dataC = 4; //data non-statik  
  
    static void test() {  
        int dataC = dataA + dataB;  
        System.out.println("dataA + dataB = " + dataC);  
        //System.out.println("dataC = "+dataC);  
        //SALAH, karena dataC non-statik  
    }  
}
```

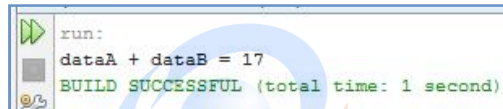
```

    public static void main(String[] args) {
        test();
    }
}

```

Pada contoh program di atas, kita dapat memanggil *method* `test()` tanpa harus membuat objek atau *instance* dari kelas *ContohStatic*. Satu hal lagi, *method* `test()` tidak dapat mengakses *dataC*, karena *data* tersebut tidak dideklarasikan sebagai *data* statis.

Saat dijalankan, program tersebut di atas akan memberikan hasil sebagai berikut:



**Gambar 6.13** Hasil eksekusi program 6.22

Sama halnya seperti *data* dan *method* normal (non-statis), *data* dan *method* statis juga dapat dipanggil melalui operator titik (`.`). Perbedaannya, pemanggilan *data* atau *method* statis tidak perlu menggunakan objek referensi, melainkan nama kelas yang memilikinya.

Coba perhatikan contoh program berikut ini.

**Program 6.23** Deklarasi kelas yang menggunakan *data* dan *method* statis

```

public class RobotStatic {
    static String nama;
    static String asalNegara;
    static void test() {
        System.out.println("Hello..");
        System.out.println("Saya adalah robot " + nama);
        System.out.println("Saya berasal dari " +
asalNegara);
    }
}

```

**Program 6.24** Contoh pemanggilan *data* dan *method* statis

```

public class ContohStatic2 {
    public static void main(String[] args) {
        RobotStatic.nama = "Gathotkaca";
        RobotStatic.asalNegara = "Indonesia";
        RobotStatic.test();
    }
}

```

Tampak jelas bahwa kita dapat mengakses *data* `nama` dan `asalNegara` beserta *method* `test()` dari kelas *RobotStatic* tanpa harus membentuk objek atau *instance* dari kelas *RobotStatic* terlebih dahulu. Namun, yang perlu diperhatikan, kita dituntut untuk



membubuhkan nama kelas yang memiliki data atau *method* yang akan diakses. Saat dijalankan, program tersebut diatas akan memberikan hasil sebagai berikut:

```
run:
Hello..
Saya adalah Robot Gathotkaca
Saya berasal dari Indonesia
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 6.14 Hasil eksekusi program 6.24

Bentuk umum dari pengaksesan data dan *method* statis dapat dituliskan sebagai berikut:

```
NamaKelas.data
```

Atau

```
NamaKelas.method()
```

### Kata Kunci final

Dalam java, kata kunci *final* dapat kita gunakan untuk mendeklarasikan sebuah konstanta. Jika kita ingin menggunakan kata kunci *final* untuk mendeklarasikan suatu variabel sebagai konstanta, maka kita perlu mengisikan nilai ke dalam variabel tersebut. nilai tersebut akan bersifat tetap atau tidak dapat diubah selama eksekusi program berlangsung. Berikut ini bentuk umum dari pendeklarasikan sebuah konstanta di dalam Java.

```
tingkat-akses final tipe-data nama-
konstanta = nilai;
```

Seperti halnya pada bahasa C/C++, dalam Java, konstanta pada umumnya ditulis dengan huruf kapital. Berikut ini beberapa contoh pendeklarasian konstanta.

```
final double PHI = 3.14;
private final int ERROR = -1;
private final String NEGARA = "INDONESIA";
```

Berikut ini contoh program yang akan menunjukkan penggunaan kata kunci *final* untuk mendeklarasikan sebuah konstanta.

**Program 6.25** Deklarasi kelas *Pegawai* yang mengimplementasikan kata kunci *final*

```
public class Pegawai {
    String nama;
    final double bonusLembur = 50000;

    double hitungBonusLembur(double jumlahLembur) {
        return bonusLembur * jumlahLembur;
    }
}
```

**Program 6.26** Contoh implementasi kata kunci *final*

```
public class ImplemetasiFinal {
    public static void main(String[] args) {
        Pegawai Dhennis = new Pegawai();
    }
}
```

```

        Dhennis.nama = "Dhennis Al-MAhandiz";

        System.out.println("Data Karyawan : ");
        System.out.println("nama : " +
Dhennis.nama);
        System.out.println("Bonus lembur : " +
Dhennis.hitungBonusLembur(7));
    }
}

```

Saat dijalankan, Program 6.26 di atas akan memberikan hasil sebagai berikut:

```

run:
Data Karyawan :
nama : Dhennis Al-Mahandiz
Bonus lembur : 350000.0
BUILD SUCCESSFUL (total time: 1 second)

```

Gambar 6.15 Hasil eksekusi program 6.26

### Kelas di dalam Kelas (*Inner Class*)

Java mengizinkan kita untuk mendefinisikan suatu kelas di dalam kelas lainnya. Kelas semacam ini dinamakan sebagai *inner class* (kelas bagain dalam). *Inner class* dapat mengakses data dan *method* dari kelas yang berada di bagian luarnya, meskipun data dan *method* tersebut bersifat *private*.

Berikut ini contoh program yang akan mengilustrasikan pendefinisian dan penggunaan *inner class*.

### Program 6.27 Deklarasi kelas yang mengandung *inner class*

```

public class Notebook {
    private String merk = "Lenovo";
    class SistemOperasi {

        private String namaOS = "Windows 7";

        public void cetakData() {
            System.out.println("Merk notebook : " + merk);
            System.out.println("Sistem operasi : " + namaOS);
        }
    }

    public void cetakdata() {
        SistemOperasi OS = new SistemOperasi();
        OS.cetakData();
    }
}

```

### Program 6.28 Implementasi konsep *inner class*

```

public class ImplemetasiInnerClass {
    public static void main(String[] args) {
        Notebook myNotebook = new Notebook();
        myNotebook.cetakData();
    }
}

```

Saat dijalankan, Program 6.28 di atas akan memberikan hasil sebagai berikut:

```
run:  
Merk notebook : Lenovo  
Sistem operasi : Windows 7  
BUILD SUCCESSFUL (total time: 1 second)
```

**Gambar 6.16** Hasil eksekusi program 6.28.

### 3. Latihan

Lakukan praktek diatas sebagai bentuk latihan dan lakukan evaluasi

### 4. Tugas

Bualah sebuah program kalkulator sederhana yang didalamnya terdapat operasi aritmatika sederhana dan terdapat validasi inputan implementasikan program ini dengan interface class dan objek.



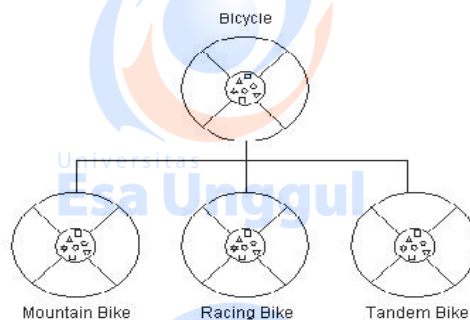
## Modul 8 Pewarisan (*Inheritance*)

### 1. Tujuan Pembelajaran:

- a. Praktikan mampu menterjemahkan UML ke dalam bentuk pemrograman.
- b. Praktikan dapat membedakan pemakaian *overloading* dan *overriding*.
- c. Praktikan mampu menerapkan konsep pewarisan, *overloading* dan *overriding* dalam pemrograman dengan Java.

### 2. Teori Singkat Inheritance (Pewarisan)

- ✓ Suatu class dapat mewariskan atribut dan method kepada class lain (subclass), serta membentuk class hierarchy
- ✓ Penting untuk Reusability
- ✓ Java Keyword: `extends`



Sepeda.java

```

public class Sepeda{
    int gir;
    void setGir(int penambahanGir) {
        gir= gir+ penambahanGir;
    }
    int getGir() {
        return gir;
    }
}
    
```

**Class SepedaGunung mewarisi Class Sepeda**

SepedaGunung.java

```

public class SepedaGunung extends Sepeda{
    private int sadel;
    void setSadel (int jumlah) {
        sadel = getGir() - jumlah;
    }

    int getSadel(){
        return sadel;
    }
}
    
```

SepedaGunungBeraksi.java

```

public class SepedaGunungBeraksi {
    public static void main(String[] args) {
        SepedaGunung sg=new SepedaGunung();
        sg.setGir(3);

        System.out.println(sg.getGir());
        sg.setSadel(1);
        System.out.println(sg.getSadel());
    }
}
    
```

### LATIHAN: INHERITANCE MATEMATIKA

1. Buat class MatematikaCanggih yang merupakan inherit dari class Matematika
  - a. Tambahkan method `modulus(int a, int b)` yang menghitung modulus dari a dan b
  - b. Operator modulus adalah %
2. Buat class MatematikaCanggihBeraksi yang memanggil method pertambahan, perkalian dan modulus



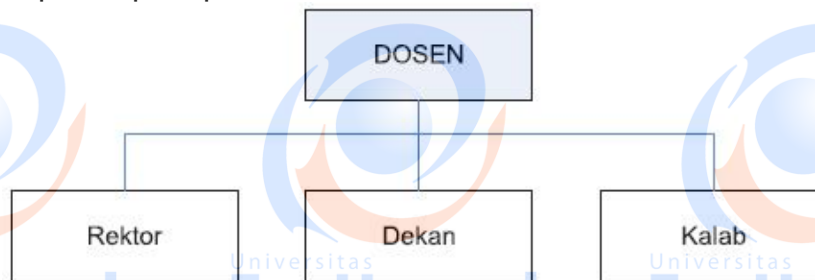
## Pewarisan

Pewarisan (*inheritance*) adalah suatu cara pembuatan class baru dengan menggunakan kembali class yang sudah didefinisikan sebelumnya dengan menambahkan atribut dan method baru. Sehingga dengan demikian class baru tersebut tetap memiliki variabel dan fungsi yang dimiliki oleh class sebelumnya. Pada konsep pewarisan ada beberapa istilah yang perlu diketahui, yaitu:

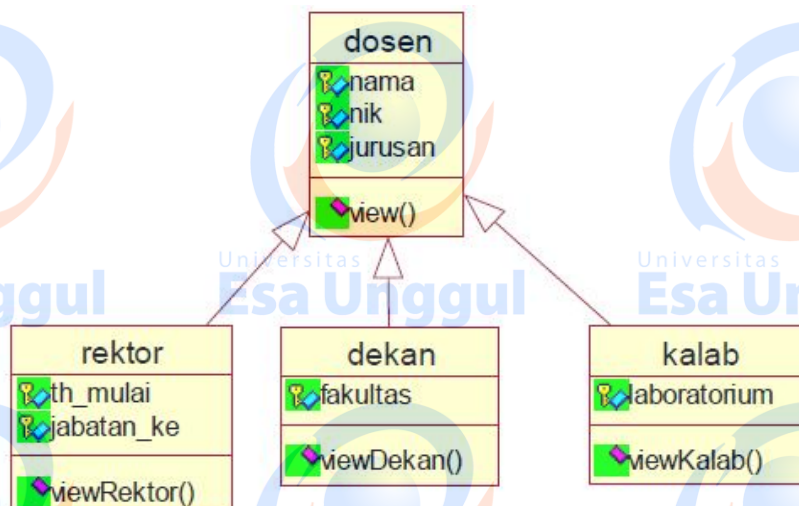
- Sub class, digunakan untuk menunjukkan class anak atau turunan secara hirarkis dari super class.
- Super class, digunakan untuk menunjukkan class induk secara hirarkis dari sub class (class anak).
- Extends, digunakan untuk menunjukkan bahwa suatu class merupakan turunan dari class lain. Misal A extends B, berarti class A adalah turunan dari class B. A merupakan subclass, sedangkan B merupakan superclass.
- Super, digunakan untuk memanggil konstruktor dari superclass atau memanggil variabel yang mengacu pada superclass. Misal `super(x,y,z)`, berarti atribut x, y, dan z diambil dari atribut pada class induk.

Contoh 7.1:

**Gambar 7.1** menunjukkan hirarki klas dosen. Klas dosen tersebut memiliki turunan berupa klas rektor, klas dekan dan klas kalab. Apabila dirancang ke dalam diagram class, akan nampak seperti pada **Gambar 7.2**.



**Gambar 7.1** Hirarki klas dosen.



**Gambar 7.2** Class diagram untuk hirarki dosen.

Pada **Gambar 7.2** tersebut, class induk (class dosen) memiliki atribut nama, nik dan jurusan. Method yang dimiliki oleh class dosen adalah view(). Class turunan dari class dosen ada tiga class. Pada class rektor, terdapat tambahan atribut

berupath\_mulai dan jabatan\_ke, serta method viewRektor(). Pada class dekan terdapat tambahan atribut fakultas, dan method viewDekan(). Pada class kalab terdapat tambahan atribut laboratorium, dan method viewKalab(). Pendefinisian class dosen seperti terlihat pada **Gambar 7.3**.

```

1 // Class dosen sebagai super class
2 package dosen_uui_v1;
3
4 public class dosen {
5     protected String nama;
6     protected String nik;
7     protected String jurusan;
8
9     //Membuat konstruktor
10    dosen (String namaX, String nikX, String jurX)
11    {
12        nama = namaX;
13        nik = nikX;
14        jurusan = jurX;
15    }
16
17    //Menampilkan informasi
18    public void view()
19    {
20        System.out.println("Nama : "+nama);
21        System.out.println("NIK : "+nik);
22        System.out.println("Jurusan: "+jurusan);
23    }
24 }

```

Konstruktor untuk class dosen

**Gambar 7.3** Pendefinisian class dosen pada Contoh 7.1.

Selanjutnya pendefinisian class rektor, dekan, dan kalab seperti terlihat pada **Gambar 7.4 – 7.6**.

```

1 package dosen_uui_v1;
2
3 public class rektor extends dosen {
4     private int th_mulai;
5     private int jabatan_ke;
6
7     //Inisialisasi
8     rektor(String namaX, String nikX, String jurX, int thX, int keX)
9     {
10        super(namaX, nikX, jurX);
11        th_mulai = thX;
12        jabatan_ke = keX;
13    }
14
15    //Menampilkan informasi
16    public void viewRektor()
17    {
18        System.out.println("Th mulai jabatan : "+th_mulai);
19        System.out.println("Jabatan rektor ke- : "+jabatan_ke);
20    }
21 }

```

Memanggil variabel yang mengacu pada dosen (super class)

**Gambar 7.4** Pendefinisian class rektor Contoh 7.1.

```

1 package dosen uii v1:
2
3 public class dekan extends dosen {
4     private String fakultas;
5
6     //Inisialisasi
7     dekan(String namaX, String nikX, String jurX, String fakX)
8     {
9         super(namaX, nikX, jurX);
10        fakultas = fakX;
11    }
12
13    //Menampilkan informasi
14    public void viewDekan()
15    {
16        System.out.println("Fakultas : "+fakultas);
17    }
18 }

```

Gambar 7.5 Pendefinisian class dekan Contoh 7.1.

```

1 package dosen_uui_v1;
2
3 public class kalab extends dosen {
4     private String laboratorium;
5
6     //Inisialisasi
7     kalab(String namaX, String nikX, String jurX, String labX)
8     {
9         super(namaX, nikX, jurX);
10        laboratorium = labX;
11    }
12
13    //Menampilkan informasi
14    public void viewKalab()
15    {
16        System.out.println("Laboratorium : "+laboratorium);
17    }
18 }

```

Gambar 7.6 Pendefinisian class kalab Contoh 7.1.

Program utama untuk pemanggilan fungsi sebagaimana terlihat pada Gambar 7.7.

```

1 /* contoh pewarisan pada class dosen
2 */
3
4 package dosen_uui_v1;
5
6 public class Main {
7
8     public static void main(String[] args) {
9         rektor rek = new rektor("Andi", "885230202", "Informatika", 2006, 2);
10        dekan dek = new dekan("Ahmad", "995230101", "T. Kimia", "TI");
11        kalab lab = new kalab("Indah", "035230302", "Informatika", "RSC");
12
13        rek.view();
14        rek.viewRektor();
15
16        dek.view();
17        dek.viewDekan();
18
19        lab.view();
20        lab.viewKalab();
21    }
22 }

```

Gambar 7.7 Program utama menunjukkan pewarisan kelas dosen.

## Overriding

Overriding adalah suatu cara untuk mendefinisikan ulang method yang ada pada class induk apabila class anak menginginkan adanya informasi yang lain. Overriding dilakukan dengan cara menulis ulang method yang ada pada class induk dengan syarat bahwa nama dan parameter fungsi tersebut harus sama (tidak boleh diubah). Meskipun fungsi telah ditulis ulang oleh class anak, fungsi yang asli pada class induk masih dapat dipanggil di class anak dengan menggunakan class super.

Contoh 7.2:

Konsep pewarisan pada contoh 7.1 dapat dipadukan dengan konsep overriding. **Gambar 7.8** menunjukkan class dosen yang tidak jauh berbeda dengan class dosen pada contoh 7.1.

```
1  /* Contoh penggunaan overriding
2  */
3
4  package dosen_uit_v2;
5
6  public class dosen {
7      protected String nama;
8      protected String nik;
9      protected String jurusan;
10
11     // Inisialisasi
12     dosen (String namaX, String nikX, String jurX)
13     {
14         nama = namaX;
15         nik = nikX;
16         jurusan = jurX;
17     }
18
19     // Menampilkan informasi
20     public void view()
21     {
22         System.out.println("Nama : "+nama);
23         System.out.println("NIK : "+nik);
24         System.out.println("Jurusan: "+jurusan);
25     }
26 }
```

Method yang akan dilakukan

**Gambar 7.8** Pendefinisian class dosen pada Contoh 7.1.

Konsep overriding digunakan pada method view(). Method ini ditulis ulang pada setiap subclass yaitu subclass rektor, dekan, dan kalab dengan menambahkan instruksi tambahan yang dibutuhkan pada setiap class. **Gambar 7.9 – 7.11** menunjukkan class tersebut.

```
1  package dosen_uit_v2;
2
3  public class rektor extends dosen {
4      private int th_mulai;
5      private int jabatan_ke;
6
7     // Inisialisasi
8     rektor(String namaX, String nikX, String jurX, int thX, int keX)
9     {
10         super(namaX, nikX, jurX);
11         th_mulai = thX;
12         jabatan_ke = keX;
13     }
14
15     // Menampilkan informasi
16     public void view()
17     {
18         super.view();
19         System.out.println("Th mulai jabatan : "+th_mulai);
20         System.out.println("Jabatan rektor ke- : "+jabatan_ke);
21     }
22 }
```

Overriding method view() pada super class dosen

**Gambar 7.9** Pendefinisian class rektor Contoh 7.2.



```

1 package dosen_uui_v2;
2
3 public class dekan extends dosen {
4     private String fakultas;
5
6     //Inisialisasi
7     dekan(String namaX, String nikX, String jurX, String fakX)
8     {
9         super(namaX, nikX, jurX);
10        fakultas = fakX;
11    }
12
13    //Menampilkan informasi
14    public void view()
15    {
16        super.view();
17        System.out.println("Fakultas : "+fakultas);
18    }
19 }

```

**Gambar 7.10** Pendefinisian class dekan Contoh 7.2.

```

1 package dosen_uui_v2;
2
3 public class kalab extends dosen {
4     private String laboratorium;
5
6     //Inisialisasi
7     kalab(String namaX, String nikX, String jurX, String lahX)
8     {
9         super(namaX, nikX, jurX);
10        laboratorium = lahX;
11    }
12
13    //Menampilkan informasi
14    public void view()
15    {
16        super.view();
17        System.out.println("Lakoratorium : "+laboratorium);
18    }
19 }

```

**Gambar 7.11** Pendefinisian class kalab Contoh 7.2.

Untuk melakukan pemanggilan, program utama diberikan pada **Gambar 7.12**.

```

1 /* contoh overriding
2 */
3
4 package dosen_uui_v2;
5
6 public class Main {
7
8     public static void main(String[] args) {
9         dosen P;
10        rektor rek = new rektor("Andi", "885230202", "Informatika", 2006, 2);
11        dekan dek = new dekan("Ahmad", "995230101", "T. Kijia", "TI");
12        kalab lab = new kalab("Indah", "005230302", "Informatika", "KSC");
13
14        P = rek;
15        P.view();
16        P = dek;
17        P.view();
18        P = lab;
19        P.view();
20    }
21 }

```

**Gambar 7.12** Program utama menunjukkan konsep overriding kelas dosen.

### Overloading

Overloading fungsi adalah penulisan beberapa fungsi (dua atau lebih) yang memiliki nama yang sama. Pada bahasan overloading dikenal istilah signature. Signature

sebuah fungsi adalah parameter lengkap dengan tipe datanya yang terdapat dalam fungsi tersebut. Misal terdapat fungsi:

```
public int jumlahMahasiswa (int laki2, int perempuan, String kelas);
```

maka signature dari fungsi tersebut adalah (int, int, String).

Suatu fungsi dikatakan di-overload manakala terdapat beberapa fungsi dengan nama yang sama namun memiliki signature yang berbeda-beda, sebagai contoh:

```
public void infoMahasiswa (int laki2, int perempuan, String kelas)
{
    ...
}
public void infoMahasiswa (int mhsLama, int mhsBaru, int mhsCuti, int angkatan)
{
    ...
}
```

Contoh 7.3:

Berikut terdapat class mahasiswa (**Gambar 7.13**). Pada class tersebut terdapat dua subclass dengan nama yang sama yaitu infoMahasiswa namun memiliki signature yang berbeda. Subclass pertama digunakan untuk menghitung jumlah mahasiswa kelas A angkatan 2008. Sedangkan pada subclass kedua digunakan untuk menghitung jumlah mahasiswa aktif sampai tahun 2008.

```
1 package informas_mhs;
2
3 public class mahasiswa {
4
5     //Informasi tentang mahasiswa kelas A
6     public void infoMahasiswa (int laki2, int perempuan, String kelas)
7     {
8         int jumlah = laki2 + perempuan;
9         System.out.println(kelas + ", jumlah mahasiswa = " + jumlah);
10    }
11
12    //Informasi tentang mahasiswa sampai tahun 2008
13    public void infoMahasiswa (int mhsLama, int mhsBaru, int mhsCuti, int angkatan)
14    {
15        int jumlah = mhsLama + mhsBaru + mhsCuti;
16        System.out.println("Sampai tahun " + angkatan + ", jumlah mahasiswa = " + jumlah);
17    }
18 }
```

**Gambar 7.13** Dua class bernama infoMahasiswa dengan signature berbeda.

Pada program utama dilakukan pemanggilan terhadap kedua class tersebut, masing-masing dengan parameter yang sesuai (**Gambar 7.14**).

```

1  /* Contoh overloading
2  */
3
4  package informasimhs;
5
6  public class Main {
7
8      public static void main(String[] args) {
9
10         mahasiswa M = new mahasiswa();
11         M.infoMahasiswa(60, 18, "Kelas A angkatan 2008");
12         M.infoMahasiswa(1000, 400, 25, 2008);
13     }
14 }

```

**Gambar 7.14** Pemanggilan fungsi yang di-overload pada Contoh 7.3.

Overloading fungsi ditentukan oleh signature nya, tidak ditentukan oleh nilai balikan fungsi. Dalam satu class, apabila ada dua atau lebih fungsi yang memiliki nilai balikan yang berbeda namun memiliki signature yang sama, maka fungsi tersebut tidak dapat dikatakan sebagai overloading.

Contoh 7.4:

Berikut terdapat dua class bernama jumlah untuk untuk menjumlahkan dua bilangan. Class jumlah yang pertama memiliki nilai balikan yang bertipe integer untuk signature (int x1, int x2). Sedangkan class jumlah yang kedua memiliki nilai balikan yang bertipe double untuk signature (double y1, double y2). Class yang dibentuk sebagaimana terlihat pada **Gambar 7.15**.

```

1  /* class penjumlahan
2  */
3
4  package jumlahbil;
5
6  public class penjumlahan {
7
8      //Bilangan integer
9      public int jumlah (int x1, int x2)
10     {
11         return(x1+x2);
12     }
13
14     //Bilangan real
15     public double jumlah (double y1, double y2)
16     {
17         return(y1+y2);
18     }
19 }
20 }

```

**Gambar 7.15** Dua class bernama jumlah dengan signature berbeda.

Pada program utama dilakukan pemanggilan terhadap kedua class tersebut, masing-masing dengan parameter yang bersesuaian (**Gambar 7.16**).

```
1  /* Contoh polimorfisme untuk penjumlahan bilangan
2  */
3
4  package jumlahbil;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          int x1=10, x2=15;
10         double y1=10.5, y2=15.8;
11         penjumlahan P = new penjumlahan();
12         System.out.println(x1+" "+x2+" = "+P.jumlah(x1,x2));
13         System.out.println(y1+" "+y2+" = "+P.jumlah(y1,y2));
14     }
15 }
```

Gambar 7.16 Pemanggilan fungsi yang di-overload pada Contoh 7.4.

3. **Latihan**

Lakukan praktek diatas dan lakukan evaluasi

4. **Tugas**

Buatlah sebuah **class kendaraan** dengan turunannya **kedaraan darat**, **kendaraan laut**, kemudian diturunkan lagi menjadi **sepeda motor** dan **perahu layar**.





## Modul 9 dan 10 Polymorphism

### 1. Tujuan Pembelajaran:

Praktikan mampu menerapkan konsep polimorfisme dalam pemrograman dengan Java.

### 2. Teori Singkat

- ✓ Kemampuan untuk memperlakukan object yang memiliki perilaku (bentuk) yang berbeda
- ✓ Implementasi konsep polymorphism:
  1. Overloading: Kemampuan untuk menggunakan nama yang sama untuk beberapa method yang berbeda parameter (tipe dan atau jumlah)
  2. Overriding: Kemampuan subclass untuk menimpa method dari superclass, yaitu dengan cara menggunakan nama dan parameter yang sama pada method

### Polymorphism – Overloading

```
class Lingkaran{
    void gambarLingkaran(){
    }
    void gambarLingkaran(int diameter){
    ...
    }
    void gambarLingkaran(double diameter){
    ...
    }
    void gambarLingkaran(int diameter, int x, int y){
    ...
    }
    void gambarLingkaran(int diameter, int x, int y, int warna, String namaLingkaran){
    ...
    }
}
```

```
public class Sepeda{
    int gir;
    void setGir(int penambahanGir) {
        gir= gir+ penambahanGir;
    }
    int getGir() {
        return gir;
    }
}
```

SepedaGunung.java

```
public class SepedaGunung extends Sepeda{
    void setGir(int penambahanGir) {
        super.setGir(pertambahanGir);
        gir = 2*getGir();
    }
}
```

SepedaGunungBeraksi.java

```
public class SepedaGunungBeraksi {
    public static void main(String[] args) {
        SepedaGunung sg=new SepedaGunung();
        sg.setGir(2);
        System.out.println(sg.getGir());
        sg.setGir(3);
        System.out.println(sg.getGir());
    }
}
```

## LATIHAN: OVERLOADING PADA MATEMATIKA

1. Kembangkan class Matematika, MatematikaCanggih dan MatematikaBeraksi
2. Lakukan overloading pada Method yang ada (pertambahan, pengurangan, perkalian, pembagian)
3. Tambahkan method baru bertipe data double (pecahan) dan memiliki 3 parameter
4. Uji di kelas MatematikaBeraksi dengan parameter pecahan: 12.5, 28.7, 14.2
5. Misalnya:  
pertambahan(12.5, 28.7, 14.2)      perkalian(12, 28, 14)  
pengurangan(23, 34, 5.5)      pembagian(3.4, 4.9, 1.2)

Universitas  
**Esa Unggul**  
Matematika.java

```
public class Matematika{
    void pertambahan (int a, int b){
        int hasil= a + b;
        System.out.println("hasil:" + hasil);
    }
    void pertambahan (double a, double b,
        double c){
        double hasil= a + b + c;
        System.out.println("hasil:" + hasil);
    }
    ...
}
```

Polimorfisme digunakan untuk menyatakan suatu nama yang merujuk pada beberapa fungsi yang berbeda (Sinaga, 2004). Pada polimorfisme, rujukan dapat dilakukan pada berbagai tipe objek. Hal ini dilakukan karena setiap objek dimungkinkan memiliki instruksi yang berbeda. Dalam mengimplementasikan polimorfisme, perlu diperhatikan hal-hal sebagai berikut (Rickyanto, 2005):

1. Method yang dipanggil harus melalui variabel dari super class.
2. Method yang dipanggil juga harus merupakan method yang ada pada super class.
3. Signature method harus sama baik yang ada pada super class maupun di subclass.
4. Method access attribute pada subclass tidak boleh lebih terbatas daripada yang ada pada super class.

Contoh 8.1:

Pada **Gambar 8.1** merupakan program untuk membangun class kendaraan. Pada class kendaraan mewaris ke tiga class, yaitu class pesawat, mobil, dan kapal.

**Gambar 8.2 – 8.4** menunjukkan pembentukan ketiga subclass tersebut.

```
1  /* super class kendaraan
2  */
3
4  package transportasi;
5
6  public class kendaraan {
7      private String model;
8
9      //Inisialisasi:
10     public kendaraan (String model)
11     {
12         this.model = model;
13     }
14
15     //Informasi yang merupakan method tanpa instruksi
16     public void informasi (){}
17
18 }
```

**Gambar 8.1** Pendefinisian class kendaraan pada Contoh 8.1.

```
1  /* Untuk kelas pesawat
2  */
3  package transportasi;
4
5  public class pesawat extends kendaraan{
6      private String nama;
7      private String jenis;
8
9      public pesawat (String nama)
10     {
11         super("Pesawat");
12         this.nama = nama;
13         jenis = "belum teridentifikasi";
14     }
15
16     public pesawat (String nama, String jenis)
17     {
18         super("Pesawat");
19         this.nama = nama;
20         this.jenis = jenis;
21     }
22
23     public void informasi()
24     {
25         System.out.println("Nama pesawat adalah "+nama);
26         System.out.println("Jenis pesawat adalah "+jenis);
27     }
28 }
```

**Gambar 8.2** Pendefinisian class pesawat Contoh 8.1.

```
1  /* Untuk kelas mobil
2  */
3  package transportasi;
4
5  public class mobil extends kendaraan {
6      private String nama;
7      private String jenis;
8
9      public mobil (String nama)
10     {
11         super("Mobil");
12         this.nama = nama;
13         jenis = "belum teridentifikasi";
14     }
15
16     public mobil (String nama, String jenis)
17     {
18         super("Mobil");
19         this.nama = nama;
20         this.jenis = jenis;
21     }
22
23     public void informasi()
24     {
25         System.out.println("Nama mobil adalah "+nama);
26         System.out.println("Jenis mobil adalah "+jenis);
27     }
28 }
```

**Gambar 8.3** Pendefinisian class mobil Contoh 8.1.

```

1  /* Untuk kelas kapal
2  */
3  package transportasi;
4
5  public class kapal extends kendaraan {
6      private String nama;
7      private String jenis;
8
9      public kapal(String nama)
10     {
11         super("Kapal");
12         this.nama = nama;
13         jenis = "belum teridentifikasi";
14     }
15
16     public kapal(String nama, String jenis)
17     {
18         super("Kapal");
19         this.nama = nama;
20         this.jenis = jenis;
21     }
22
23     public void informasi()
24     {
25         System.out.println("Nama kapal adalah "+nama);
26         System.out.println("Jenis kapal adalah "+jenis);
27     }
28 }

```

**Gambar 8.4** Pendefinisian class kapal Contoh 8.1.

Pada program utama dilakukan pemanggilan terhadap ketiga class tersebut, masing-masing dengan parameter yang bersesuaian (**Gambar 8.5**).

```

1  //Contoh polimorfisme untuk alat2 transportasi
2
3  package transportasi;
4
5  public class Main {
6
7      public static void main(String[] args) {
8          kendaraan P;
9          pesawat psw = new pesawat("Boeing 707", "Pesawat Komersial");
10         mobil mb11 = new mobil("Toyota Kijang", "Jeep");
11         mobil mb12 = new mobil("Suzuki Baleno", "Sedan");
12         mobil mb13 = new mobil("VW Combi");
13         kapal kpl = new kapal("Queen Mary 2", "Kapal Pesiar");
14
15         P = psw;
16         P.informasi();
17         P = mb11;
18         P.informasi();
19         P = mb12;
20         P.informasi();
21         P = mb13;
22         P.informasi();
23         P = kpl;
24         P.informasi();
25     }
26 }

```

**Gambar 8.5** Program utama untuk contoh 8.1.

### 3. Latihan

Lakukan praktek diatas dan lakukan evaluasi

### 4. Tugas

Buatlah semua program yang mengimplemtasikan konsep polymorphisme dari sebuah class dengan nama class hewan



## Modul 11 dan 12 Penanganan Eksepsi

### 1. Tujuan Pembelajaran:

Praktikan mampu menangani berbagai kesalahan dengan penanganan eksepsi dalam pemrograman dengan Java.

Saat kita membuat program, sebisa mungkin program kita terhindar dari kesalahan. Namun, yang lebih penting adalah bagaimana dalam program kita dapat mengantisipasi kemungkinan munculnya kesalahan pada saat program kita dieksekusi. Java menyediakan sebuah mekanisme penanganan kesalahan yang biasa disebut dengan *exception-handling*. Dalam hal ini setiap kesalahan akan dibentuk menjadi objek.

### 2. Teori Singkat

Dalam Java, *runtime error* (kesalahan-kesalahan yang terjadi pada saat program sedang berjalan) direpresentasikan dengan eksepsi. Eksepsi adalah suatu objek yang dibuat pada saat program mengalami suatu kondisi yang tidak wajar (*abnormal*). Eksepsi dapat dibangkitkan secara otomatis oleh *Java runtime* maupun secara manual oleh kita sendiri melalui kode yang kita tulis.

Java menyediakan lima buah kata kunci untuk menangani eksepsi, yaitu: *try*, *catch*, *throw*, *throws*, dan *finally*. Kata kunci *try* digunakan untuk membuat blok yang berisi statemen-statemen yang mungkin menimbulkan eksepsi. Jika dalam proses eksekusi runtunan statemen tersebut terjadi sebuah eksepsi, maka eksepsi akan dilempar ke bagian blok penangkap yang dibuat dengan kata kunci *catch*. Pada kasus-kasus tertentu, terkadang kita juga ingin melempar eksepsi secara manual. Untuk itu, kita harus menggunakan kata kunci *throw*. Jika kita ingin membangkitkan sebuah eksepsi tanpa menuliskan blok *try*, maka kita perlu menambahkan kata kunci *throws* pada saat pendeklarasian *method*. Dalam mendefinisikan blok *try*, kita dapat menuliskan statemen tambahan, yaitu menggunakan kata kunci *finally*. Statemen tambahan ini dipastikan dieksekusi baik terjadi eksepsi maupun tidak.

Berikut ini bentuk umum penanganan eksepsi di dalam Java.

```
try {
    //kumpulan statemen yang mungkin menimbulkan
    eksepsi
} catch (TipeEksepsi1 objekEksepsi1) {
    //penanganan untuk tipe eksepsi1
} catch (TipeEksepsi2 objekEksepsi2) {
    //penanganan untuk tipe eksepsi2
}
...
finally {
    //statemen tambahan yang pasti akan dieksekusi
}
```

Di sini, *TipeEksepsi* adalah tipe dari eksepsi yang terjadi. Tipe tersebut direpresentasikan dengan sebuah kelas, misalnya: *NullPointerException*, *ArithmeticException*, *ArrayIndexOutOfBoundsException*, dan sebagainya.

Perhatikan kode program berikut ini:

### Program 9.1 Contoh program yang menimbulkan eksepsi

```
class ContohEksepsi1 {
    public static void main(String[] args) {
        int[] arrayInteger = new int[5];
        // SALAH, karena tidak terdapat indeks ke-7
        arrayInteger[7] = 9;
    }
}
```

Jika dijalankan, program di atas akan membangkitkan eksepsidengan tipe *ArrayIndexOutOfBoundsException* karena kitamengakses indeks yang tidak terdapat di dalam array A. ArrayA terdiri dari 5 elemen dengan indeks 0 sampai 4. Dengandemikian, Jika kita mengakses indeks ke-7 maka Java akanmemberikan pesan kesalahan berikut:



```
Output - ModulPraktikumPBO (run)
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 7
    at KelasDanObjek.ContohEksepsi1.main(ContohEksepsi1.java:18)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Gambar 9.1** Pesan kesalahan hasil eksekusi program 9.1

Ini artinya, indeks 7 menimbulkan eksepsi *ArrayIndexOutOfBoundsException*. Kode yang menimbulkankesalahan tersebut terdapat pada *method main()* di dalam kelas *ContohEksepsi1*, yang tersimpan dalam file *ContohEksepsi1.java*, aris ke-18.

Tampak pada contoh di atas bahwa ketika terjadi sebuah eksepsi maka program akan dihentikan secara tidak normal. Oleh karena itu, kita perlu mendefinisikan sebuah mekanisme yang dapat menangani kesalahan tersebut. Mekanisme inilah yang dinamakan dengan *exception-handling*.

#### Menggunakan Kata Kunci *try* dan *catch*

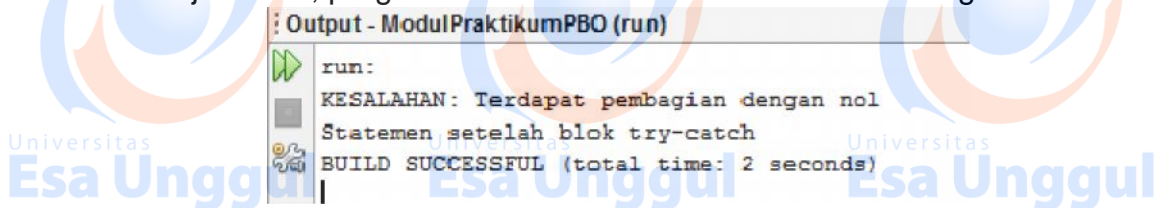
Pada bagian sebelumnya telah dicontohkan bagaimana Javamencegah eksepsi secara *default*. Namun, pada umumnya, pencegahan eksepsi dilakukan oleh kita sendiri, sebagai *programmer*. Untuk melakukan hal tersebut, kita dapat menggunakan blok *try-catch*. Berikut ini contoh program yang menggunakan blok *try-catch* untuk menangani kesalahan yang muncul.

### Program 9.2 Contoh implementasi blok *try-catch*

```
class ContohEksepsi2 {
    public static void main(String[] args) {
        int pembilang = 7;
        int penyebut = 0;
        try {
            int hasil = pembilang / penyebut; // SALAH
            // tidak dieksekusi
            System.out.println("Hasil = " + hasil);
        } catch (Exception e) {
            System.out.println("KESALAHAN: "
                + "Terdapat pembagian dengan nol");
        }
        System.out.println("Statemen setelah blok
```

```
trycatch" );
    }
}
```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:



Gambar 9.2 Hasil eksekusi program 9.2

Pada program di atas, pada saat muncul eksepsi, program akan langsung melompat ke bagian *catch*. Namun bagian statemen yang dituliskan setelah blok *try-catch* tetap akan dieksekusi oleh program.

Dengan menggunakan blok *try-catch* seperti contoh-contoh di atas, meskipun di dalam program sebenarnya terkandung kesalahan, namun proses eksekusi program tetap dapat dilanjutkan, tidak dihentikan secara tiba-tiba. Inilah yang disebut dengan proses penjebakan kesalahan atau penjebakan eksepsi.

### Penjebakan Beberapa Tipe Eksepsi

Pada kenyataan di lapangan, sebuah blok *try-catch* biasanya terdiri atas beberapa bagian blok penangkap. Hal tersebut bertujuan agar dalam satu blok *try-catch*, kita dapat mengatasi beberapa kemungkinan eksepsi yang terjadi. Berikut ini contoh program yang akan menunjukkan bagaimana cara mendefinisikan sebuah blok *try-catch* yang dapat digunakan untuk menjebak beberapa tipe eksepsi.

#### Program 9.3 Contoh penjebakan beberapa tipe eksepsi

```
class ContohMultiEksepsi {
    public static void cobaEksepsi(int pembilang, int
    penyebut) {
        try {
            int hasil = pembilang / penyebut;
            System.out.println("Hasil bagi: " + hasil);
            int[] Arr = {1, 2, 3, 4, 5}; // array dengan 5
            elemen
            Arr[ 10] = 23; // mengakses indeks ke-10
        } catch (ArithmeticException eksepsi1) {
            System.out.println(
                "Terdapat pembagian dengan 0");
        } catch (ArrayIndexOutOfBoundsException eksepsi2)
        {
            System.out.println("Indeks di luar rentang");
        }
    }

    public static void main(String[] args) {
        cobaEksepsi(4, 0); // menimbulkan
        rithmeticException
    }
}
```

```

        System.out.println();
    // menimbulkan ArrayIndexOutOfBoundsException
        cobaEksepsi(12, 4);
    }
}

```

Hasil eksekusi program di atas adalah sebagai berikut:

```

: Output - ModulPraktikumPBO (run)
run: Universitas
Terdapat pembagian dengan 0
Hasil bagi: 3
Indeks di luar rentang
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Gambar 9.3** Hasil eksekusi program 9.3

Tampak pada hasil di atas bahwa ketika kita melewati argumen ke dalam *method test()* dengan nilai 4 dan 0, program akan membangkitkan eksepsi *ArithmeticException*. Sedangkan jika argumen bernilai 12 dan 4, maka program akan melakukan pembagian dengan benar dan menghasilkan nilai 3 (hasil dari 12/4).

Namun, pada saat mengeksekusi statemen:

```

Arr[10] = 23;

```

program akan membangkitkan eksepsi *ArrayIndexOutOfBoundsException - Exception*. Hal ini disebabkan karena array *Arr* hanya terdiri dari 5 buah elemen.

### Menggunakan Kata Kunci *throw*

Kita dapat membangkitkan eksepsi secara manual dengan menggunakan kata kunci *throw*. Berikut ini bentuk umum penggunaan kata kunci tersebut.

```

throw eksepsi;

```

eksepsi yang dimaksud harus berupa objek *Throwable* maupun objek dari kelas-kelas turunannya. Kita tidak dapat melempar objek non-*Throwable*, misalnya objek *String*. Sebagai contoh, jika kita ingin membangkitkan eksepsi *NullPointerException*, maka kita dapat menuliskan kodenya sebagai berikut:

```

throw NullPointerException();

```

Program berikut akan menunjukkan bagaimana cara menggunakan kata kunci *throw* untuk membangkitkan eksepsi *NullPointerException*.

### Program 9-4 Contoh pengebakan eksepsi dengan kata kunci *throw*

```

class Mahasiswa {
    private String nim;
    private String nama;
    public void setNIM(String inputNIM) {
        try {

```



```

        nim = inputNIM;
        if (inputNIM == null) {
            throw new NullPointerException();
        }
    } catch (NullPointerException npe) {
        System.out.println("KESALAHAN: "
            + "NIM tidak boleh null");
    }
}

public String getNIM() {
    return nim;
}

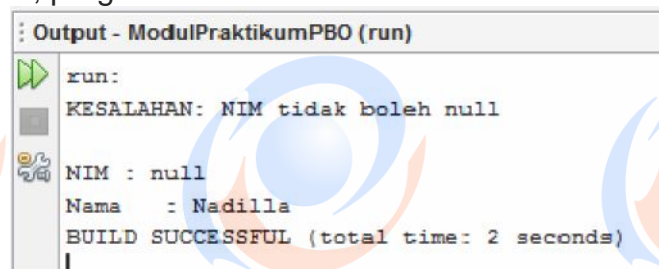
public void setName(String inputNama) {
    try {
        nama = inputNama;
        if (nama == null) {
            throw new NullPointerException();
        }
    } catch (NullPointerException npe) {
        System.out.println("KESALAHAN: " + "Nama
mahasiswa tidak boleh null");
    }
}

public String getName() {
    return nama;
}
}

class DemoThrow {
    public static void main(String[] args) {
        Mahasiswa mhs = new Mahasiswa();
        mhs.setNIM(null);
        mhs.setName("Nadilla");
        System.out.println("\nNIM : " + mhs.getNIM());
        System.out.println("Nama : " + mhs.getName());
    }
}
}

```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:



```

: Output - ModulPraktikumPBO (run)
run:
KESALAHAN: NIM tidak boleh null
NIM : null
Nama : Nadilla
BUILD SUCCESSFUL (total time: 2 seconds)

```

Gambar 9.4 Hasil eksekusi program 9.4

Tampak pada hasil di atas, ketika kita melewatkan nilai *null* kedalam *method setNIM()* dari kelas *Mahasiswa*, program akan membangkitkan eksepsi *NullPointerException* yang kita lempar atau kita bangkitkan secara manual dengan menggunakan kata kunci *throw*.

### Menggunakan Kata Kunci *throws*

Jika kita tidak menyertakan blok *try-catch* di dalam *method* yang mengandung kode-kode yang mungkin menimbulkan eksepsi, maka kita harus menyertakan klausa *throws* pada saat deklarasi *method* yang bersangkutan. Jika tidak, maka program tidak dapat dikompilasi. Berikut ini bentuk umum dari penggunaan kata kunci *throws*.

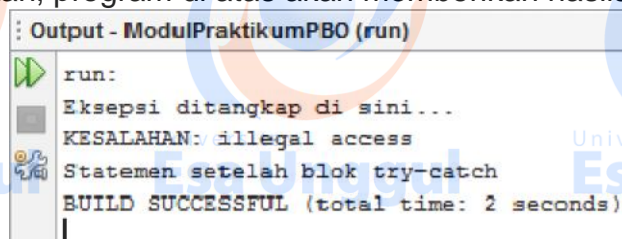
```
 tipe nama-method(daftar-parameter) throws tipe-eksepsi1,  
 tipe-eksepsi2, ..., tipe-eksepsiN {  
 //badan-method  
 }
```

Berikut ini contoh program yang menggunakan kata kunci *throws* untuk menangani eksepsi yang muncul.

**Program 9.5** Contoh pengebakan eksepsi dengan kata kunci *throws*

```
Class DemoThrows {  
    public static void cobaEksepsi() throws  
        IllegalAccessException {  
        throw new IllegalAccessException(  
            "KESALAHAN: illegal  
access");  
    }  
    public static void main(String[] args)  
    {  
        try {  
            cobaEksepsi();  
        } catch (Exception e) {  
            System.out.println("Eksepsi  
ditangkap di sini...");  
            System.out.println(e.getMessage());  
        }  
        System.out.println("Statemen  
setelah blok trycatch");  
    }  
}
```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:



```
Output - ModulPraktikumPBO (run)  
run:  
Eksepsi ditangkap di sini...  
KESALAHAN: illegal access  
Statemen setelah blok trycatch  
BUILD SUCCESSFUL (total time: 2 seconds)
```

**Gambar 9.5** Hasil eksekusi program 9.5

Perhatikan deklarasi *method cobaEksepsi()* di atas. Di situ kita menambahkan klausa *throws* yang diikuti dengan nama eksepsi yang akan dilempar/dibangkitkan. Melalui teknik seperti ini, *compiler* akan mengetahui bahwa *method* tersebut dapat membangkitkan eksepsi *IllegalAccessException* sehingga program dapat dikompilasi dengan sukses.

### Menggunakan Kata Kunci *finally*

Terkadang kita ingin menempatkan kode yang pasti akan dieksekusi, baik terjadi eksepsi maupun tidak. Untuk melakukan hal itu, kita perlu menempatkan kode tersebut di dalam bagian finalisasi dari blok *try-catch*. Blok finalisasi ini dibuat dengan menggunakan kata kunci *finally*. Bentuk umum pendefinisian blok *finally* ini sebagai berikut:

```
try
{
//statemen yang mungkin menimbulkan eksepsi A, B, dan C
} catch (A ea) {
//blok penangkap untuk eksepsi A
} catch (B eb) {
//blok penangkap untuk eksepsi B
} catch (C ec) {
//blok penangkap untuk eksepsi C
} finally {
//statemen yang pasti akan dieksekusi, baik
terjadi
//eksepsi maupun tidak
}
```

Untuk mengetahui cara kerja dari kata kunci *finally*, perhatikan program berikut ini.

### Program 9.6 Contoh penggunaan kata kunci *finally*

```
class DemoFinally {
    public static void cobaEksepsi(int pembilang, int
penyebut) {
        try {
            int hasil = pembilang / penyebut;
            System.out.println("Hasil bagi: " + hasil);
            int[] Arr = {1, 2, 3, 4, 5}; // array dengan 5
elemen
            Arr[10] = 23; // mengakses indeks ke-10
        } catch (ArithmeticException eksepsi1) {
            System.out.println("Terdapat pembagian dengan
0");
        } catch (ArrayIndexOutOfBoundsException eksepsi2)
{
            System.out.println("Indeks di luar rentang");
        } finally {
            System.out.println("Ini adalah statemen dalam
blok finally");
        }
    }
}
```

```

    }

    public static void main(String[] args) {
        cobaEksepsi(4, 0); //
        menimbulkanArithmeticException
        System.out.println();
        cobaEksepsi(12, 4); // menimbulkan
        ArrayIndexOutOfBoundsException
    }
}

```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:

```

: Output - ModulPraktikumPBO (run)
run:
Terdapat pembagian dengan 0
Ini adalah statemen dalam blok finally
Hasil bagi: 3
Indeks di luar rentang
Ini adalah statemen dalam blok finally
BUILD SUCCESSFUL (total time: 2 seconds)

```

Gambar 9.6 Hasil eksekusi program 9.6

### 3. Latihan

Lakukan praktek diatas dan lakukan evaluasi

### 4. Tugas

Buatlah program penyisipan array dan gunakan blok try-catch untuk menangani kesalahannya. Contoh

1	2
---	---

Sisipkan 3 di indeks ke-1 maka array berubah

1	3	2
---	---	---



## Modul 13 dan 14 Input dan Output

### 1. Tujuan Pembelajaran:

*Praktikan mampu mengenali dan memahami input dan output (I/O) dalam pemrograman dengan Java.*

Java menyediakan dukungan terhadap proses I/O dengan menghadirkan paket *java.io*. Di dalam paket tersebut tersimpan banyak kelas dan *interface* yang akan memudahkan kita, sebagai *programmer*, dalam pengambilan dan penyimpanan informasi dari/ke media lain (misalnya: *file*). Bab ini akan memperkenalkan beberapa kelas yang terdapat pada paket *java.io* yang diperlukan untuk melakukan proses input dan output di dalam Java.

### 2. Dasar-Dasar I/O

#### Pengertian *Stream*

Program Java melakukan proses I/O melalui *stream*. *Stream* adalah sebuah abstraksi yang dapat memberikan atau mendapatkan informasi. *Stream* dapat dihubungkan dengan peralatan fisik yang terdapat dalam system I/O Java, seperti: *keyboard*, *file*, layar *console*, dan yang lainnya. Cara kerja *stream* selalu sama, meskipun jenis peralatan yang terhubung dengan *stream* tersebut berbeda. Ini artinya, sebuah *stream* input dapat mengabstraksikan beberapa tipe peralatan fisik, seperti: *keyboard*, *file*, atau *socket* jaringan. Begitu pula dengan *stream* output, yang dapat dihubungkan dengan layar *console*, *file*, maupun koneksi jaringan. Dengan demikian, *stream* akan memudahkan kita dalam melakukan proses I/O, karena kode program yang kita tulis akan sama untuk masing-masing peralatan fisik yang dihubungkan dengan *stream* bersangkutan.

#### Tipe *Stream*

Terdapat dua buah tipe *stream* yang dapat digunakan, yaitu: *stream byte* dan *stream karakter*. Sesuai dengan namanya, *stream byte* digunakan untuk memberikan atau menyimpan informasi data dalam bentuk *byte*, misalnya untuk menulis dan membaca *file* biner. Sedangkan *stream karakter* digunakan untuk melakukan proses I/O yang melibatkan data-data dalam bentuk karakter, misalnya pada saat kita melakukan proses baca/tulis ke *file* teks. Dalam Java, *stream* didefinisikan dengan menggunakan empat kelas abstrak, yaitu: *InputStream*, *OutputStream*, *Reader*, dan *Writer*. Kelas *InputStream* dan *OutputStream* merupakan kelas abstrak yang dirancang sebagai kelas induk atau *superclass* untuk kelas-kelas yang termasuk ke dalam kategori *stream byte*. Adapun kelas *Reader* dan *Writer* merupakan kelas abstrak yang akan diturunkan menjadi kelas-kelas baru yang termasuk ke dalam kategori *stream karakter*.

Melalui proses pewarisan (*inheritance*), semua kelas yang diturunkan dari *InputStream* maupun *Reader* akan memiliki method *read()*, yang akan digunakan untuk proses pembacaan data. Adapun semua kelas yang diturunkan dari *OutputStream* maupun *Writer* akan memiliki method *write()*, yang akan digunakan untuk proses penulisan data. Kedua method tersebut dideklarasikan sebagai method abstrak sehingga harus diimplementasi oleh setiap kelas turunannya.

#### Melakukan *Input*

Dalam Java, input *console* dilakukan melalui pembacaan terhadap *stream System.in*. Untuk mendapatkan karakter-karakter yang dimasukkan melalui *keyboard* ke dalam

layarconsole, kita perlu membungkus *System.in* di dalam objek *BufferedReader*. Hal ini, dilakukan untuk membentuk *stream* karakter karena *System.in* sebenarnya merupakan *stream byte*. Adapun bentuk *constructor* dari *BufferedReader* sebagai berikut:

```
BufferedReader(Reader inputReader)
```

*inputReader* adalah *stream* yang akan dihubungkan dengan *instance* atau objek dari kelas *BufferedReader* yang dibuat. Karena *Reader* berupa kelas abstrak, maka kita perlu mencari kelas turunannya yang berupa kelas konkrit. Salah satunya adalah kelas *InputStreamReader*, yang dapat mengkonversi *byte* ke karakter. Sekarang, agar objek dari *InputStreamReader* dapat dihubungkan dengan *System.in*, kita perlu menggunakan bentuk *constructor* sebagai berikut:

```
InputStreamReader(InputStream inputStream)
```

Di sini, *inputStream* dapat kita isi dengan *System.in*. Dengan demikian, untuk membuat objek *BufferedReader* yang dapat terhubung dengan *keyboard*, kita perlu menggunakan kode berikut:

```
InputStreamReader isr = new  
InputStreamReader(System.in);  
BufferedReader br = new BufferedReader(isr);
```

Sampai tahap ini, objek *br* sudah siap digunakan untuk melakukan proses input, yaitu dengan melakukan pemanggilan terhadap method *read()* maupun *readline()*, yang akan kita bahas lebih lanjut.

### Membaca Input Data Karakter

Untuk membaca input berupa karakter, kita bisa menggunakan method *read()* yang terdapat pula pada kelas *BufferedReader*. Berikut ini contoh program yang akan menunjukkan bagaimana menangani proses input data karakter.

#### Program 10.1 Program dengan proses input data karakter

```
import java.io.*;  
class DemoInputKarakter {  
    public static void main(String[] args) throws  
        IOException {  
        System.out.print("Masukkan sebuah karakter: ");  
        char KarakterInput;  
        InputStreamReader isr = new  
        InputStreamReader(System.in);  
        BufferedReader br = new BufferedReader(isr);  
        KarakterInput = (char) br.read();  
        System.out.println("Karakter yang dimasukkan  
adalah '\" + KarakterInput + '\"");  
    }  
}
```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:

```
Output - ModulPraktikumPBO (run)
run:
Masukkan sebuah karakter: B
Karakter yang dimasukkan adalah 'B'
BUILD SUCCESSFUL (total time: 5 seconds)
```

Gambar 10.1 hasil eksekusi program 10.1

### Membaca Input Data *String*

Untuk membaca input berupa *string*, kita bisa menggunakan method *readLine()*, bukan method *read()*.

Berikut ini contoh program yang akan menunjukkan bagaimana menangani proses input data *string*.

### Program 10.2 Program dengan proses input data *string*

```
import java.io.*;
class DemoInputString {
    public static void main(String[] args) throws
        IOException {
        System.out.print("Masukkan nama Anda: ");
        String namaInput;
        InputStreamReader isr = new
        InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        namaInput = br.readLine();
        System.out.println("Halo " + namaInput + "...");
        System.out.println("Semoga Anda Sukses Selalu..");
    }
}
```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:

```
Output - ModulPraktikumPBO (run)
run:
Masukkan nama Anda: Aurel
Halo Aurel...
Semoga Anda Sukses Selalu..
BUILD SUCCESSFUL (total time: 9 seconds)
```

Gambar 10.2 Hasil eksekusi program 10.2

### Membaca Input Data Numerik

Untuk membaca input berupa data numerik, kita bisa menggunakan method *readLine()* seperti pada saat menangani input data *string*, dan selanjutnya *string* hasil input tersebut dikonversi ke tipe numerik.

Berikut ini contoh program yang akan menunjukkan bagaimana menangani proses input data numerik.

### Program 10.3 Program dengan proses input data numerik

```
import java.io.*;
class DemoInputNumerik {
    public static void main(String[] args) throws
```

```

        IOException {
            System.out.print("Masukkan sebuah bilangan bulat :
");
            String temp;
            int bilanganInput = 0;
            InputStreamReader isr = new
InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);
// input data dianggap sebagai string
            temp = br.readLine();
            try {
// konversi dari string ke integer
                bilanganInput = Integer.parseInt(temp);
            } catch (NumberFormatException nfe) {
                System.out.println("Nilai yang dimasukkan "
                    + "bukan bilangan bulat");
                System.exit(1);
            }
            System.out.println("Bilangan yang anda masukkan
adalah " + bilanganInput);
        }
    }
}

```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:

```

: Output - ModulPraktikumPBO (run)
run:
Masukkan sebuah bilangan bulat : 7
Bilangan yang anda masukkan adalah 7
BUILD SUCCESSFUL (total time: 6 seconds)

```

Gambar 10.3 Hasil eksekusi program 10.3

### Dasar-Dasar Baca Tulis File

Java menyediakan dua buah *stream* yang paling sering digunakan untuk melakukan proses pembacaan/penulisan data dari/ke dalam *file*, yaitu: *FileInputStream* (untuk membaca data) dan *FileOutputStream* (untuk menulis data). Keduanya akan membentuk *stream byte* yang terhubung ke sebuah *file*. Untuk membuka *file*, kita harus membentuk objek dari salah satu kelas *stream* tersebut dengan menyertakan nama *file* sebagai argumen pada *constructor*-nya. Bentuk secara umum adalah sebagai berikut:

```

FileInputStream(String filename) throws
FileNotFoundException
FileOutputStream(String filename) throws
FileNotFoundException

```

Di sini, *fileName* menunjukkan nama *file* yang akan dibuka. Pada saat menggunakan *stream input*, jika *file* tidak ditemukan maka kedua *constructor* akan membangkitkan eksepsi *FileNotFoundException*, sedangkan pada saat menggunakan *stream output*, eksepsi tersebut akan muncul jika *file* output tidak dapat terbentuk.

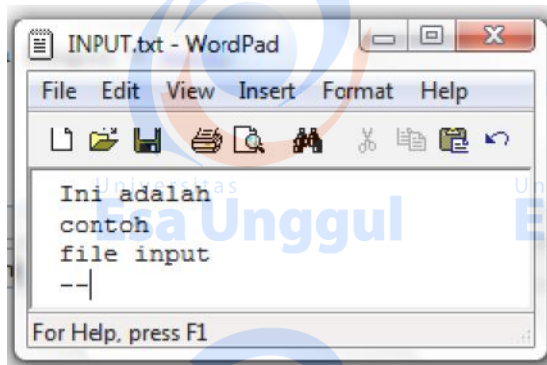


Berikut ini contoh program yang akan menunjukkan proses pembacaan data dari sebuah *file* dan menampilkan isi *file* tersebut ke layar *console*.

**Program 10.4** Program simulasi pembacaan data dari *file*

```
import java.io.*;
class DemoBacaFile {
    public static void main(String[] args) {
        FileInputStream finput = null;
        int data;
        // membuka file
        try {
            finput = new FileInputStream("d:/INPUT.TXT");

        } catch (FileNotFoundException fnfe) {
            System.out.println("File tidak ditemukan.");
            return; // keluar dari method
        }
        // membaca data dari dalam file
        // dan menampilkan hasilnya ke layar console
        try {
            while ((data = finput.read()) != -1) {
                // ketika ditampilkan, data dikonversi ke tipe char
                System.out.print((char) data);
            }
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
            return;
        }
        // menutup file
        try {
            finput.close();
        } catch (IOException ioe) {
        }
    }
}
```



**Gambar 10.4** File input yang akan dibaca melalui program 10.4

Hasil eksekusi program 10.4 dalam membaca file teks *INPUT.txt* tersebut di atas adalah sebagai berikut:

```
Output - ModulPraktikumPBO (run)
run:
Ini adalah
contoh
file input
--BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 10.5 Hasil eksekusi program 10.4

Selanjutnya akan dibahas tentang proses penulisan data kedalam *file*. Untuk melakukan hal ini, kita perlu menggunakan method *write()*. Bentuk umum dari method *write()* sebagaiberikut:

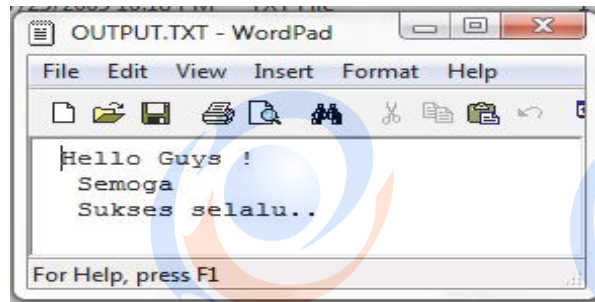
```
void write(int nilaiByte)
```

Berikut ini contoh program yang akan menunjukkan penggunaan method *write()* di atas.

**Program 10.5** Program penulisan data ke sebuah *file*

```
import java.io.*;
class DemoTulisFile {
    public static void main(String[] args) {
        FileOutputStream foutput = null;
        String data = "Hello Guys ! \n Semoga \n Sukses
selalu..";
        // membuka file
        try {
            foutput = new FileOutputStream("d:/OUTPUT.TXT");
        } catch (FileNotFoundException fnfe) {
            System.out.println("File tidak dapat
terbentuk.");
            return; // keluar dari method
        }
        // menulis data ke dalam file
        try {
            for (int i = 0; i < data.length(); i++) {
                // data akan dikonversi per karakter
                foutput.write((int) data.charAt(i));
            }
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
            return;
        }
        // menutup file
        try {
            foutput.close();
        } catch (IOException ioe) {
        }
    }
}
```

Jika program tersebut di atas dijalankan, maka akan menghasilkan *file* output sebagai berikut:



Gambar 10.6 File output hasil eksekusi program 10.5

### 3. Latihan

Lakukan praktek diatas dan lakukan evaluasi

### 4. Tugas

Buatlah sebuah program aplikasi file yang bisa melakukan beberapa operasi berikut :

- Buat File dengan dua ekstensi .txt dan .dat
- Mengecek File dalam direktory tertentu
- Mengisi File
- Menampilkan isi File
- Merename nama File
- Menghapus File

