

UNIVERSITAS INDRAPRASTA PGRI  
FAKULTAS TEKNIK, MATEMATIKA DAN ILMU PENGETAHUAN ALAM  
TEKNIK INFORMATIKA

# DIKTAT: PEMROGRAMAN BERORIENTASI OBJEK

---

Pemrograman Java I

**Forkas Tiroy Santos Butarbutar, M.Kom**

30/10/2014



"Aksi diinisiasi di dalam pemrograman berorientasi obyek dengan mengirimkan sebuah pesan ke suatu agen (obyek) untuk merespon aksi. Pesan menkodekan permintaan untuk sebuah aksi dan biasanya ditemani oleh beberapa informasi (argument) yang diperlukan dalam permintaan tersebut. Pihak penerima adalah obyek tujuan dimana pesan akan dikirim. Jika penerima telah menerima pesan, pihak ini akan bertanggungjawab untuk menjalankan aksi yang berkaitan. Untuk merespon pesan, pihak penerima akan menjalankan metode untuk memenuhi permintaan tersebut."



## **Kata Pengantar**

Perkembangan teknologi dan informasi dewasa ini semakin menuntut otomasi yang dapat membuat banyak hal yang sebelumnya tidak dapat diperoleh. Namun demikian hal ini menuntut pemahaman pemrograman yang tidak sederhana untuk pemula. Hal ini terutama karena sintaks dan alur program yang mungkin baru pertama kali ditemui.

Konsep pemrograman berorientasi objek (*object-oriented*) digunakan memungkinkan penggunaan bagian dari program digunakan kembali untuk bagian lain (*reusable*) dari program tersebut atau bahkan dari program lain. Salah satu bahasa komputer yang dapat memenuhi pemrograman berorientasi objek tersebut adalah bahasa Java. Bahasa ini merupakan bahasa yang tidak tergantung pada platform (*operating system* dan *hardware*) serta mempunyai banyak fasilitas antara lain untuk pemrograman berorientasi objek, *graphic user interface* dan internet.

Diktat ini diharapkan berfungsi sebagai panduan untuk teori dan praktikum pada mata kuliah pemrograman 2 (semester II), pemrograman 3 (semester III) dan pemrograman berorientasi objek (semester V). Mahasiswa diharapkan membaca pula detail pembahasan yang terdapat dalam buku-buku teks berbahasa Inggris mengenai pemrograman JAVA lainnya. Beberapa contoh latihan yang tercantum dalam diktat diharapkan dipahami oleh mahasiswa.

Diktat ini masih dalam taraf awal dan masih terus dikembangkan lebih lanjut maka untuk pengertian dan kerjasamanya, penulis ucapkan banyak terimakasih dan sukses selalu. Tuhan Memberkati.

Jakarta, 30 Oktober 2014



## Daftar Isi

Kata Pengantar .....	1
Daftar Isi .....	3
Daftar Tabel .....	7
 BAB I Pengenalan Java .....	8
A. Sejarah Java .....	8
B. Karakteristik Bahasa Java .....	9
C. Bagaimana Java Bekerja? .....	11
D. Platform Java .....	12
E. Aplikasi Java .....	12
F. Identifier di Java .....	13
G. Keyword di Java .....	13
H. Tipe Data di Java .....	14
1. Tipe data sederhana .....	14
2. Tipe data komposit .....	16
J. Percabangan di Java .....	17
K. Perulangan di Java .....	18
 BAB II Pemrograman Berorientasi Obyek .....	20
A. Pendahuluan .....	20
B. Pemrograman Prosedural .....	21
C. Pemrograman Berorientasi Obyek .....	22
D. Kelas dan Obyek .....	24
E. Fitur Pemrograman Berorientasi Obyek .....	25
1. Enkapsulasi .....	25
2. Abstraksi .....	25
3. Pewarisan .....	26
4. Polimorfisme .....	28
 BAB III Kelas dan Obyek .....	30
A. Gambaran Umum tentang Orientasi Obyek .....	30
B. Kelas dan Obyek .....	31
C. Komponen Kelas .....	33



D. Membuat Kelas dan Obyek di Java .....	33
1. Pembuatan kelas di Java .....	34
2. Pembuatan Obyek di Java .....	34
E. Access Specifiers dan Access Modifiers .....	35
1. Access Specifiers .....	35
2. Access Modifiers .....	36
F. Atribut dan Method .....	37
1. Mengakses atribut.....	37
2. Mengakses Method.....	38
G. Method main() .....	38
H. Constructor (Konstruktor) .....	39
Praktek.....	40
 BAB IV Hubungan Antar Kelas dan Pewarisan .....	 43
A. Array.....	43
1. Mendeklarasikan Array .....	43
2. Membuat Array .....	44
3. Menginisialisasi Array.....	45
4. Array Multidimensi.....	46
5. Batasan Array .....	47
6. Manipulasi Array .....	47
B. Diagram Kelas.....	48
C. Hubungan Antar Kelas .....	49
1. Asosiasi.....	49
2. Agregasi .....	50
3. Komposisi .....	52
D. Pewarisan .....	53
1. Pewarisan di Java .....	54
2. Kelas Abstrak.....	55
3. Interface .....	56
 BAB V. Polimorfisme .....	 60
A. Polymorphism .....	60
1. Virtual Method Invocation (VMI) .....	61
2. Polymorphic arguments.....	62
3. Pernyataan instanceof.....	62



B. Overloading.....	63
1. Overloading konstruktor.....	63
2. Melakukan overloading pada method.....	66
C. Overriding.....	67
 BAB VI. Exception Handling.....	68
A. Pendahuluan .....	68
B. Kelas Exception .....	69
C. Mengimplementasikan Exception Handling .....	70
1. Statement Try dan Catch .....	70
2. Multiple Catch.....	71
3. Statement Finally .....	72
4. Throwing Exception .....	73
D. User defined Exception .....	75
 BAB VII. Input/Output.....	78
A. Pendahuluan .....	78
B. Implementasi Kelas File.....	79
C. Penggunaan Stream dalam Java .....	82
1. Implementasi Byte Stream .....	82
2. Implementasi Character Stream.....	92
D. Implementasi RandomAccessFile .....	99
 Daftar Pustaka.....	104



## Daftar Gambar

Gambar 1-1 Mekanisme Kompilasi dan Eksekusi Program Java .....	11
Gambar 1-2 Konsep <i>Write Once, Run Anywhere</i> pada Java.....	11
Gambar 1-3 <i>Platform Java</i> .....	12
Gambar 2-1 PP – Dekomposisi berdasar fungsi.....	22
Gambar 2-2 OOP – Dekomposisi berdasar obyek .....	22
Gambar 2-3 Ilustrasi keuntungan OOP : <i>resilience to change 1</i> .....	23
Gambar 2-4 Ilustrasi keuntungan OOP : <i>resilience to change 2</i> .....	23
Gambar 2-5 <i>Superclass</i> dan <i>Subclass</i> .....	26
Gambar 2-6 Contoh pewarisan .....	27
Gambar 2-7 <i>Single Inheritance</i> .....	27
Gambar 2-8 <i>Multiple Inheritance</i> .....	27
Gambar 3-1 Kelas .....	33
Gambar 3-2 Mengakses Atribut.....	37
Gambar 4-1 Representasi <i>Array</i> Primitif dalam <i>Heap Memory</i> .....	45
Gambar 4-2 Simbol Diagram Kelas.....	48
Gambar 4-3 Asosiasi.....	48
Gambar 4-4 Agregasi.....	48
Gambar 4-5 Komposisi .....	48
Gambar 4-6 Turunan .....	49
Gambar 4-7 Contoh hubungan asosiasi .....	49
Gambar 4-8 Contoh hubungan agregasi.....	50
Gambar 4-9 Contoh hubungan komposisi .....	52
Gambar 4-10 Contoh Pewarisan Kelas .....	54
Gambar 4-11 Contoh Interface.....	56
Gambar 6-1 Kesalahan pengkodean .....	68
Gambar 7-1 Kelas <i>OutputStream</i> .....	82
Gambar 7-2 Hirarki kelas <i>InputStream</i> .....	88
Gambar 7-3 Hirarki Kelas <i>Reader</i> .....	92
Gambar 7-4 Hirarki kelas <i>Writer</i> .....	95



## Daftar Tabel

Tabel 1-1 <i>Keyword</i> dalam Java .....	13
Tabel 1-2 Tipe data Integer .....	14
Tabel 1-3 Tipe data <i>Floating Point</i> .....	15
Tabel 1-4 Karakter <i>Unicode</i> .....	16
Tabel 1-5 <i>Operator unary</i> .....	16
Tabel 1-6 Operator aritmatika .....	17
Tabel 1-7 Operator relasi .....	17
Tabel 1-8 Operator boolean .....	17
Tabel 6-1 Contoh <i>Exception</i> .....	70
Tabel 7-1 Method dalam kelas File .....	79
Tabel 7-2 Method pada kelas <i>OutputStream</i> .....	83
Tabel 7-3 Method dalam kelas <i>FileOutputStream</i> .....	85
Tabel 7-4 Method dalam kelas <i>BufferedOutputStream</i> .....	86
Tabel 7-5 Method <i>InputStream</i> .....	88
Tabel 7-6 Method dalam kelas <i>FileInputStream</i> .....	90
Tabel 7-7 Method dalam kelas <i>Reader</i> .....	92
Tabel 7-8 Method dalam kelas <i>Writer</i> .....	95
Tabel 7-9 Jenis akses pembukaan file .....	100
Tabel 7-10 Method dalam kelas <i>RandomAccessFile</i> .....	101



## BAB I      *Pengenalan Java*

Ulasan :	Tujuan :
Bab ini akan menjelaskan tentang dasar - dasar pemrograman menggunakan bahasa Java. Penjelasan tentang sejarah bahasa Java, karakteristik bahasa Java, <i>identifier</i> , tipe data, dan operator pada Java, serta <i>expression</i> dan <i>flow control</i> pada Java.	Mengetahui sejarah perkembangan bahasa Java Mengetahui karakteristik bahasa Java Mengetahui dasar - dasar pemrograman menggunakan Java Mengetahui <i>identifier</i> dan tipe data pada Java Mengetahui <i>expression</i> dan <i>flow control</i> pada Java

---

### A. Sejarah Java

Pada tahun 1991, **Sun** dipimpin **Patric Naughton** dan **James Gosling** ingin merancang bahasa computer untuk perangkat *consumer* seperti *cable TV Box*. Karena perangkat itu tidak mempunyai banyak memori, bahasa harus berukuran kecil dan menghasilkan kode program yang liat. Juga karena manufaktur-manufaktur berbeda memilih pemroses-pemroses yang berbeda, maka bahasa harus bebas dari arsitektur manapun. Proyek ini diberi nama kode "**Green**".

Kebutuhan untuk kecil, liat dan kode netral terhadap *platform* mengatur tim mempelajari implementasi pascal yang pernah dicoba. **Niklaus Wirth**, pencipta bahasa Pascal telah merancang bahasa *portable* yang menghasilkan kode *intermediate* untuk mesin hipotetis. Mesin ini sering disebut *Virtual machine*. Kode antara ini kemudian dapat digunakan disembarang mesin yang memiliki *interpreter*. Proyek **Green** menggunakan *virtual machine* untuk mengatasi isu utama netral terhadap arsitektur mesin.

Karena orang-orang di proyek Green berbasis C++ bukan Pascal maka kebanyakan Sintaks diambil dari C++, serta mengadopsi orientasi obyek bukan *procedural*. Mulanya bahasa yang diciptakan diberi nama "**Oak**" kemudian diganti "**Java**" karena telah ada bahasa pemrograman bernama "**Oak**". Produk pertama proyek **Green** adalah "**\*7**", sebuah kendali jauh yang sangat cerdas. Karena pasar masih belum tertarik dengan produk *consumer* cerdas maka proyek **Green** harus menemukan pasar lain dari teknologi yang diciptakan. Kemudian, penerapan





mengarah menjadi teknologi yang berperan di web.

Pada 1995, **Netscape** memutuskan membuat *browser* yang dilengkapi dengan Java. Setelah itu diikuti oleh **IBM, Symantec, Inprise**, bahkan **Microsoft**. Setelah itu Java mulai didengar. Dengan strategi terbukanya, banyak industri yang melirikinya. Bersamaan itu disusul berbagai universitas Amerika, Jepang, dan Eropa yang mengubah pengenalan bahasa pemrograman komputer menjadi Java, meninggalkan C++. Java lebih sederhana dan telah mengakomodasikan hampir seluruh fitur penting bahasa-bahasa pemrograman yang ada semenjak perkembangan komputasi modern.

Java pertama kali diluncurkan sebagai bahasa pemrograman umum (*general purpose programming language*) dengan kelebihan dia bisa dijalankan di web *browser* sebagai *applet*. Sejak awal, para pembuat Java telah menanamkan visi mereka ke dalam Java untuk small embedded customer device) seperti TV, telepon, radio, dan sebagainya supaya dapat berkomunikasi satu sama lain. Langkah pertama yang diambil oleh Sun Microsystems adalah dengan membuat JVM (*Java Virtual machine*) yang kemudian diimplementasikan dalam bentuk JRE (*Java Runtime Environment*). JVM adalah lingkungan tempat eksekusi program Java berlangsung dimana para obyek saling berinteraksi satu dengan yang lainnya. *Virtual machine* inilah yang menyebabkan Java mempunyai kemampuan penanganan memori yang lebih baik, keamanan yang lebih tinggi serta portabilitas yang besar. Apabila kita hanya ingin menjalankan program Java, maka kita cukup memiliki JRE saja. Tapi seandainya kita ingin mengembangkan perangkat lunak sendiri, JRE saja tidak cukup.

Untuk lebih meningkatkan produktivitas pengembang perangkat lunak, Sun juga meluncurkan SDK (*Standard Development Kit*) yang berisi kaskas dan API untuk membuat program aplikasi berbasis Java. Pada tahun 1999 Sun meluncurkan J2EE (*Java 2 Enterprise Edition*) sebagai *framework* untuk membuat aplikasi *enterprise* berskala besar. Pada tahun 2001, Sun meluncurkan J2ME yang kelak menjadi salah satu standar pemrograman di dalam PDA maupun *handphone*.

## **B. Karakteristik Bahasa Java**

1. Sederhana, semudah C dan seampuh C++: berlawanan dengan anggapan orang-orang bahwa bahasa Java sulit untuk dipelajari, Java gampang untuk dipelajari terutama untuk orang yang sudah mengenal pemrograman tapi belum terlalu terikat pada paradigma pemrograman prosedural. Tentu saja ini



berarti bahwa kita harus siap mempelajari salah satu teknologi yang berkembang paling cepat di dunia dalam dua tahun terakhir ini dengan banyak membaca tentunya baik dari buku maupun melalui web.

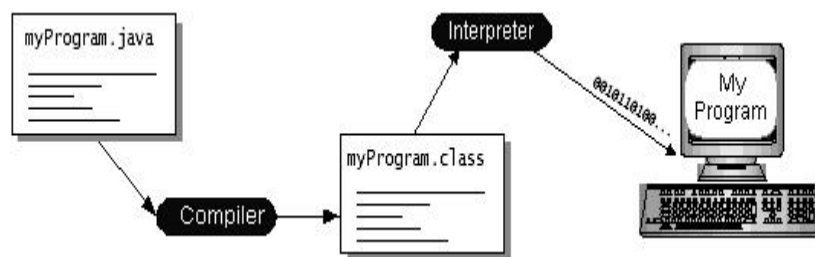
2. Sangat berorientasi obyek (OOP) dengan implementasi yang sangat baik sehingga kita bukan hanya belajar bagaimana membuat program yang baik (*reusable, scalable, dan maintainable*) tetapi juga kita belajar bagaimana cara berfikir yang baik untuk mengenali struktur masalah yang sedang kita hadapi dan memecahkannya secara sistematis dengan pola-pola tertentu (*patterns* *OpenPlatform, Write Once Run Anywhere* (WORA), *portable* atau *multiplatform*, program yang kita buat dapat dijalankan di Windows, Linux/Unix, Solaris, dan Macintosh tanpa perlu diubah maupun di kompilasi ulang. Java adalah juga bahasa yang paling sesuai digunakan bersama dengan XML yang membuat data menjadi *portable*, ini karena kelahiran XML tidak terlepas dari dukungan parser-parser berbahasa Java.
3. Arsitekturnya yang kokoh dan pemrograman yang aman. Dalam Java program yang kita buat tidak mudah untuk "*hang*" karena konflik pada memori biasanya diselesaikan dengan mengumpulkan obyek-obyek yang sudah tak terpakai lagi secara otomatis oleh *garbage collector*. Penanganan kesalahan juga dipermudah dalam Java dengan konsep *Exception*.
4. Bukan sekedar bahasa tapi juga *platform* sekaligus arsitektur. Java mempunyai portabilitas yang sangat tinggi. Ia dapat berada pada *smartcard, pager, POS (Point of Service), handphone, PDA, palm, TV, Embedded device (PLC, micro controller), laptop, pc, dan bahkan server*. Menyadari akan hal ini Sun membagi arsitektur Java menjadi tiga bagian, yaitu:
  - *Enterprise Java (J2EE)* untuk aplikasi berbasis web, aplikasi sistem tersebar dengan beraneka ragam klien dengan kompleksitas yang tinggi. Merupakan superset dari Standar Java
  - *Standard Java (J2SE)*, ini adalah yang biasa kita kenal sebagai bahasa Java, dan merupakan fokus kita sekarang.
  - *Micro Java (J2ME)* merupakan subset dari J2SE dan salah satu aplikasinya yang banyak dipakai adalah untuk *wireless device/mobile device*
5. Program Java dijalankan menggunakan *interpreter* melalui Java *Virtual machine (JVM)*. Hal ini menyebabkan *source code* Java yang telah dikompilasi menjadi Java *bytecodes* dapat dijalankan pada *platform* yang berbeda-beda.
6. Fitur-fitur utama yang lain:
  - Mendukung *multithreading*.
  - Selalu memeriksa tipe obyek pada saat *runtime*.



- Mempunyai *automatic garbage collection* untuk membersihkan obyek yang tidak terpakai dari memori
- Mendukung *exception* sebagai salah satu cara penanganan kesalahan

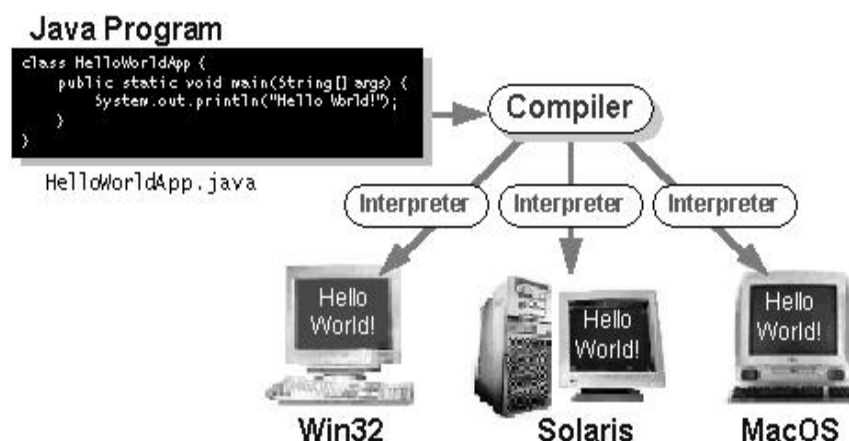
### C. Bagaimana Java Bekerja?

Lingkungan pemrograman pada Java menggunakan *compiler* sekaligus *interpreter* agar dapat berjalan pada *platform* yang berbeda. Java *compiler* melakukan kompilasi pada *source code* (**.java**) menjadi Java *bytecodes* (**.class**) seperti ditunjukkan oleh Gambar 1.1 berikut.



Gambar 1-1 Mekanisme Kompilasi dan Eksekusi Program Java  
(Sumber: <http://java.sun.com/docs/books/tutorial/>)

Java *bytecodes* merupakan instruksi mesin yang tidak spesifik terhadap *processor*. Oleh karena itu, program Java hasil kompilasi akan dapat dijalankan pada berbagai *platform* sistem komputer dengan menggunakan *Java Virtual machine* (JVM), "*write once, run anywhere*" (lihat Gambar 1.2). JVM disebut juga *bytecodes interpreter* atau *Java runtime interpreter*.



Gambar 1-2 Konsep *Write Once, Run Anywhere* pada Java  
(Sumber: <http://java.sun.com/docs/books/tutorial/>)



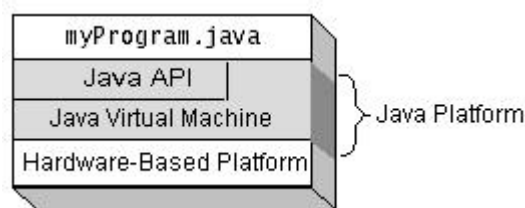
#### D. Platform Java

*Platform* dapat diartikan sebagai lingkungan perangkat keras atau perangkat lunak dimana program dijalankan. Umumnya *platform* dinyatakan berdasarkan nama sistem operasi yang digunakan, misalnya Windows 2000, Linux, Solaris, atau MacOS.

Tidak seperti bahasa pemrograman lainnya, *platform* Java mempunyai dua komponen, yaitu:

- *Java Virtual machine* (Java VM)  
Merupakan fondasi untuk *platform* Java yang dapat digunakan di berbagai *platform* perangkat keras
- *Java Application Programming Interface* (Java API)  
Kumpulan komponen-komponen perangkat lunak siap pakai (*ready-made software components*) untuk berbagai keperluan penulisan program, seperti *graphical user interface* (GUI).

Gambar 1.3. berikut memperlihatkan gambaran program yang berjalan diatas di kedua komponen *platform* Java.



Gambar 1-3 Platform Java

(Sumber: <http://java.sun.com/docs/books/tutorial/>)

#### E. Aplikasi Java

Ada dua tipe aplikasi pada Java yang umumnya sering ditulis, yaitu aplikasi *stand alone* dan *applets*. Aplikasi *stand alone* merupakan aplikasi yang dijalankan langsung diatas *platform* Java. Sedangkan *applets* adalah aplikasi yang dijalankan melalui *web browser* ataupun *applet viewer*.

Perbedaan *applets* dengan Java *stand alone* adalah:

- *Applets* melakukan *extends* dari *class applets*.
- *Applets* tidak mempunyai fungsi *main()*.
- *Applets* mempunyai beberapa batasan keamanan, seperti tidak diperbolehkan membaca atau menulis file pada sistem komputer.



- *Applets* tidak dapat menjalankan program lain pada komputer yang menjalankan *applets*.

Selain kedua tipe aplikasi tersebut, ada beberapa tipe aplikasi Java yang lain yaitu:

- a. Aplikasi berbasis Windows, misalnya dengan menggunakan fasilitas *Swing*.
- b. *Java Servlet*, yaitu aplikasi seperti *applets* tetapi dijalankan di sisi *server*.
- c. *Java Server Pages*, yaitu aplikasi di sisi *server* yang ditulis dalam bahasa *script* yang melekat pada halaman HTML untuk memberikan efek tampilan pada *web browser*.
- d. *Java Beans* dan *Enterprise Java Beans*, yaitu aplikasi untuk program *multitiers*.
- e. *Java Micro Edition*, yaitu aplikasi yang diperuntukan bagi perangkat genggam (*handheld devices*), seperti telepon genggam.

#### F. *Identifier* di Java

*Identifier* adalah nama yang diberikan kepada variabel, *method*, kelas, paket, dan interface untuk unik mengidentifikasikan bagi kompilator dan memberikan nama yang berarti bagi pemrogram. Adapun tatacara penamaan *identifier*:

1. *Case sensitive*, huruf kapital dan kecil dibedakan
2. *Identifier* yang diberikan oleh pemrogram tidak boleh sama dengan *keyword* yang ada di Java
3. Dimulai dengan huruf atau *underscore* (garis bawah) atau tanda (\$). Namun sebisa mungkin diawali dengan huruf karena mungkin *identifier* dengan awalan *underscore* dan (\$) digunakan untuk pemrosesan internal dan *file import*
4. Karakter berikutnya dapat berupa huruf atau angka 0 sampai 9. Simbol-simbol seperti '+' dan spasi tidak dapat digunakan

#### G. *Keyword* di Java

*Keyword* adalah *identifier* yang digunakan Java untuk suatu tujuan khusus. Daftar *keyword* Java adalah sebagai berikut:

Tabel 1-1 *Keyword* dalam Java

abstract	do	implements	private	this
boolean	double	import	protected	throw



break	else	instanceof	public	throws
byte	extends	int	return	transient
case	false	interface	short	true
catch	final	long	static	try
char	finally	native	strictfp	void
class	float	new	super	volatile
continue	for	null	switch	while
default	if	package	synchronized	

## H. Tipe Data di Java

Tipe data dalam Java dibagi dalam dua kategori:

1. Sederhana
2. Komposit

### 1. Tipe data sederhana

Tipe data sederhana merupakan tipe inti. Tipe sederhana tidak diturunkan dari tipe lain. Tipe ini adalah tipe data primitif. Terdapat delapan tipe data primitif di Java:

- Empat tipe adalah untuk bilangan bulat: **byte, short, int, long**
- Dua untuk tipe angka pecahan (*floating point*): **float, double**
- Satu untuk tipe karakter, yaitu **char**
- Satu untuk tipe **boolean** yang berisi nilai logika: *true/false*

#### a. Tipe data integer

Tipe data *integer* memiliki jangkauan nilai sebagai berikut:

Tabel 1-2 Tipe data Integer

Panjang Integer	Tipe data	Jangkauan Nilai
8 bit	<i>byte</i>	-27 to 27-1
16 bit	<i>short</i>	-215 to 215-1
32 bit	<i>int</i>	-231 to 231-1
64 bit	<i>long</i>	-263 to 263-1



kebanyakan situasi, tipe **int** paling banyak dipakai. Untuk data yang berukuran besar, digunakan tipe data **long**. Tipe **short** dan **byte** terutama digunakan untuk aplikasi khusus yang menangani file level rendah.

**b. Tipe data *Floating point***

Tipe data ini digunakan untuk perhitungan yang melibatkan bilangan pecahan, seperti perhitungan kosinus, akar persamaan, dan sebagainya. Java mengimplementasikan standar himpunan tipe dan operator titik mengambang IEEE-754. Keakuratan nilai untuk tipe data *floating point* adalah sebagai berikut:

Tabel 1-3 Tipe data *Floating Point*

Panjang Float	Tipe data	Nilai terbesar
32 bit	<i>Float</i>	3.40282e+38
64 bit	<i>Double</i>	1.79769e+308

Masing-masing tipe data *floating point* memiliki kebutuhan memori yang berbeda. Tipe data **float** memerlukan 32 bit sebagai *single-precision*, sedangkan tipe data **double** memerlukan 64 bit sebagai *double precision*. Nama **double** mengacu pada presisinya yang sebesar dua kali dibandingkan **float**. Presisi **float** kebanyakan tidak memadai untuk banyak aplikasi komputasi. Angka literal bertipe **float** berakhiran F, contoh 3.14F sedangkan kalau tidak diberi akhiran F akan dipandang sebagai bertipe **double**.

**c. Tipe data *Char***

Tipe data **char** merupakan tipe data yang direpresentasikan dengan 16-bit *Unicode character*. Literal dari *char* harus berada diantara *single quotes* ( ' '). Contohnya :

'a' huruf a

'\t' karakter tab

*Unicode* dirancang untuk menangani semua karakter di dunia dalam kode 2 byte. Kode 2 byte memungkinkan 65.536 karakter, dimulai dari nilai byte 0 sampai 65.535. Himpunan karakter ASCII dipandang sebagai bagian dari *Unicode* dan ditempatkan sebagai 256 karakter pertama dari *Unicode*. Terdapat pula beberapa barisan escape untuk karakter *Unicode* yang spesial, seperti berikut:



Tabel 1-4 Karakter *Unicode*

Barisan <i>Escape</i>	Nama	Nilai <i>Unicode</i>
<code>\b</code>	<i>Backspace</i>	<code>\u008</code>
<code>\t</code>	<i>Tab</i>	<code>\u009</code>
<code>\n</code>	<i>Linefeed</i>	<code>\u00a</code>
<code>\r</code>	<i>Carriage return</i>	<code>\u00d</code>
<code>\"</code>	Petik ganda	<code>\u0022</code>
<code>\'</code>	Petik tunggal	<code>\u0027</code>
<code>\\</code>	<i>Backslash</i>	<code>\u005c</code>

#### d. Tipe data Boolean

Tipe data *boolean* memiliki 2 literal yaitu : *true* dan *false*. Contoh, *Statemen* :

```
boolean truth = true;
```

mendeklarasikan variabel *truth* sebagai tipe data boolean dan memberikan nilai *true*

### 2. Tipe data komposit

Tipe data komposit merupakan tipe data yang disusun dari tipe data sederhana atau tipe komposit lain yang sudah ada. Tipe ini dapat berupa *array*, *string*, kelas, dan *interface*.

Khusus untuk *String* pada Java dikenali sebagai kelas, bukan sebagai *array of character*. *String* pada Java diapit oleh tanda petik ganda ("....."), contoh :

```
String s = "Saya makan nasi";
```

## I. Operator di Java

Java memiliki beberapa jenis operator di antaranya:

### - **Operator unary:**

Tabel 1-5 *Operator unary*

Nama Operator	Simbol	Definisi
Increment	<code>++</code>	Akan menambahkan nilai sejumlah satu
Decrement	<code>--</code>	Akan mengurangi nilai sejumlah satu

Contoh penggunaan:

```
int x = 5;  
int y = x++;
```





Pada kasus di atas nilai y akan berisi 5 dan nilai x akan berisi 6 karena nilai y akan mengambil nilai x dahulu setelah itu baru nilai x ditambahkan satu, berbeda kasusnya pada contoh di bawah ini:

```
int x = 5;  
int y = ++x;
```

pada kasus di atas, nilai y akan berisi 6 dan x berisi 6 karena nilai x akan ditambahkan satu dahulu baru kemudian dimasukkan ke variabel y.

- **Operator aritmatika:**

Tabel 1-6 Operator aritmatika

Nama Operator	Simbol	Deskripsi
Penambahan	+	Menambahkan dua buah nilai
Pengurangan	-	Pengurangan dua buah nilai
Perkalian	*	Perkalian dua buah nilai
Pembagian	/	Pembagian dua buah nilai
Sisa bagi	%	Sisa pembagian dua buah nilai

- **Operator relasi:**

Tabel 1-7 Operator relasi

Simbol	Deskripsi
<	Kurang dari
>	Lebih dari
<=	Kurang dari atau sama dengan
>=	Lebih dari atau sama dengan
==	Sama dengan
!=	Tidak sama dengan

- **Operator boolean :**

Tabel 1-8 Operator boolean

Simbol	Deskripsi
&&	AND
	OR
^	XOR
!	NOT

## J. Percabangan di Java

Percabangan di Java menggunakan dua jenis Sintaks:

### 1. Sintaks if

```
Sintaks if-else, sebagai berikut :  
if (boolean expression) {  
    Statement or block  
}  
else if (boolean expression) {
```



```
Statement or block
}
else {
Statement or block
}
```

## 2. Sintaks *switch*

Sintaks *switch* sebagai berikut :

```
switch (expression) {
    case constant1    : Statements;
                      break;
    case constant2    : Statements;
                      break;
    default           : Statements;
                      break;
}
```

## K. Perulangan di Java

Perulangan di Java menggunakan 3 jenis Sintaks:

### 1. Perulangan *for*

Sintaks *for* sebagai berikut :

```
for (init_expr; boolean testexpr; alter_expr)
{
    Statement or block
}
```

Contoh :

```
for (int i=0;i<10;i++)
{
    System.out.println(i);
}
```

### 2. Perulangan *while*

Sintaks *looping while* sebagai berikut :

```
while (boolean testexpr)
{
    Statement or block
}
```

Contoh :

```
int i = 0;
while (i<10) {
    Sistem.out.println(i);
    i++;
}
```

### 3. Perulangan *do....while*

Sintaks *do/while loop*, sebagai berikut

```
do {
    Statement or block
```



```
}  
while (boolean testexpr)
```

Contoh :

```
int i = 0;  
do {  
    Sistem.out.println(i);  
    i++;  
}  
while (i<10);
```

## Essay

1. Sebutkan dan jelaskan 3 fitur yang ada pada bahasa pemrograman Java!
2. Sebutkan dan jelaskan 3 arsitektur bahasa pemrograman Java!
3. Buatlah program dengan bahasa pemrograman Java untuk menampilkan bilangan genap antara 1 s.d. 100
4. Buatlah program dengan bahasa pemrograman Java untuk menampilkan bilangan ganjil antara 1 s.d. 100!
5. Buatlah program dengan bahasa pemrograman Java untuk menampilkan hasil dari  $f(x) = x^2 + 2x + 1$  dengan nilai  $x$  1 s.d. 10!



## BAB II *Pemrograman Berorientasi Obyek*

Ulasan :	Tujuan :
<b>Dalam Bab ini akan dibahas tentang pengenalan terhadap pemrograman berorientasi obyek. Perbandingan pemrograman berorientasi obyek dengan pemrograman prosedural. Juga akan dibahas tentang fitur-fitur yang ada dalam pemrograman berorientasi obyek</b>	Mengetahui konsep pemrograman berorientasi obyek. Mengetahui perbedaan antara pemrograman berorientasi obyek dan pemrograman prosedural. Mengetahui keuntungan dari pemrograman berorientasi obyek. Mengetahui kelas dan obyek. Mengetahui fitur dari pemrograman berorientasi obyek

---

### A. Pendahuluan

Dalam dunia teknologi informasi saat ini, pengembangan sebuah solusi untuk memenuhi kebutuhan yang ada di dunia nyata harus dikembangkan dengan mudah, cepat dan memenuhi berbagai macam keperluan. Hal ini akan membawa kita kedalam proses pengembangan sebuah perangkat lunak yang sangat kompleks. Meskipun hal itu tidak bisa dihindari, namun harus ada jalan agar kompleksitas tersebut tidak menjadi sebuah batasan terhadap fungsionalitas dari perangkat lunak tersebut.

Salah satu cara untuk mengurangi kompleksitas dari perangkat lunak adalah dengan membaginya berdasarkan hirarki tertentu berdasarkan fungsi-fungsi yang ada. Salah satu pendekatan pengembangan perangkat lunak sebelum adanya pendekatan berorientasi obyek (*Object oriented Programming* atau OOP) adalah pemrograman prosedural, yang membagi sebuah aplikasi berdasarkan berbagai macam fungsi yang ada didalamnya. Fungsi-fungsi tersebut bisa dieksekusi dari baris manapun dalam bagian program, dimana setiap fungsi terdiri dari serangkaian langkah-langkah pengeksekusian perintah untuk menyelesaikan sebuah aktivitas.

Fungsi-fungsi didalam program prosedural saling bergantung, sehingga sulit untuk dipisahkan satu dengan yang lainnya. Fungsi yang saling bergantung ini sulit untuk dapat digunakan (*reused*) didalam aplikasi lain, dan untuk menggunakannya maka harus dilakukan perubahan terhadap kode yang ada.



Beberapa contoh bahasa pemrograman prosedural adalah COBOL (*COmmon Business Oriented Language*), Pascal dan BASIC (*Beginner's All Purpose Symbolic Instruction Code*).

Kekurangan dalam pemrograman prosedural membawa kepada munculnya pemrograman berorientasi obyek. Pemrograman berorientasi obyek diciptakan untuk mempermudah pengembangan program yang mengikuti model yang telah ada dalam kehidupan nyata. Dalam paradigma ini, sesuai dengan model kehidupan nyata, segala sesuatu entitas yang ada merupakan sebuah obyek. Obyek-obyek inilah yang saling berinteraksi dan membentuk sebuah sistem.

Mari kita ambil sebuah contoh yaitu sebuah mobil. Mobil adalah sebuah benda yang nyata ada dalam kehidupan. Kalau kita perhatikan bahwa sebuah mobil sebenarnya terdiri dari banyak komponen-komponen yang juga merupakan sebuah obyek, sebut saja mesin, pedal gas dan rem, spion, kursi penumpang, roda, dan lain-lain. Masing-masing komponen tersebut saling berinteraksi sehingga membentuk sebuah mobil yang akhirnya bisa dikendarai dan berjalan. Tak jauh beda dengan sebuah program atau aplikasi, terdiri dari banyak obyek-obyek yang saling berinteraksi. Misal program simulator mobil dari contoh diatas tadi. Semua komponen mobil akan dimodelkan menjadi obyek sendiri-sendiri yang akan berinteraksi. Spion berinteraksi dengan pengguna dalam bentuk pemilihan sudut pandangan dari kursi pengemudi. Kecepatan simulasi perjalanan bisa diperbesar dan diperkecil tenaganya dengan menekan pedal gas. Semua obyek tersebut saling berinteraksi, sehingga apa yang ada dalam dunia nyata bisa dimodelkan dengan tepat dalam sebuah program.

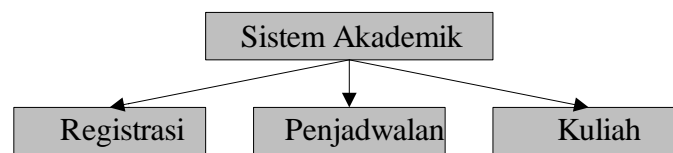
Dalam pemrograman berorientasi obyek, obyek bisa ditempatkan kedalam sebuah *library*, sehingga bisa digunakan dan diakses oleh berbagai macam aplikasi lainnya. Keuntungannya adalah kemampuannya dalam "*reusability of code*", sehingga akan menghemat waktu dan usaha dalam pengembangan program.

## **B. Pemrograman Prosedural**

Dalam Pemrograman prosedural, sebuah program akan dibagi menjadi beberapa *subprogram* (modul) yang akan melakukan tugas-tugas tertentu. Modul tersebut bisa berupa pernyataan yang ditentukan oleh user ataupun diambil dari *library* dari program lainnya. Dalam sebuah modul bisa terdiri dari satu atau lebih prosedur atau dalam bahasa pemrograman juga disebut sebagai fungsi, rutin, subrutin, atau *method*.



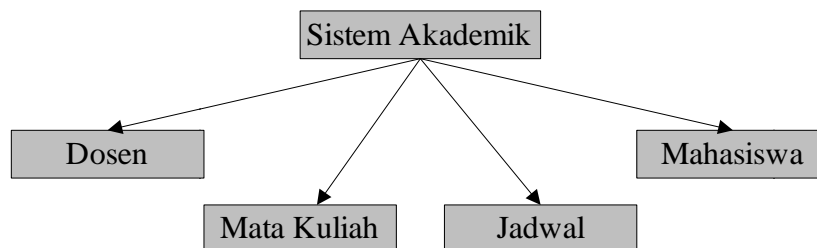
Sebuah prosedur adalah sekelompok perintah yang bisa dijalankan secara independen. Tetapi yang menjadi masalah dalam pemrograman *procedural* adalah bahwa setiap langkah dalam sebuah prosedur yang dieksekusi, selalu terkait dengan langkah yang sebelumnya. Dalam hal ini data juga bisa dengan mudah menjadi "rusak", karena bisa diakses secara bebas dari keseluruhan fungsi, termasuk dari bagian yang harusnya tidak memiliki hak akses kedalamnya. Berikut adalah contoh dekomposisi sebuah sistem akademik menggunakan paradigm pemrograman prosedural.



Gambar 2-1 PP – Dekomposisi berdasar fungsi

### C. Pemrograman Berorientasi Obyek

Berbeda dengan pemrograman prosedural, seperti yang telah dijelaskan sebelumnya bahwa pemrograman berorientasi obyek memecah komponen-komponennya menjadi obyek-obyek yang saling berinteraksi. Dunia nyata yang terdiri dari obyek-obyek dapat dengan mudah dimodelkan sehingga program bisa lebih mendekati kondisi yang sebenarnya.



Gambar 2-2 OOP – Dekomposisi berdasar obyek

Terdapat beberapa keuntungan menggunakan pemrograman berorientasi obyek, antara lain:

- ***Real world programming***

Dunia ini disusun atas obyek obyek yang saling berinteraksi. Sebuah program yang memodelkan dunia nyata, sebisa mungkin menggambarkan kondisi yang ada dalam bentuk yang seakurat mungkin. Dalam Pemrograman berorientasi obyek, sebuah program disusun oleh obyek-obyek yang masing-masing memiliki fungsi sesuai dengan peran dan kebutuhan interaksinya.

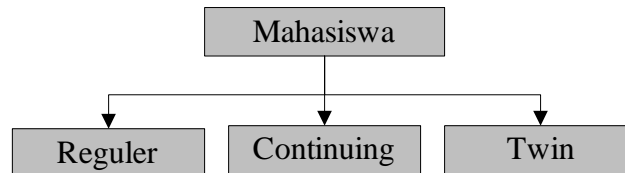


- **Reusability of code**

Kelas yang telah dibuat dalam pemrograman berorientasi obyek bisa digunakan oleh program lain. Penggunaan komponen yang telah dibuat tidak hanya mengurangi usaha pembuatan komponen tersebut, tetapi juga mengurangi kemungkinan kesalahan jika harus mengembangkan lagi dari awal. Keuntungannya adalah penghematan dari segi waktu, dan usaha yang akhirnya akan membawa kepada penghematan biaya pengembangan.

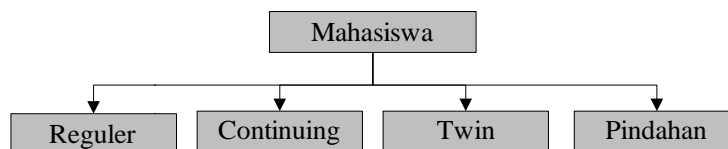
- **Resilience to change**

Dunia nyata adalah sesuatu yang dinamis, perubahan akan selalu terjadi didalamnya. Program atau aplikasi yang memodelkan dunia nyata, diharapkan juga bisa bersifat dinamis pula. Sebagai contoh perhatikan ilustrasi berikut :



Gambar 2-3 Ilustrasi keuntungan OOP : *resilience to change* 1

Misal kondisi saat ini jenis mahasiswa yang ada adalah seperti yang terlihat diatas. Sebuah aplikasi harus bisa menangani jika terdapat penambahan jenis mahasiswa baru yang kedepannya bakal ada. Tanpa harus mengubah keseluruhan aplikasi, maka kebutuhan tersebut harus bisa ditangani.



Gambar 2-4 Ilustrasi keuntungan OOP : *resilience to change* 2

- **Information hiding**

Informasi dalam sebuah obyek sedapat mungkin disembunyikan dari luar kelas yang ada. Tujuannya adalah mengamankan data agar hanya fungsi yang ada dalam kelas itu saja yang bisa membaca, mengubah dan memanipulasi data tersebut. Tetapi tetap disediakan sebuah cara (*interface*) agar obyek dari luar bisa mengakses dan mengubah data secara tidak langsung.

Konsep menyembunyian informasi ini bertujuan agar pihak luar yang membutuhkan layanan dari kelas tersebut hanya perlu untuk menerima data yang dibutuhkan saja, tanpa perlu mengetahui bagaimana sebenarnya cara kerja di dalam kelas tersebut.



Sebagai contoh pada sebuah jam. Kita sebagai pengguna tidak perlu memahami dengan pasti bagaimana cara kerja didalam jam tersebut. Para pengguna tidak perlu mengetahui sebanyak apa roda giginya, menggunakan tenaga baterai atau kinetik, dan sebagainya. Yang perlu diketahui oleh pengguna adalah bahwa jam tangan tersebut menyediakan sebuah penunjuk (yang bisa kita anggap sebagai fungsi) yang menunjukkan bahwa saat ini waktu menunjukkan pukul berapa.

- **Modularity of code**

Salah satu keuntungan dari pemrograman berorientasi obyek adalah modularitas, yang berarti bahwa setiap obyek yang dibentuk dikelola secara terpisah dari obyek lainnya meskipun berasal dari sebuah kelas yang sama. Modifikasi terhadap sebuah obyek bisa dilakukan tanpa mempengaruhi fungsionalitas dari obyek yang lainnya.

Sebagai sebuah contoh adalah dari obyek mahasiswa dan dosen. Masing-masing obyek akan saling berinteraksi didalam sebuah perkuliahan yang terjadwal. Akan tetapi jika terjadi perubahan terhadap jumlah mahasiswa (bertambah atau berkurang) yang mengikuti proses perkuliahan, hal itu tidak akan mempengaruhi kondisi dosen dalam melakukan pengajaran.

#### **D. Kelas dan Obyek**

Sebuah kelas menentukan struktur dan *behaviour* dari sebuah obyek. Sebagai sebuah contoh, truk, bus, mobil, sepeda motor didefinisikan sebagai sebuah kendaraan bermotor karena memiliki beberapa karakteristik yang sama yaitu memiliki mesin, menggunakan bahan bakar minyak untuk pengoperasiannya. Kendaraan bermotor tersebut juga memiliki beberapa atribut yang sama dalam kelasnya, misalkan memiliki jumlah roda, no kendaraan, nomor rangka mesin, jumlah kursi penumpang, dan lain-lain.

Obyek merupakan dasar dari pemrograman berorientasi obyek, didalam dunia nyata, setiap obyek yang ada memiliki dua buah karakteristik: *State* dan *behaviour*.

- *State* merupakan atribut yang dimiliki oleh sebuah obyek.
- *Behaviour* adalah fungsi yang dimiliki dan bisa dijalankan oleh penggunanya.

Sebagai contoh obyek mahasiswa. Setiap mahasiswa memiliki atribut seperti NIM, Nama, alamat, dll. dan sebagai *behaviour*-nya yaitu mahasiswa melakukan fungsi registrasi, perkuliahan, ujian, praktikum, dll.





Kelas merupakan sebuah cetak biru dari obyek. Seorang programmer akan menulis kode-kode untuk masing kelas yang akan digunakan. Sedangkan obyek merupakan bentukan (*instance*) dari sebuah kelas yang dijalankan pada saat *runtime*. Pada saat runtime, variabel yang ada pada sebuah kelas akan menjadi *State* pada sebuah obyek, sedangkan *method* yang dibuat akan menjadi *behaviour*-nya.

## **E. Fitur Pemrograman Berorientasi Obyek**

Didalam pemrograman berorientasi obyek terdapat beberapa fitur yaitu Enkapsulasi (*Encapsulation*), Abstraksi (*Abstraction*), Pewarisan (*Inheritance*), dan Polimorfisme (*Polymorphism*).

### **1. Enkapsulasi**

Enkapsulasi adalah suatu cara untuk menyembunyikan implementasi detail dari sebuah kelas. Terdapat dua hal mendasar dari enkapsulasi yaitu:

- *Information hiding*: penyembunyian detil dari atribut dan *method* pada sebuah kelas.
- *Interface* untuk pengaksesan data: suatu *method* untuk mengambil, memberikan atau mengubah suatu nilai.

Dalam pemrograman berorientasi obyek, kebutuhan akan enkapsulasi muncul karena adanya proses *sharing* data antar *method*. Dengan menggunakan enkapsulasi, maka keamanan dari data dan *method* yang ada didalamnya akan terjaga.

Sebagai contoh adalah pada sebuah mesin ATM. Yang disediakan oleh mesin ATM adalah sebuah layar informasi dan perangkat untuk membaca kartu. Saat kartu ATM dimasukkan, maka pengguna akan memasukkan nomor pin. Ketika pin dimasukkan, berarti telah terjadi proses pengaksesan data yang akan memberikan nomor pin untuk dikirimkan dan divalidasi pada server dari ATM tersebut. Disini terjadi penyembunyian informasi tentang bagaimana cara kerja pengecekan validitas kartu, kecocokan pin yang dimasukkan, koneksi ke database server, dll, dimana hal-hal tersebut tidak perlu diketahui oleh pengguna tentang bagaimana cara kerjanya.

### **2. Abstraksi**

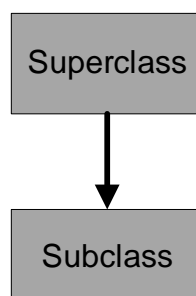
Arti abstraksi mengacu kepada atribut dari sebuah obyek yang membedakan antara satu obyek dengan obyek yang lain. Misalnya pada sebuah sistem akademik, dimana terdapat beberapa komponen misal saja mahasiswa. Didalam obyek tersebut, terdapat suatu atribut yang akan saling



membedakan antara satu obyek mahasiswa dengan mahasiswa lainnya, yaitu NIM, Nama, tanggal lahir, alamat, dan sebagainya. Dalam pemrograman berorientasi obyek konsep ini berada pada pembuatan sebuah kelas. Semua atribut dari obyek didefinisikan dalam sebuah kelas. Sebenarnya kelas tidak memiliki data, tetapi sebuah obyek-lah yang akan menyimpan data, karena obyek diciptakan dari sebuah kelas dan oleh sistem operasi akan dialokasikan sejumlah memori kepada obyek tersebut.

### 3. Pewarisan

Salah satu fitur yang paling kuat dalam pemrograman berorientasi obyek adalah penggunaan kode kembali (*code reuse*). Sekali sebuah prosedur dibuat, maka kita bisa menggunakannya berulang kali. Dalam pemrograman berorientasi obyek, kemampuan yang dimiliki tidak hanya itu, tetapi kita juga bisa mendefinisikan hubungan antar kelas yang tidak hanya dimanfaatkan untuk *code reuse*, tetapi juga dari segi pendesainannya yang secara garis besar jauh lebih baik, dengan cara mengelola kelas-kelas dan faktor kemiripan diantara kelas-kelas tersebut. Tujuan utama dari pewarisan (*inheritance*) adalah untuk menyediakan fungsionalitas tersebut.

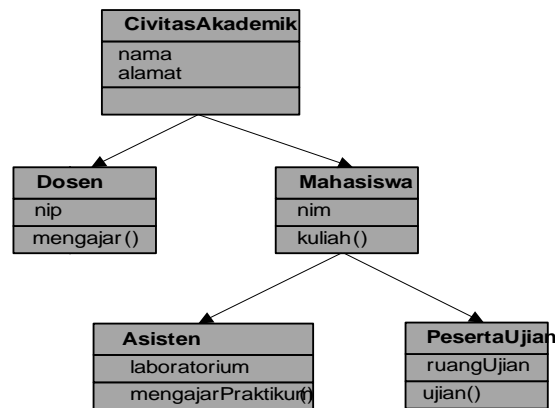


Gambar 2-5 *Superclass* dan *Subclass*

Fitur pewarisan mengijinkan sebuah kelas yang dinamakan *superclass* untuk menurunkan atribut-atribut dan *method*nya kepada yang lainnya, yaitu yang disebut *subclass* atau kelas turunannya. Hal ini akan mengijinkan pembuatan kelas baru yang didasarkan dari peng-abstrakan atribut-atribut dan *behaviour* yang sama.

Terdapat beberapa karakteristik dari *subclass*, yaitu:

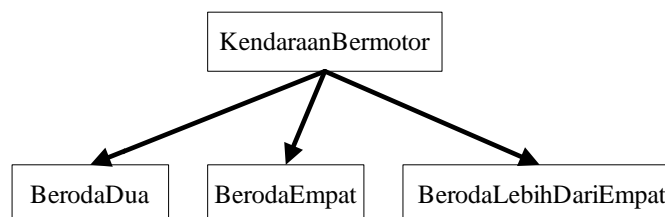
- *Subclass* merupakan bentuk khusus dari sebuah *superclass*
- Dalam sebuah *subclass* dapat memiliki atribut dan *method* yang diturunkan dari sebuah *superclass*
- *Subclass* bisa memiliki fitur tambahan dan berbeda yang berbeda dari fitur-fitur yang diturunkan dari *superclass*nya.



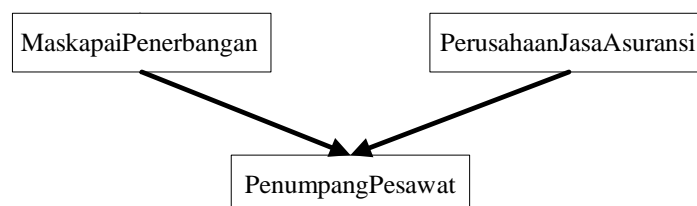
Gambar 2-6 Contoh pewarisan

Dari contoh diatas bisa dilihat bahwa terdapat sebuah kelas *CivitasAkademik* yang memiliki 2 (dua) kelas turunan, yaitu *Dosen* dan *Mahasiswa*. Meskipun didalam kelas *Dosen* dan *Mahasiswa* tidak tercantum atribut Nama dan Alamat, tetapi keduanya tetap memiliki kedua atribut tersebut yang diturunkan dari kelas *CivitasAkademik*. Begitu juga untuk kelas *Asisten* dan *PesertaUjian*. Keduanya tetap memiliki atribut Nama dan Alamat yang diturunkan dari kelas *Mahasiswa*, yang sebenarnya juga turunan dari kelas *CivitasAkademik*. Hal tersebut tidak hanya berlaku untuk atribut saja, tetapi juga *method* - *method* yang ada di kelas induknya. Didalam pemrograman berorientasi obyek, terdapat beberapa jenis pewarisan, yaitu:

- *Single inheritance*, yaitu hanya terdapat satu *superclass*.
- *Multiple inheritance*, yaitu terdapat lebih dari satu *superclass*. Didalam java bahasa pemrograman java sendiri konsep tentang *Multiple inheritance* dihapuskan, karena kompleksitas dari konsep tersebut.



Gambar 2-7 *Single Inheritance*



Gambar 2-8 *Multiple Inheritance*



#### 4. Polimorfisme

Polimorfisme diturunkan dari bahasa latin yaitu *poly* yang berarti banyak dan *morph* yang berarti bentuk. Polimorfisme sendiri berarti sesuatu yang memiliki banyak bentuk. Sebagai contoh adalah sebuah obyek wanita, beberapa peran yang mungkin dimiliki adalah:

- Bagi suami maka dia berperan sebagai seorang istri.
- Buat anak-anak berperan sebagai ibu.
- Di tempat kerja maka dia akan berperan sebagai seorang karyawan.
- Di tempat kuliah berperan sebagai mahasiswa.
- Di tempat arisan berperan sebagai ketua arisan.

Dari contoh diatas bisa kita lihat bahwa wanita tersebut adalah orang yang sama, tetapi memiliki peran yang berbeda bagi orang yang berinteraksi dengannya.

Didalam pemrograman berorientasi obyek, polimorfisme adalah sebuah fitur yang memungkinkan kita untuk memberikan arti atau penggunaan yang berbeda bagi sebuah entitas dalam konteks yang berbeda-beda. Entitas tersebut bisa berupa variabel, *method*, atau sebuah obyek. Polimorfisme bisa digunakan sebagai kategori umum bagi sebuah entitas dalam tindakan yang berbeda-beda.

#### **Dalam materi ini kita telah mempelajari tentang:**

- Beberapa metodologi pengembangan perangkat lunak:
  - Pemrograman *procedural*
  - Pemrograman berorientasi obyek
- Pemrograman prosedural membagi sebuah program menjadi prosedur dan fungsi untuk melakukan tugas khusus.
- Pemrograman berorientasi obyek membagi sebuah program berdasarkan kelas dan obyek yang saling berinteraksi.
- Kelas merupakan kumpulan dari beberapa obyek yang sejenis.
- Obyek merupakan instantiasi kelas pada saat *runtime*.
- Beberapa fitur pemrograman berorientasi obyek.
  - *Real world programming*
  - *Reusability of code*
  - *Resilience to change*
  - *Information hiding*
  - *Modularity of code*



- Enkapsulasi adalah menyembunyikan detail implementasi dari sebuah obyek.
- Pewarisan adalah pembuatan hirarki dari sebuah kelas dan membantu pendefinisian atribut dan *method* bagi kelas turunannya.
- Polimorfisme adalah pemberian arti dan fungsi yang berbeda dalam penggunaan sebuah entitas yang memiliki identitas yang sama sesuai dengan konteksnya.

## **Essay**

1. Sebutkan dan jelaskan fitur-fitur dari pemrograman berorientasi obyek.
2. Berikan contoh bagi masing-masing fitur dari pemrograman berorientasi obyek.
3. Sebutkan perbedaan antara pemrograman berorientasi obyek dan pemrograman prosedural.



## BAB III *Kelas dan Obyek*

### Ulasan :

**Bab ini akan menjelaskan tentang dasar Pemrograman Berorientasi Obyek yaitu Kelas dan Obyek. Penjelasan tentang karakteristik kelas dan obyek, komponen kelas, serta cara pembuatan dan pendeklarasian kelas di dalam Java. Akan dijelaskan juga tentang cara pengaksesan komponen kelas.**

### Tujuan

1. Mengetahui pengertian kelas dan obyek
2. Mengetahui karakteristik kelas dan obyek
3. Mengetahui fitur Pemrograman Berorientasi Obyek
4. Mengetahui cara menentukan komponen dari kelas
5. Mengetahui cara mendeklarasikan kelas dan obyek di Java

---

### A. Gambaran Umum tentang Orientasi Obyek

Teknik pemecahan masalah pada *object oriented* lebih dekat merupakan model tersekat dengan pemecahan masalah sehari-hari oleh manusia. Kita bayangkan bagaimana kita memecahkan permasalahan sehari-hari: kita ingin menyewa buku dari Perpustakaan Unindra. Untuk memecahkan masalah ini, peminjam dapat dengan mudah berjalan ke perpustakaan dan bertanya kepada petugas, misal nama petugas tersebut adalah Ibu Ulfa. Peminjam memberitahukan kepada Ibu Ulfa judul buku yang akan dipinjam.

Sekarang kita kaji mekanisme yang dapat digunakan untuk memecahkan permasalahan :

- o Pertama peminjam menemukan petugas dan memberikan pesan kepada petugas yang berisi sebuah permintaan
- o Merupakan kewajiban dari Ibu Ulfa untuk melayani permintaan tersebut
- o Ada beberapa metode (sebuah algoritma atau sekumpulan operasi) yang digunakan oleh Ibu Ulfa
- o Peminjam tidak perlu mengetahui metode yang digunakan untuk menyelesaikan permintaan seperti informasi disembunyikan

Pemecahan masalah ini membawa kita ke gambaran konsep pertama pada pemrograman berorientasi obyek :



*"Sebuah program berorientasi obyek terstruktur atas bersatunya agen-agen yang berinteraksi yang disebut obyek. Tiap obyek mempunyai peran. Tiap obyek menyediakan layanan atau menampilkan aksi yang digunakan oleh anggota lain dalam komunitas."*

Dalam permasalahan antara peminjam dengan petugas perpustakaan, antara peminjam dan petugas terjadi beberapa interaksi dimana petugas perpustakaan menyediakan sebuah layanan.

Anggota dari *object oriented* saling membuat permintaan. Prinsip penting berikutnya adalah :

*"Aksi diinisiasi di dalam pemrograman berorientasi obyek dengan mengirimkan sebuah pesan ke suatu agen (obyek) untuk merespon aksi. Pesan menkodekan permintaan untuk sebuah aksi dan biasanya ditemani oleh beberapa informasi (argument) yang diperlukan dalam permintaan tersebut. Pihak penerima adalah obyek tujuan dimana pesan akan dikirim. Jika penerima telah menerima pesan, pihak ini akan bertanggungjawab untuk menjalankan aksi yang berkaitan. Untuk merespon pesan, pihak penerima akan menjalankan metode untuk memenuhi permintaan tersebut."*

Beberapa hal penting disini adalah :

- Klien mengirim permintaan dan tidak mengetahui cara memenuhi request. Hal ini disebut dengan prinsip penyembunyian informasi (*information hiding*)
- Prinsip dalam penyampaian pesan adalah mencari obyek lain untuk melakukan pekerjaan
- Interpretasi dari sebuah pesan dijelaskan oleh penerima dan hal ini akan berbeda antar penerima yang berbeda
- Tanggung jawab obyek pada pemrograman berorientasi obyek disebut *behaviour*
- Obyek mempunyai tanggung jawab dalam memenuhi suatu permintaan

## **B. Kelas dan Obyek**

Sebuah **Kelas** mendefinisikan struktur (*structure*) dan tingkah laku (*behaviour*) sebuah obyek atau sekumpulan obyek.

Sebuah **Obyek** merupakan *instance* dari kelas. Metode yang dijalankan oleh sebuah obyek ketika merespon sebuah pesan (*message*) dibuat oleh kelas yang menerima pesan.



Semua obyek dari sebuah kelas menggunakan metode yang sama untuk merespon pesan yang sama. Ibu Vika juga *instance* dari kategori Petugas Perpustakaan. Petugas perpustakaan merepresentasikan sebuah kelas atau kategori untuk semua petugas perpustakaan.

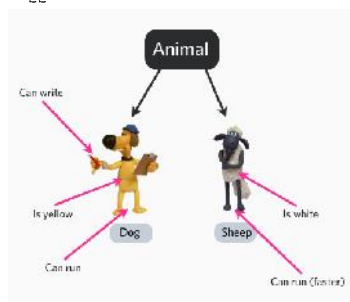
Obyek merupakan dasar bangunan pada struktur Pemrograman Berorientasi Obyek. (OOP=*Object oriented Programming*). Konsep berorientasi obyek berdasarkan obyek. Kita berinteraksi dengan *instance* dari kelas tetapi kelas yang menjelaskan tingkah laku (*behaviour*) dari sistem, Contoh : kita dapat mengetahui lebih banyak tentang tingkah laku Ibu Vika sebagai petugas perpustakaan dengan mengetahui bagaimana seorang petugas perpustakaan bertindak.

Di dalam kehidupan yang nyata, obyek mempunyai dua karakteristik : *State* dan *behaviour*

- o *State* : *State* dari obyek diindikasikan dengan sekumpulan atribut dan nilai dari atribut tersebut. Contoh : Siswa mempunyai *State* nama, nim, mata kuliah yang diambil, jenis kelamin. Buku mempunyai *State* judul, jumlah halaman, pengarang.
- o *Behaviour* : *behaviour* (tingkah laku) mengindikasikan bagaimana sebuah obyek beraksi dan bereaksi ketika terjadi perubahan *State* dan pengiriman pesan (*message*). Contoh : Siswa mempunyai *behaviour* mengikuti ujian dan masuk kelas.



### Latihan



Dari gambar disamping, identifikasi kelas dan obyek yang add, dan juga sebutkan *State* dan *behaviour* nya.

Setiap obyek mempunyai identitas yang unik, seperti halnya setiap orang mempunyai identitas yang unik. Contoh : Siswa mempunyai Nim dimana nim seorang siswa berbeda dengan siswa yang lain.

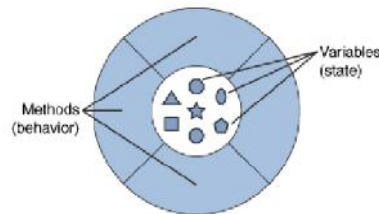
Dua obyek mungkin memiliki *behaviour* yang sama, *State* yang mungkin sama atau mungkin tidak sama, dan identitas yang tidak akan pernah sama. Identitas sebuah obyek tidak akan berubah pada waktu hidup nya (*lifetime*).





### C. Komponen Kelas

Dalam pemrograman berorientasi obyek, kita membuat obyek perangkat lunak (*software*) yang memodelkan obyek dunia nyata. Obyek perangkat lunak dimodelkan setelah adanya obyek di dunia nyata dimana mereka juga mempunyai *State* dan *behaviour*. Sebuah obyek mengatur *State* nya dalam satu atau lebih **variabel/atribut** dan mengimplementasikan *behaviour* dengan **method**. *Method* merupakan suatu fungsi yang berasosiasi dengan sebuah obyek. Kita dapat mengakses data dari sebuah kelas dengan menggunakan *method*. Contoh : terdapat banyak obyek siswa, tiap obyek mempunyai *instance* variabel dan tiap obyek mempunyai nilai yang berbeda yang disimpan dalam *instance* variabel. Seperti nim, seorang siswa akan mempunyai nilai yang berbeda untuk nimnya dan akan disimpan di atribut `StudentNumber`



Gambar III-1 Kelas

Sebuah kelas merupakan cetak biru yang mendefinisikan atribut dan *method*. Setelah membuat kelas maka kita dapat membuat obyek dari kelas tersebut. Sebuah kelas dapat menjadi pabrik untuk membangun obyek.

### D. Membuat Kelas dan Obyek di Java

Struktur dasar pemrograman dengan aplikasi Java adalah kelas dan obyek. Aplikasi java terdiri dari kelas-kelas dimana terdapat deklarasi atribut dan *method*. Sebuah obyek merupakan *instance* dari kelas dan akan mengenkapsulasi *method* dan atribut pada suatu kelas. Obyek dapat digunakan sepanjang program.

Sebuah kelas mendefinisikan obyek dan karakteristiknya. Komponen utama dari sebuah kelas adalah atribut dan *method*. Di dalam kelas terdapat statemen yang didalamnya termasuk deklarasi atribut untuk menyimpan nilai dan *method* yang didalamnya terdapat kode-kode untuk mengeluarkan nilai yang diharapkan. Blok kelas ditandai dengan tanda {}, yang mengindikasikan permulaan dan akhir dari suatu kelas.



## 1. Pembuatan kelas di Java

Atribut dan *method* dideklarasikan di dalam kelas. Semua *Statement* di dalam Java diakhiri dengan tanda ";". Di bawah ini Sintaks untuk mendeklarasikan kelas

```
class <classname>
{
    //declaration of data member
    //declaration of methods
}
```

Kata *class* merupakan *keyword* pada Java yang digunakan untuk mendeklarasikan kelas dan <classname> digunakan untuk memberikan nama kelas. Nama kelas akan dibutuhkan ketika pembuatan obyek dari kelas tersebut. Tanda *slash* ganda, // digunakan untuk memberikan komentar dan tidak akan dieksekusi. Kelas kosong (*Empty class*) merupakan kelas yang didalamnya tidak terdapat atribut dan *method*. Berikut adalah contoh pendeklarasian kelas librarian :

```
class Librarian
{
    String librarianName;        //data member
    int librarianID;             //data member
    void getLibrarianName() {}; //method
}
```

## 2. Pembuatan Obyek di Java

Obyek merupakan *instance* dari class dan mempunyai identitas yang unik. Kelas merupakan abstraksi dari *property* umum obyek. Langkah-langkah pembuatan obyek :

- a. Deklarasi : mendeklarasikan suatu variabel yang akan menyimpan referensi ke obyek

```
<class name> <object name>;
Librarian Lib1;
```

- b. Instansiasi atau pembuatan : Membuat obyek dari kelas yang diinginkan. Ketika sebuah obyek dideklarasikan. Memori belum dialokasikan sehingga tidak dapat dilakukan penyimpanan data obyek. Operator yang harus digunakan untuk mengalokasikan memori adalah :

```
<object name> = new <class name>();
Lib1 = new Librarian();
```

Kedua langkah diatas dapat disatukan menjadi sebuah *Statement* :

```
Librarian Lib1 = new Librarian();
```



## **E. Access Specifiers dan Access Modifiers**

Kelas memungkinkan obyek untuk mengakses atribut atau *method* dari kelas lain. Java menyediakan *Access Specifiers* dan *Modifiers* untuk menentukan bagian dari kelas yang boleh diakses oleh kelas lain dan bagaimana penggunaan atribut dan *method* tersebut. Penggunaan *Access Specifiers* merupakan implementasi dari fitur OOP Encapsulation.

### **1. Access Specifiers**

*Access Specifiers* mengontrol pengaksesan dari bagian kelas oleh obyek yang lain. *Access Specifiers* di Java adalah : *Public*, *Private*, *Protected* dan *Friendly*.

#### **a. Access Specifiers Public**

*Access Specifiers* public membuat kelas dapat diakses dari manapun termasuk dari kelas lain dan kelas-kelas yang berada dalam package yang berbeda. *Keyword* public digunakan untuk mendeklarasikan bahwa bagian kelas tersebut public. *Access Specifiers* ini dapat digunakan baik untuk *method*, atribut, maupun kelas.

```
public <data type> <variable name>;  
public string librarianName;
```

#### **b. Access Specifiers Private**

*Access Specifiers* Private digunakan untuk menyediakan akses yang sangat terbatas. Bagian yang dideklarasikan sebagai private hanya dapat diakses oleh anggota kelas itu sendiri. *Keyword* private digunakan untuk mendeklarasikan *Access Specifiers* private.

```
private <data type> <variable name>;  
private int librarianID;
```

#### **c. Access Specifiers Protected**

*Access Specifiers* protected digunakan jika anggota kelas dapat diakses oleh kelas turunan. *Keyword* protected digunakan untuk mendeklarasikan anggota sebagai protected.

```
protected <data type> <variable name>;  
protected string librarianName;
```

#### **d. Access Specifiers Friendly**

Jika kita lupa memberikan atau tidak memberikan suatu *Access Specifier* pada anggota kelas, maka secara otomatis *Access Specifiers*nya adalah *friendly*. *Access Specifiers* ini mempunyai akses seperti *public* tetapi hanya untuk kelas yang berada dalam satu package. *Access*



Apecifiers `friendly` tidak memerlukan *keyword*, karena ini hanya jenis akses level. *Friendly* merupakan default *Access Specifiers* pada Java.

```
<data type> <variable name>;  
int librarianID;
```

## 2. **Access Modifiers**

*Access Modifiers* digunakan untuk mendefinisikan bagaimana suatu atribut dan *method* digunakan oleh kelas dan obyek yang lain. Perbedaan utama *Access Specifiers* dan *Modifiers* adalah, *Access Specifiers* mendefinisikan hak akses anggota kelas, sedangkan *Modifiers* digunakan untuk mendefinisikan bagaimana suatu *method* atau atribut dapat dimodifikasi oleh kelas yang lain. *Modifiers* di Java adalah : `static`, `final`, `abstract`, `native`, `synchronized`.

### a. **Access Modifiers static**

Atribut dan *method* dapat menjadi statik atau tidak statik. Bagian kelas (atribut dan *method*) yang tidak statik disebut sebagai atribut *instance* dan *method instance*, sedangkan bagian kelas yang statik disebut sebagai atribut kelas dan *method* kelas yang dimiliki oleh kelas dan bukan *instance* dari kelas.

```
static int LibrarianID; //static attribute  
int LibrarianID;      //non static attribute  
static void getLibrarianName();//static methods  
void getLibrarianName();//non static method
```

Setiap *instance* dari kelas (*object*) mendapatkan copy-an atribut *instance* dan *method instance* yang didefinisikan di dalam kelas. Atribut kelas membawa informasi yang dibagi ke semua *instance* dari kelas, jika satu obyek mengubah atribut maka akan mengubah semua obyek yang berasal dari kelas yang sama. *Method* kelas dapat digunakan secara langsung tanpa harus membuat *instance* (obyek), sedangkan jika akan mengakses *method instance* maka harus membuat *instance* dari kelas terlebih dahulu (obyek).

### b. **Access Modifiers final**

*Modifiers final* mengindikasikan bahwa anggota kelas tidak dapat dimodifikasi. *Keyword final* digunakan dalam *method*, atribut, dan kelas. Jika sebuah atribut telah dideklarasikan *final* maka kita tidak dapat mengubah nilai atribut tersebut, *final* juga membuat kelas tidak dapat diturunkan (*inheritance*) karena *method* yang *final* tidak dapat dimodifikasi di dalam kelas turunan.



**c. Access Modifiers abstract**

Kelas abstract digunakan sebagai kelas dasar yang nantinya akan diturunkan ke kelas lainnya. Kelas abstrak selalu mempunyai kelas turunan.

**d. Access Modifiers native**

Modifier native digunakan hanya untuk *method*. Modifier ini digunakan untuk menginformasikan compiler bahwa *method* tersebut telah dikodekan dengan bahasa pemrograman selain Java, seperti C atau C++.

**e. Access Modifiers synchronized**

Modifier synchronize digunakan hanya untuk *method*. Modifier ini digunakan untuk mengontrol akses ke bagian kode ketika sedang menjalankan sistem secara multithread (banyak thread).

## F. Atribut dan Method

Sebaiknya memberikan nilai ke atribut sebelum obyek digunakan. Perhatikan contoh dibawah untuk melihat bagaimana kelas dan obyek bekerja. Contoh kelas student, dimana kelas ini akan menyimpan informasi tentang student

### 1. Mengakses atribut

Atribut dapat diakses dari luar kelas dengan cara memanggil nama obyek kemudian diikuti tanda titik dan nama dari atribut. Sintaksnya sebagai berikut :

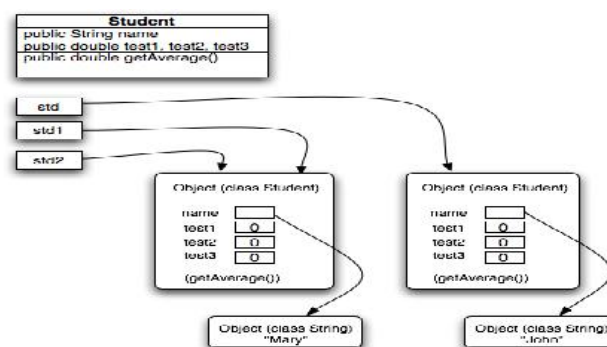
`<object name>.<attribute name> = <value>`

Contoh penggunaannya :

`Std.name = "John";`

`Std1.name = "Mary";`

Jika suatu atribut *instance* tidak diisi oleh suatu nilai maka atribut akan diisi nilai *default*.



Gambar III-2 Mengakses Atribut



Kode `std = new Student();` digunakan untuk mengalokasikan memori ke variabel `std` dan juga variabel `std1`, sehingga sebuah obyek baru telah dibuat. Kode `std2 = std1` digunakan untuk mengalamatkan nilai `std1` ke `std2`, tidak ada obyek baru yang dibuat, `std2` merupakan sebuah variabel yang menunjuk ke memori dimana dialokasikan.

Kode `Std.name = "John";` digunakan untuk memberikan nilai "John" ke atribut `std.name` dan `Std1.name = "Mary";` digunakan untuk memberikan nilai "Mary" ke atribut `std1.name`, dan karena `std2` menunjuk ke memori yang mengalokasikan `std1` maka `std2.name` adalah "Mary". Atribut lain yang tidak diberikan nilai akan diberikan nilai *default*, dimana disini diberikan nilai 0.

## 2. Mengakses Method

Sintaks untuk mendefinisikan *method* :

```
void <method name> () {  
    // Method body  
}
```

Di dalam Sintaks diatas, kata `void` menspesifikasikan bahwa fungsi tidak mengembalikan nilai, *<method name>* menspesifikasikan nama dari *method* tersebut.

```
public void getLibrarianName() {  
    sistem.out.println("Name : ", +librarianName)  
}
```

Pada contoh diatas nama *method* adalah dan tidak mengembalikan nilai.

```
public double getAverage() {  
    return((test1+test2+test3)/3);  
}
```

Pada contoh diatas nama *method* adalah dan mengembalikan sebuah nilai dengan tipe data `double`.

## G. Method main()

Sebuah program Java mempunyai *method* `main()`, *method* ini akan memanggil *method-method* yang dideklarasikan di dalam kelas. Kita dapat membuat beberapa kelas di dalam program Java. *Compiler* Java akan meng-*compile* semua kelas di dalam program, tetapi untuk menjalankan program tersebut dibutuhkan *method* `main()` di dalam program, Sintaks *method* `main()` :



```
public static void main(String[] args) {  
    //code for the main method  
}
```

Header pada *method* *main* terdapat 3 *keyword* yaitu :

- o *Public* : *keyword public* mengindikasikan bahwa *method* dapat diakses dari obyek manapun di dalam program java
- o *Static* : *keyword static* digunakan dengan *method* *main()*, sehingga tidak perlu membuat obyek untuk memanggil *method* *main()*
- o *void* : *keyword void* mengindikasikan bahwa *method* tersebut tidak mengembalikan nilai.

Parameter mengindikasikan bahwa *method* *main()* akan menerima sebuah argumen dalam bentuk *array* dengan bertipe *String*. *Method* *main()* dapat dideklarasikan di kelas manapun tetapi nama file dan nama kelas tempat *method* *main()* dideklarasikan harus sama.

## H. Constructor (Konstruktor)

Konstruktor merupakan *method* yang akan dieksekusi secara otomatis ketika sebuah obyek dibuat. *Method* ini mempunyai nama yang sama dengan nama kelas tempat konstruktor dideklarasikan. Konstruktor tidak mengembalikan nilai dan tidak dapat dideklarasikan sebagai *static*. Konstruktor biasanya digunakan untuk memberikan nilai pada atribut ketika obyek dibuat dari kelas.

Sintaks konstruktor adalah :

```
public class <class name> {  
    <constructor name same with class name>();  
}
```

Contoh penggunaan konstruktor adalah :

```
public class Student {  
    public String name;//attribute  
    public double test1, test2, test3;//attribute  
  
    Student(String theName) { //Konstruktor  
        name = theName;  
    }  
  
    public double getAverage() {  
        return((test1+test2+test3)/3);  
    }  
}
```



## Praktek

Dibutuhkan untuk menyimpan data mahasiswa dalam Aplikasi akademik. Setiap mahasiswa akan disimpan nim, nama, nilai UTS dan UAS untuk menghitung nilai akhir. Untuk mengimplementasikan permasalahan diatas maka dibuatlah program Java, karena mewakili permasalahan nyata yaitu pada suatu institusi pendidikan terdapat banyak obyek mahasiswa. Untuk membuat obyek mahasiswa terlebih dahulu dibuat kelas Mahasiswa

```
// creating file MainProgramAkademik.java //
class Student {
    private String studentId;
    private String studentName;
    private int scoreUTS,scoreUAS;

    public Student(String name, String id) {
        studentId = id;
        studentName = name;
    }
    public double getTotalScore() {
        return((scoreUTS+scoreUAS)/2);
    }
    public setScore(int UAS,int UTS) {
        scoreUAS = UAS;
        scoreUTS = UTS;
    }
    public void show() {
        System.out.println("Student ID : "+studentId);
        System.out.println("Student Name : "+studentName);
        System.out.println("Total Score : "+getTotalScore());
    }
}

public class MainProgramAkademik {
    public static void main(String[] args) {
        Student std = new Student("Mary","R12300123");
        std.setScore(90,80);
        std.show();
    }
}

//Running in command prompt
javac MainProgramAkademik.java
java MainProgramAkademik
```





## Rangkuman

1. Kelas mendefinisikan struktur (*structure*) dan tingkah laku (*behaviour*) sebuah obyek atau sekumpulan obyek. Obyek merupakan *instance* dari kelas.
2. Obyek mempunyai dua karakteristik : *State* dan *behaviour*. *State* merupakan sekumpulan atribut dan nilai dari atribut tersebut. *Behaviour* merupakan aksi dan reaksi obyek ketika terjadi perubahan *State* dan pengiriman pesan.
3. Java menyediakan *Access Specifiers* dan *Modifiers* untuk menentukan bagian dari kelas yang boleh diakses oleh kelas lain dan bagaimana penggunaan atribut dan *method* tersebut.
4. *Header* pada *method* main terdapat 3 *keyword* yaitu *public*, *static*, dan *void*.
5. *Method* *main()* dapat dideklarasikan di kelas manapun tetapi nama file dan nama kelas tempat *method* *main()* dideklarasikan harus sama.
6. Konstruktor merupakan *method* yang akan dieksekusi secara otomatis ketika sebuah obyek dibuat dan mempunyai nama yang sama dengan nama kelas tempat konstruktor dideklarasikan.

## Latihan

1. Pada suatu sistem supermarket dibutuhkan untuk menyimpan data pembelian barang-barang yang dibeli oleh konsumen. Tiap barang yang ada didalam supermarket mempunyai nomor *barcode*, nama barang, pabrik yang memproduksi, dan harga. Buatlah deklarasi kelas untuk class Barang
2. Pada kelas barang, harga barang tidak boleh diakses oleh kelas lain meskipun turunannya, hanya boleh diakses oleh kelas barang saja. Aplikasikan syarat ini pada kelas Barang
3. Buatlah sebuah main class untuk mengakses kelas barang.
4. Perhatikan code di bawah ini

```
public class Pegawai {  
    private String idPegawai;  
    String namePegawai;  
    public addressPegawai;  
    public String getIdPegawai() {  
        return idPegawai;  
    }  
}
```



```
public void setIdPegawai(String Id) {  
    idPegawai = Id;  
}  
public static void main (String[] argument)  
{  
    Pegawai peg1 = new Pegawai();  
    Pegawai.setIdPegawai("P23001");  
}
```

Apabila terdapat kesalahan Sintaks, koreksi Sintaks diatas sehingga dapat dieksekusi.



## BAB IV Hubungan Antar Kelas dan Pewarisan

Ulasan	Tujuan
Bab ini akan menjelaskan tentang hubungan antar kelas dan implementasinya pada Java. Berisi tentang hubungan asosiasi, agregasi, komposisi, diawali dengan penjelasan <i>array</i> pada Java. Kemudian dilanjutkan dengan penjelasan tentang pewarisan kelas, kelas abstrak, dan juga <i>interface</i>	<ol style="list-style-type: none"><li>1. Mengetahui konsep <i>array</i> di Java</li><li>2. Mengetahui hubungan antar kelas dan implementasinya di Java</li><li>3. Mengetahui pewarisan kelas di Java</li><li>4. Mengetahui konsep kelas abstrak di Java</li><li>5. Mengetahui konsep <i>interface</i> di Java</li></ol>

### A. Array

*Array* biasanya digunakan untuk mengelompokkan data atau obyek yang memiliki tipe yang sama. *Array* memungkinkan untuk mengacu ke sekumpulan obyek dengan nama yang sama.

#### 1. Mendeklarasikan *Array*

*Array* dapat dideklarasikan dengan tipe apa saja, baik itu yang bertipe data primitif maupun obyek. Berikut contoh deklarasi *Array* :

```
char s[];  
point p[]; // point adalah sebuah kelas
```

Dalam bahasa pemrograman Java, *Array* merupakan sebuah obyek meskipun ia terdiri dari elemen yang bertipe data primitif. Seperti halnya kelas yang lain, ketika mendeklarasikan *Array* belum dibentuk sebuah obyek *Array*. Deklarasi *Array* hanya membuat sebuah referensi yang dapat digunakan untuk mengacu ke sebuah obyek *Array*.

Besarnya memori yang digunakan oleh elemen-elemen *Array* akan dialokasikan secara dinamis dengan menggunakan pernyataan *new* atau dengan *array initializer*.

Contoh deklarasi *array* di atas, dengan menggunakan kurung siku setelah nama variabel, merupakan standar penulisan *array* dalam C, C++ dan Java. Format demikian agak sulit untuk dibaca. Oleh karenanya, bahasa Java menggunakan bentuk deklarasi *array* alternatif dengan menggunakan kurung siku di sebelah kiri seperti dalam contoh berikut :



```
char[] s;  
point[] p; // point adalah sebuah kelas
```

Sebuah deklarasi *array* dapat ditulis sebagai pendeklarasian tipe *array* di bagian kiri dan nama variabel di bagian kanan. Kedua bentuk deklarasi *array* di atas dapat digunakan, tetapi sebaiknya konsisten dalam menggunakan satu bentuk saja. Dalam deklarasi *array*, tidak ditentukan ukuran aktual dari *array*. Ketika mendeklarasikan *array* dengan kurung siku yang berada di sebelah kiri nama variabel, kurung siku tersebut berlaku bagi semua variabel yang berada di sebelah kanannya.

## 2. Membuat *Array*

Seperti halnya dalam pembuatan obyek, *Array* dibuat dengan menggunakan *keyword* *new*. Contoh di bawah ini membuat sebuah *Array* dengan tipe primitif *char* :

```
s = new char[26];
```

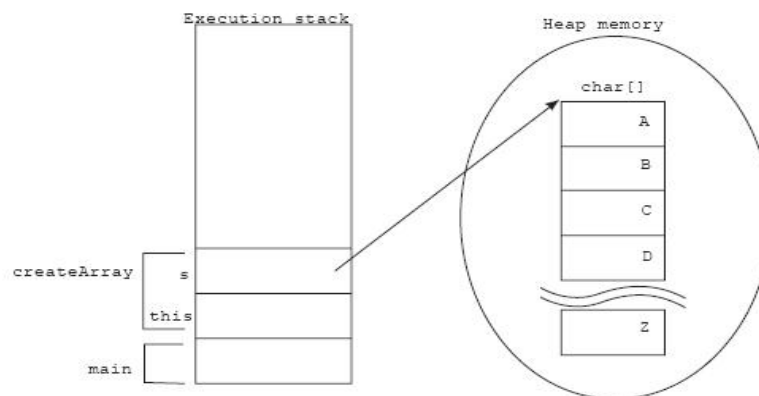
Baris di atas membuat sebuah *array* dengan 26 nilai *char*. Ketika membuat sebuah *Array*, elemen di dalamnya akan diinisialisasi dengan nilai default ('`\u0000`' untuk data tipe *char*). Biasanya nilai elemen *Array* dapat diinisialisasi dengan nilai *default* sebagaimana ditunjukkan berikut ini :

```
s[0] = 'A';  
s[1] = 'B';  
...
```

Perhatikan penggalan kode sebagai berikut :

```
public char[] createArray {  
    char[] s;  
    s = new char[26];  
    for(int i=0; i<26; i++) {  
        s[i] = (char)('A' + i);  
    }  
    return s;  
}
```

Penggalan kode di atas mempunyai representasi dalam heap memory sebagai berikut :



Gambar IV-1 Representasi *Array* Primitif dalam *Heap Memory*

Indeks *array* dimulai dengan angka 0. Batasan legal untuk nilai indeks ini adalah dari nol hingga jumlah elemen *array* – 1. Apabila program berusaha mengakses *array* di luar batas yang legal, maka akan muncul *runtime exception*. Untuk membuat *Array* dengan elemen obyek, gunakan cara penulisan berikut :

```
p = new point[10];
```

Pada kode di atas, dibuat sebuah *Array* yang masing-masing elemennya merupakan referensi obyek yang bertipe *Point*. Pernyataan tersebut tidak membuat 10 obyek *Point*. Untuk membuat obyek *Point*, gunakan pernyataan berikut :

```
p[0] = new point(0, 1);  
p[1] = new point(1, 2);  
...
```

### 3. Menginisialisasi *Array*

Ketika membuat sebuah *Array*, setiap elemen di dalamnya akan diinisialisasi dengan nilai *default*. Dalam kasus *Array* *char* *s* seperti pada contoh di atas, setiap elemen dari *Array* *s* diinisialisasi dengan nilai `'\u0000'`. Dalam kasus *Array* *p* seperti juga contoh di atas, setiap elemen diinisialisasi dengan nilai *null*, yang berarti elemen-elemen *array* *p* tersebut belum merujuk ke suatu obyek *Point*. Setelah proses pengisian `p[0] = new Point()`, barulah diperoleh elemen pertama dari *Array* *p* yang merujuk ke obyek *Point*. Java juga memiliki cara singkat untuk membuat sebuah obyek *Array* dengan elemen-elemennya memiliki nilai awal :

```
String names[] = {"Georgianna", "Jen", "Simon"};
```

Kode di atas adalah sama dengan kode berikut ini:



```
String names[];  
names[0] = "Georgianna";  
names[1] = "Jen";  
names[2] = "Simon";
```

Cara singkat ini dapat digunakan untuk *Array* dengan berbagai jenis tipe, seperti dalam contoh berikut :

```
MyDate dates[] = {    new MyDate(22, 7, 1964),  
                    new MyDate(1, 1, 2000),  
                    new MyDate(22, 12, 1964)    };
```

#### 4. **Array Multidimensi**

Dalam Java, dapat dibuat *Array* dari *Array* sehingga dinamai *Array* multidimensi. Contoh berikut ini memperlihatkan cara membuat *Array* dua dimensi :

```
int twoDim [][] = new int[4][];  
twoDim[0] = new int[5];  
twoDim[1] = new int[5];
```

Pada baris pertama, dibuat sebuah obyek *Array* yang memiliki empat elemen. Masing-masing elemen merupakan referensi yang bertipe *Array* dari *int* yang bernilai *null*. Pada baris berikutnya, diinisialisasi elemen pertama dan kedua dengan obyek *Array* dari *int* yang masing-masing berisi lima elemen yang bertipe *integer*.

*Array* dari *Array* yang bersifat *non-rectangular* dapat dibuat seperti dalam contoh berikut :

```
twoDim[0] = new int[2];  
twoDim[1] = new int[4];  
twoDim[2] = new int[6];  
twoDim[3] = new int[8];
```

Proses pembuatan *Array* dari *Array* yang demikian, di mana harus diinisialisasi masing-masing elemennya satu persatu, cukup merepotkan. Di samping itu, *array* dari *array* yang berbentuk *rectangular* lebih sering digunakan dibandingkan dengan yang *nonrectangular*.

Untuk membuat *Array* dari *Array* yang *rectangular* dengan cara mudah, gunakan bentuk berikut ini :

```
int twoDim[][] = new int[4][5];
```

Kode di atas membuat sebuah *Array* dari empat *Array* yang masing-masing elemennya berisi lima nilai *int*.



## 5. Batasan Array

Dalam bahasa Java, index *Array* dimulai dengan angka nol. Jumlah elemen di dalam *Array* disimpan sebagai bagian dari obyek *Array* di dalam atribut `length`. Atribut ini dapat digunakan untuk melakukan proses iterasi pada *Array* seperti dalam contoh berikut :

```
int list[] = new int[10];
for(int i = 0; i < list.length; i++) {
    System.out.println(list[i]);
}
```

Dengan menggunakan atribut `length`, pemeliharaan kode program menjadi mudah karena program tidak perlu mengetahui jumlah elemen *Array* pada saat kompilasi.

## 6. Manipulasi Array

### a. Mengubah Ukuran Array

Setelah membuat obyek *Array*, ukuran *Array* tersebut tidak dapat diubah. namun demikian, dapat digunakan variabel referensi yang sama untuk menunjuk ke sebuah obyek *Array* baru seperti dalam contoh di bawah ini :

```
int myArray[] = new int[6];
myArray = new int[10];
```

Dalam kode program ini, obyek *Array* yang pertama akan hilang kecuali ada variabel lain yang dibuat untuk merujuk ke obyek *Array* tersebut.

### b. Menyalin Array

Bahasa Java menyediakan *method* khusus untuk menyalin *Array*, yaitu dengan menggunakan *method* `arrayCopy()` dari kelas `System` seperti dalam contoh berikut :

```
// array semula
int myArray[] = {1, 2, 3, 4, 5, 6};
// array engan elemen yang lebih banyak
int hold[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
//menyalin semua elemen myArray ke hold
// dimulai dengan indeks ke 0
System.arraycopy(myArray, 0, hold, 0, myArray.length);
```

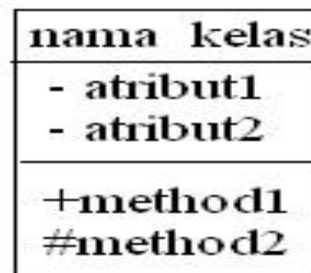
Setelah proses pengkopian di atas, maka isi dari *Array* `hold` adalah 1, 2, 3, 4, 5, 6, 4, 3, 2, 1.



## B. Diagram Kelas

Diagram kelas merupakan sebuah diagram yang digunakan untuk memodelkan kelas-kelas yang digunakan di dalam sistem beserta hubungan antar kelas dalam sistem tersebut.

Beberapa elemen penting dalam diagram kelas adalah kelas dan relasi antar kelas. Kelas digambarkan dengan simbol kotak seperti gambar di bawah ini :



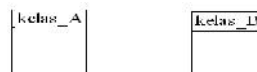
Gambar IV-2 Simbol Diagram Kelas

Baris pertama dari simbol diagram kelas menandakan nama dari kelas yang bersangkutan. Baris di bawahnya menyatakan atribut-atribut dari kelas tersebut apa saja, dan baris setelahnya menyatakan *method-method* yang terdapat pada kelas tersebut. Adapun simbol untuk *access modifier* adalah sebagai berikut:

- Untuk *public* diberi simbol + sebelum nama atribut/*method*
- Untuk *private* diberi simbol - sebelum nama atribut/*method*
- Untuk *protected* diberi simbol # sebelum nama atribut/*method*

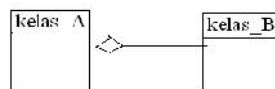
Sedangkan untuk menggambarkan hubungan antar kelas digunakan simbol garis antara dua kelas yang berelasi. Simbol garis tersebut antara lain :

- Kelas A berasosiasi dengan kelas B, digambarkan sebagai berikut:



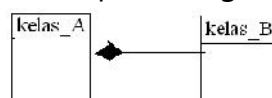
Gambar IV-3 Asosiasi

- Kelas B merupakan elemen part-of dari kelas A (kelas A berelasi agregasi dengan kelas B), digambarkan sebagai berikut:



Gambar IV-4 Agregasi

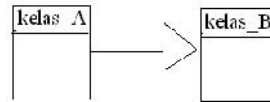
- Kelas A dengan kelas B berelasi komposisi, digambarkan sebagai berikut:



Gambar IV-5 Komposisi

- Kelas A merupakan turunan dari kelas B, digambarkan sebagai berikut:





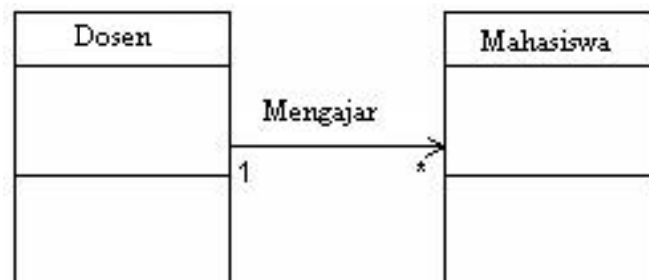
Gambar IV-6 Turunan

### C. Hubungan Antar Kelas

Dalam Obyek Oriented Programming, kelas-kelas yang terbentuk dapat memiliki hubungan satu dengan yang lainnya, sesuai dengan kondisi dari kelas-kelas yang bersangkutan. Ada beberapa jenis hubungan yang dapat terjadi antara kelas yang satu dengan kelas yang lainnya, antara lain : Asosiasi, Agregasi, Komposisi, Generalisasi (terkait dengan pewarisan), Spesialisasi (terkait dengan pewarisan)

#### 1. Asosiasi

Asosiasi merupakan hubungan antara dua kelas di yang merupakan hubungan struktural yang menggambarkan himpunan link antar obyek. Contoh dari hubungan asosiasi ini adalah :



Gambar IV-7 Contoh hubungan asosiasi

Pada diagram kelas di atas terlihat hubungan bahwa kelas dosen mengajar beberapa mahasiswa. Bentuk implementasi dari diagram kelas tersebut di Java adalah sebagai berikut :

```
//mahasiswa.java
public class mahasiswa {
    private String nim;
    private String nama;

    public void setnama (String nama) {
        this.nama = nama;
    }
    public void setnim (String nim) {
        this.nim = nim;
    }
    public String getnim () {
        return this.nim;
    }
    public String getnama () {
```



```
        return this.nama;
    }
}

//dosen.java
public class dosen {
    private String Kddosen;
    private String [] nimMHS;
    private int JmlMahasiswa;

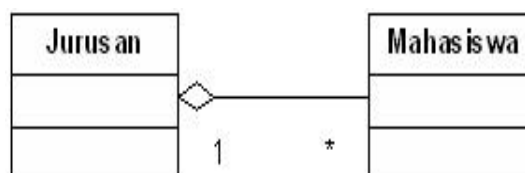
    public void setKddosen (String Kddosen) {
        this.Kddosen = Kddosen;
    }
    public void setNimMahasiswa (String nimMhs) {
        if (JmlMahasiswa<5) {
            nimMHS[JmlMahasiswa] = nimMhs;
            JmlMahasiswa++;
        }
    }

    public int getJmlMahasiswa () {
        return this.JmlMahasiswa;
    }
    public String getKddosen () {
        return this.Kddosen;
    }
    public String getmahasiswa (int i) {
        return (nimMHS[i]);
    }
}
```

Pada implementasi terlihat bahwa tidak ada relasi yang kuat antara kelas dosen dan kelas mahasiswa, hanya ada atribut dari kelas dosen yang serupa dengan atribut dari kelas mahasiswa yang menandakan bahwa kedua kelas itu berasosiasi, yaitu atribut nimMahasiswa pada kelas dosen dan atribut nim pada kelas mahasiswa.

## 2. Agregasi

Agregasi merupakan hubungan antara dua kelas di mana kelas yang satu merupakan bagian dari kelas yang lain namun kedua kelas ini dapat berdiri sendiri-sendiri. Agregasi sering juga disebut relasi "part of" atau relasi "whole-part". Contoh hubungan agregasi ini adalah :



Gambar IV-8 Contoh hubungan agregasi



Pada diagram kelas di atas, terlihat hubungan antara kelas Jurusan dengan kelas Mahasiswa. Kelas mahasiswa merupakan bagian dari kelas jurusan, akan tetapi kelas jurusan dan kelas mahasiswa dapat diciptakan sendiri-sendiri.

Implementasi dari diagram kelas tersebut dalam Java adalah sebagai berikut :

```
//mahasiswa.java
public class mahasiswa {
    private String NIM, Nama;

    public mahasiswa(String no, String nm) {
        this.NIM = no;
        this.Nama = nm;
    }

    public String GetNIM() {
        return (NIM);
    }
    public String GetNama() {
        return (Nama);
    }
}

//jurusan.java
public class Jurusan {
    private String KodeJurusan, NamaJurusan;
    private Mahasiswa[] Daftar = new Mahasiswa[10];

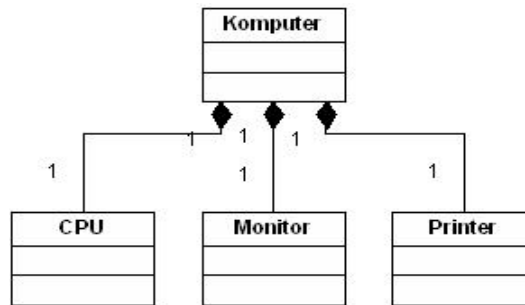
    public Jurusan(String kode, String nama) {
        this.KodeJurusan = kode;
        this.NamaJurusan = nama;
    }
    private static int JmlMhs = 0;
    public void AddMahasiswa(Mahasiswa m) {
        this.Daftar[JmlMhs] = m;
        this.JmlMhs++;
    }
    public void DisplayMahasiswa() {
        int i;
        System.out.println("Kode Jurusan : " +this.KodeJurusan);
        System.out.println("Nama Jurusan : " +this.NamaJurusan);
        System.out.println("Daftar Mahasiswa :");
        for (i=0;i<JmlMhs;i++)
            System.out.println(Daftar[i].GetNIM()+" "+Daftar[i].GetNama());
    }
}
```

Pada implementasi terlihat bahwa kelas jurusan memiliki atribut yang memiliki tipe kelas mahasiswa, sehingga kelas mahasiswa merupakan bagian dari kelas jurusan.



### 3. Komposisi

Komposisi merupakan bentuk khusus dari agregasi di mana kelas yang menjadi part (bagian) baru dapat diciptakan setelah kelas yang menjadi whole (seluruhnya) dibuat dan ketika kelas yang menjadi whole dimusnahkan, maka kelas yang menjadi part ikut musnah. Contoh hubungan komposisi adalah sebagai berikut :



Gambar IV-9 Contoh hubungan komposisi

Dari diagram kelas di atas terlihat bahwa kelas CPU, Monitor, dan Printer semuanya merupakan bagian dari kelas Komputer dan ketika kelas Komputer musnah maka kelas CPU, Monitor, dan Printer akan ikut musnah.

Implementasi dari diagram kelas tersebut dalam Java adalah sebagai berikut:

```
//CPU.java
public class CPU {
    private String Merk;
    private int Kecepatan;
    public CPU(String m, int k)
    {
        this.Merk = m;
        this.Kecepatan = k;
    }
    public void DisplaySpecCPU() {
        System.out.println(this.Merk + ", " + this.Kecepatan);
    }
}

//Monitor.java
public class Monitor {
    private String Merk;
    public Monitor(String m) {
        this.Merk = m;
    }
    public void DisplaySpecMonitor() {
        Sistem.out.println(this.Merk);
    }
}

//Printer.java
public class Printer {
    private String Merk, Type;
```



```
public Printer(String m, String t)  {
    this.Merk = m;
    this.Type = t;
}
public void DisplaySpecPrinter()  {
    Sistem.out.println(this.Merk + ", " + this.Type);
}
}

//Komputer.java
public class Komputer {
    private String Kode;
    private long Harga;
    private CPU Proc;
    private Monitor Mon;
    private Printer Prn ;

    public Komputer(String k, long h) {
        this.Kode = k;
        this.Harga = h;
        if (k == "PC-01") {
            Proc = new CPU("Pentium IV", 500);
            Mon = new Monitor("Sony Multiscan 15sf");
            Prn = new Printer("Canon BJC-210SP", "Color");
        }
    }
    public void DisplaySpec()  {
        System.out.println("Kode      : " + this.Kode);
        System.out.print("Processor: ");
        Proc.DisplaySpecCPU();
        System.out.print("Monitor  : ");
        Mon.DisplaySpecMonitor();
        System.out.print("Printer  : ");
        Prn.DisplaySpecPrinter();
        System.out.println("Harga : Rp. " + this.Harga);
    }
}
```

Pada implementasi di atas, terlihat bahwa kelas CPU, Monitor, dan Printer merupakan atribut dari kelas Komputer dan baru diciptakan pada saat instansiasi obyek dari kelas Komputer.

#### **D. Pewarisan**

Pewarisan adalah kemampuan sebuah kelas untuk mewariskan seluruh atau sebagian atribut dan *method*nya ke kelas lain, sehingga atribut dan *method* tersebut dikenal oleh kelas yang menerima pewarisan tanpa harus menuliskannya. Pewarisan ini merupakan implementasi dari hubungan antar kelas generalisasi-spesialisasi.



Kelas yang mewariskan disebut kelas induk, super class, atau base class sedangkan kelas yang menerima pewarisan disebut kelas anak, kelas turunan, atau *subclass*.

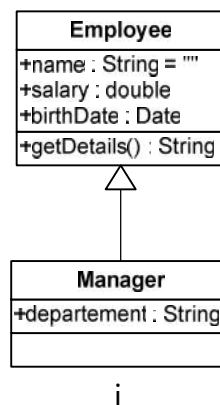
Pewarisan dapat dilakukan jika :

- Ada beberapa atribut dan *method* yang sama yang digunakan oleh beberapa kelas berbeda (reduksi penulisan kode)
- Ada satu atau beberapa kelas yang sudah pernah dibuat yang dibutuhkan oleh aplikasi (*reusability*)
- Ada perubahan kebutuhan fungsional atau *feature* aplikasi dimana sebagian atau seluruh perubahan tersebut tercakup di satu atau beberapa kelas yang sudah ada (*extend*)

### 1. Pewarisan di Java

Pewarisan di Java hanya mengenal pewarisan tunggal, artinya sebuah kelas hanya mewarisi atribut dan *method* dari satu kelas induk. Untuk menggunakan pewarisan di Java digunakan *keyword* `extends`.

Contoh pewarisan dapat dilihat pada diagram kelas berikut ini:



Gambar IV-10 Contoh Pewarisan Kelas

Pada diagram kelas di atas, kelas `Manager` merupakan kelas turunan dari kelas `Employee` sehingga mewarisi semua atribut dan *method* dari kelas `Employee` yang bersifat `public` dan `protected`. Implementasi dari diagram kelas di atas dalam Java adalah sebagai berikut :

```
//Employee.java
public class Employee {
    public String name;
    public Date birthDate;
    public double salary;
    public String getDetails() {...}
}

//Manager.java
```



```
public class Manager extends Employee {  
    public String department;  
}
```

Dari kode di atas, terlihat bahwa atribut `name`, `birthDate`, dan `salary` serta *method* `getDetails()` diturunkan ke kelas `Manager` sehingga kelas `Manager` dapat menggunakan atribut dan *method* tersebut. Konstruktor dari kelas induk tidak dapat diturunkan kepada kelas turunannya.

Untuk menggunakan *method* dan konstruktor dari kelas induk pada kelas anak digunakan *keyword* `super`, contoh:

```
//Employee.java  
public class Employee {  
    private String name;  
    private double gaji;  
    public Employee (String s, double g) {  
        name = s;  
        gaji=g;  
    }  
    public double getgaji() {  
        return gaji;  
    }  
}  
  
//Manager.java  
public class Manager extends Employee {  
    private String alamat;  
    private double tunjangan;  
    private double bonus;  
    public Manager(String nama, String s) {  
        super(nama);  
        alamat = s;  
    }  
    public double getgaji() {  
        return (super.getgaji()+tunjangan+bonus);  
    }  
}
```

## 2. Kelas Abstrak

Kelas abstrak merupakan suatu bentuk khusus dari kelas di mana kelas tersebut tidak dapat diinstansiasi dan digunakan hanya untuk diturunkan ke dalam bentuk kelas konkret atau kelas abstrak berikutnya.

Kelas abstrak dideklarasikan menggunakan *keyword* `abstract`.

Di dalam kelas abstrak dapat dideklarasikan atribut dan *method* sama seperti kelas konkret, namun ada juga *method* abstrak, yaitu suatu *method* yang tidak mempunyai implementasi hanya memiliki deklarasi saja, contoh :

```
public abstract class LivingThing {  
    public void breath(){
```



```
        System.out.println("Living Thing breathing...");
    }
    public void eat(){
        System.out.println("Living Thing eating...");
    }
    public abstract void walk(); //merupakan method abstrak
    {}
```

Ketika sebuah kelas mewarisi sebuah kelas abstrak, kelas tersebut harus mendefinisikan implementasi dari *method* abstrak yang ada dalam kelas induknya. Jika tidak didefinisikan implementasinya, maka kelas tersebut akan menjadi kelas abstrak juga, contoh:

```
public class Human extends LivingThing {
    //implementasi dari method abstrak walk()
    public void walk(){
        Sistem.out.println("Human walks...");
    }
}
```

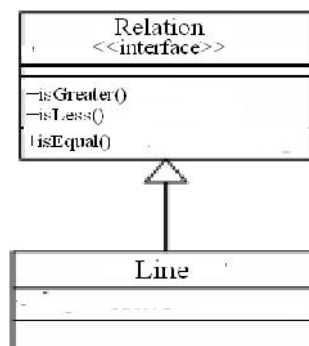
### 3. Interface

*Interface* adalah *prototype* kelas yang berisi definisi konstanta dan deklarasi *method* (hanya nama *method* tanpa definisi kode programnya). Dalam sebuah *interface*:

- Semua atribut adalah *public*, *static* dan *final* (semua atribut bertindak sebagai konstanta)
- Semua *method* adalah *abstract* dan *public*
- Tidak boleh ada deklarasi konstruktor

*Interface* digunakan untuk menyatakan spesifikasi fungsional beberapa kelas secara umum. Dengan adanya *interface*, Java menyediakan sebuah fitur untuk keperluan pewarisan jamak (*Multiple inheritance*).

Contoh *interface* dan kegunaannya:



Gambar IV-11 Contoh Interface





Untuk membuat *interface* di Java digunakan *keyword interface*. *Source code* untuk diagram kelas di atas dalam Java adalah sebagai berikut :

```
//Relation.java
public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}

//Line.java
public class Line implements Relation {
    private double x1;
    private double x2;
    private double y1;
    private double y2;

    public Line(double x1, double x2, double y1, double y2){
        this.x1 = x1;
        this.x2 = x2;
        this.y1 = y1;
        this.y2 = y2;
    }

    public double getLength(){
        double length = Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)* (y2-y1));
        return length;
    }

    public boolean isGreater( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen > bLen);
    }

    public boolean isLess( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen < bLen);
    }

    public boolean isEqual( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen == bLen);
    }
}
```

Untuk menggunakan *interface* dalam keperluan pewarisan jamak, dapat dilakukan dengan cara mengimplementasikan *interface-interface* yang bersangkutan, contohnya :

```
public class Person implements PersonInterface, LivingThing {
    //beberapa kode disini
    .....
    .....
}
```

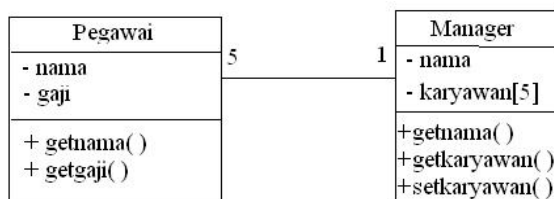


Perbedaan *abstract class* dan *interface* antara lain:

1. Semua *interface method* tidak memiliki *body* sedangkan beberapa *abstract class* dapat memiliki *method* dengan implementasi
2. Sebuah *interface* hanya dapat mendefinisikan *constant* sedangkan sebuah *abstract class* tampak seperti class biasa yang dapat mendeklarasikan variabel
3. *Interface* tidak memiliki hubungan *inheritance* secara langsung dengan sebuah *class* tertentu, sedangkan *abstract class* bisa jadi hasil turunan dari *abstract class* induknya
4. *Interface* memungkinkan terjadinya pewarisan jamak (*Multiple inheritance*) sedangkan *abstract class* tidak

## Essay

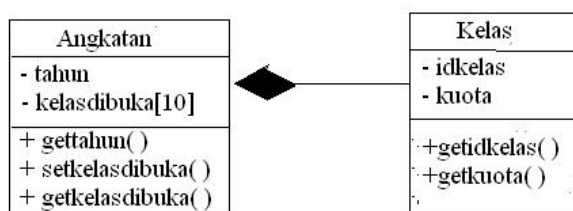
1. Buatlah program java berdasarkan class diagram dibawah ini :



Keterangan diagram kelas di atas:

- Terdapat konstruktor untuk kelas Pegawai dan kelas Manager
- nama bertipe string
- gaji bertipe double
- Karyawan adalah *array* yang bertipe string, berisi nama-nama dari karyawan yang dimiliki Manager
- Setkaryawan( ) digunakan untuk menambahkan jumlah karyawan Manager

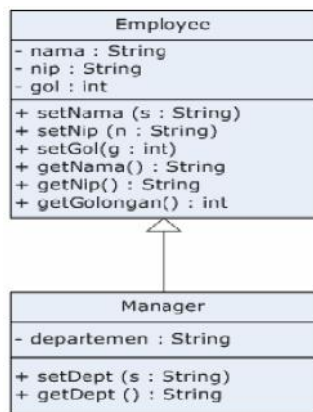
2. Buatlah program java berdasarkan class diagram dibawah ini :



Keterangan diagram kelas di atas:



- Terdapat konstruktor untuk kelas Angkatan dan kelas Kelas
  - tahun bertipe `int`
  - kelasdibuka adalah *array* bertipe `class` untuk menunjukkan jumlah kelas yang dibuka pada angkatan yang bersangkutan
  - Idkelas bertipe `string`
  - Kuota bertipe `int`
  - `setkelasdibuka()` digunakan untuk menambahkan kelas yang dibuka
3. Ubahlah kode Java pada program nomor 2 dengan relasinya adalah agregasi sebagian (shared aggregation).
4. Buatlah kode program java dari class diagram dibawah ini :



Keterangan diagram kelas di atas : Golongan seorang employee berkisar dari 1 s.d. 7



## BAB V. Polimorfisme

### Ulasan :

**Polymorphism** merupakan konsep OOP yang memberikan fleksibilitas kepada programmer dalam menulis program. Dengan mengaplikasikan konsep *polymorphism*, programmer dapat memperlakukan seluruh *object* yang berasal dari *superclass* yang sama seakan-akan mereka adalah *object* dari *superclass*.

Pada Java, konsep *polymorphism* diimplementasikan melalui beberapa cara yaitu *overloading*, *Overriding*, *abstract class* dan melalui *interface*.

### Tujuan :

Dengan praktikum ini mahasiswa diharapkan dapat:

1. Memahami dan menerapkan konsep polimorfisme dalam pemrograman
2. Memahami proses terjadinya Virtual Method Invocation
3. Memahami dan menerapkan polymorphic arguments dalam pemrograman
4. Memahami penggunaan *instanceof* dan cara melakukan casting *object*
5. Memahami tentang *overloading*
6. Memahami tentang *Overriding*
7. Memahami aturan tentang overridden

### A. Polymorphism

Polymorphism berasal dari bahasa Yunani yang berarti banyak bentuk. Dalam PBO, konsep ini memungkinkan digunakannya suatu interface yang sama untuk memerintah obyek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda.

Dalam konsep yang lebih umum sering kali polymorphism disebut dalam istilah satu interface banyak aksi. Contoh yang konkrit dalam dunia nyata yaitu mobil. Mobil yang ada dipasaran terdiri atas berbagai tipe dan berbagai merk, namun semuanya memiliki interface kemudi yang sama, seperti: stir, tongkat transmisi, pedal gas dan rem. Jika seseorang dapat mengemudikan satu jenis mobil saja dari satu merk tertentu, maka orang itu akan dapat mengemudikan hamper semua jenis mobil yang ada, karena semua mobil tersebut menggunakan interface yang sama.

Harus diperhatikan disini bahwa interface yang sama tidak berarti cara kerjanya juga sama. Missal pedal gas, jika ditekan maka kecepatan mobil akan meningkat, tapi bagaiman proses peningkatan kecepatan ini dapat berbeda-beda



untuk setiap jenis mobil. Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi obyek-obyek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki manager, sekretaris, petugas administrasi data dan lainnya.

Misal manager tersebut ingin memperoleh data dari bag administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bag administrasi untuk mengambilnya. Pada kasus tersebut seorang manager tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui obyek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar obyek-obyek yang ada karena setiap obyek memiliki deskripsi tugasnya sendiri.

### 1. **Virtual Method Invocation (VMI)**

*Virtual Method Invocation* (VMI) bisa terjadi jika terjadi polimorfisme dan *Overriding*. Pada saat obyek yang sudah dibuat tersebut memanggil overridden *method* pada parent class, kompiler Java akan melakukan invocation (pemanggilan) terhadap *Overriding method* pada *subclass*, dimana yang seharusnya dipanggil adalah overridden *method*. Berikut contoh terjadinya VMI :

```
class Parent {
    int x = 5;
    public void Info() {
        System.out.println("Ini class Parent");
    }
}

class Child extends Parent {
    int x = 10;
    public void Info() {
        System.out.println("Ini class Child");
    }
}

public class Tes {
    public static void main (String [] args)
    {
        Parent tes = new Child();
        System.out.println("Nilai x = ");
        Tes.Info();
    }
}
```



Hasil dari running program diatas adalah sebagai berikut:

```
Nilai x = 5  
Ini class Child
```

## 2. **Polymorphic arguments**

*Polymorphic arguments* adalah tipe suatu parameter yang menerima suatu nilai yang bertipe *subclass*-nya. Berikut contoh dari *polymorphics arguments* :

```
class Pegawai {  
    .....  
}  
class Manejer extends Pegawai {  
    .....  
}  
public class Tes {  
    public static void Proses(Pegawai peg) {  
        .....  
    }  
    public static void main (String [] args) {  
        Manejer men = new Manejer();  
        Proses(men);  
    }  
}
```

## 3. **Pernyataan instanceof**

Pernyataan *instanceof* sangat berguna untuk mengetahui tipe asal dari suatu polymorphic arguments. Untuk lebih jelasnya, misalnya dari contoh program sebelumnya, kita sedikit membuat modifikasi pada class Tes dan ditambah sebuah class baru Kurir, seperti yang tampak dibawah ini:

```
...  
class Kurir extends Pegawai {  
    ...  
}  
  
public class Tes {  
    public static void Proses(Pegawai peg) {  
        if (peg instanceof Manajer) {  
            ...lakukan tugas-tugas manajer...  
        }  
        else if (peg instanceof Kurir) {  
            ...lakukan tugas-tugas kurir...  
        }  
        else {  
            ...lakukan tugas-tugas lainnya...  
        }  
    }  
}
```



```
public static void main (String [] args) {  
    Manejer men = new Manejer();  
    Kurir kur = new Kurir();  
    Proses(men);  
    Proses(kur);  
}  
}
```

Seringkali pemakaian `instanceof` diikuti dengan casting *object* dari tipe parameter ke tipe asal. Misalkan saja program kita sebelumnya. Pada saat kita sudah melakukan `instanceof` dari tipe `Manajer`, kita dapat melakukan casting *object* ke tipe asalnya, yaitu `Manajer`. Caranya adalah seperti berikut:

```
1 ...  
2 if (peg instanceof Manajer)  
3 {  
4     Manajer man = (Manajer) peg;  
5     ...lakukan tugas-tugas manajer...  
6 }  
7 ...
```

## B. Overloading

*Overloading* adalah suatu keadaan dimana beberapa *method* sekaligus dapat mempunyai nama yang sama, akan tetapi mempunyai fungsionalitas yang berbeda. Contoh penggunaan *overloading* dilihat dibawah ini:

- Parameter titik, untuk menggambar titik  
`Gambar(int t1)`
- Parameter titik, untuk menggambar garis  
`Gambar(int t1,int t2)`
- Parameter titik, untuk menggambar segitiga  
`Gambar(int t1,int t2,int t3)`
- Parameter titik, untuk menggambar persegi empat  
`Gambar(int t1,int t2,int t3,int t4)`

*Overloading* ini dapat terjadi pada class yang sama atau pada suatu parent class dan *subclass*-nya. *Overloading* mempunyai ciri-ciri sebagai berikut:

1. Nama *method* harus sama
2. Daftar parameter harus berbeda
3. Return type boleh sama, juga boleh berbeda

### 1. Overloading konstruktor

Tulislah listing program berikut ini dan amati yang terjadi pada saat terjadinya *overloading* pada *method*.



Kita dapat menyembunyikan information dari suatu class sehingga anggota-anggota class tersebut tidak dapat diakses dari luar. Adapun caranya adalah cukup dengan memberikan akses kontrol *private* ketika mendeklarasikan suatu atribut atau *method*. Contoh : `private int nrp;`

Contoh:

```
public class Siswa {  
    private int nrp;  
  
    public void setNrp(int n) {  
        nrp=n;  
    }  
}
```

Konstruktor (konstruktor) adalah suatu *method* yang pertama kali dijalankan pada saat pembuatan suatu obyek. Konstruktor mempunyai ciri yaitu:

- o Mempunyai nama yang sama dengan nama class
- o Tidak mempunyai return type (seperti *void*, *int*, *double*, dan lain-lain)

Contoh:

```
public class Siswa {  
    private int nrp;  
    private String nama;  
    public Siswa(int n, String m) {  
        nrp=n;  
        nama=m;  
    }  
}
```

Suatu class dapat mempunyai lebih dari 1 konstruktor dengan syarat daftar parameternya tidak boleh ada yang sama.

Contoh:

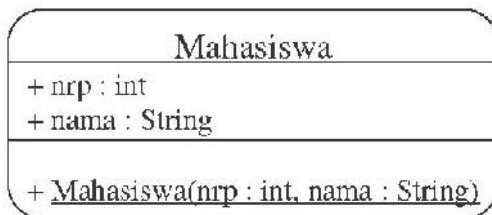
```
public class Siswa {  
    private int nrp;  
    private String nama;  
    public Siswa(String m) {  
        nrp=0;  
        nama="";  
    }  
    public Siswa(int n, String m) {  
        nrp=n;  
        nama=m;  
    }  
}
```



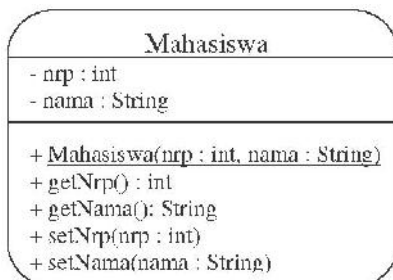


## Latihan

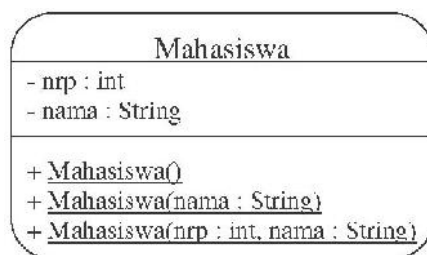
**Percobaan 1** : Melakukan enkapsulasi pada suatu class



Jika enkapsulasi dilakukan pada class diagram diatas, maka akan berubah menjadi:



**Percobaan 2** : Melakukan *overloading* konstruktor



Dari class diagram tersebut, dapat diimplementasikan ke dalam program sebagai berikut:

```
public class Mahasiswa {
    private int nrp;
    private String nama;
    public Mahasiswa() {
        nrp=0;
        nama="";
    }
    public Mahasiswa(String nama) {
        nrp=0;
        this.nama=nama;
    }
    public Mahasiswa(int nrp, String nama) {
        this.nrp=nrp;
        this.nama=nama;
    }
}
```



## 2. Melakukan *overloading* pada *method*

Tuliskan listing program berikut ini dan amati yang terjadi pada saat terjadinya *overloading* pada *method*.

```
import java.awt.Point;
public class Segiempat {
    int x1=0;
    int y1=0;
    int x2=0;
    int y2=0;
    public void buatSegiempat(int x1,int y1, int x2, int y2) {
        this.int x1=x1;
        this.int y1=y1;
        this.int x2=x2;
        this.int y2=y2;
    }
    public void Segiempat(Point topLeft, Point bottomRight) {
        x1 = topLeft.x;
        y2 = topLeft.y;
        x2 = bottomRight.x;
        y2 = bottomRight.y;
    }
    public void buatSegiempat(Point topLeft, int w, int h) {
        x1 = topLeft.x;
        y1 = topLeft.y;
        x2 = (x1 + w);
        y2 = (y1 + h);
    }
    void cetakSegiempat() {
        System.out.print("Segiempat <" +x1 +" " +y1);
        System.out.println(" "+x2 +" "+y2 +" > ");
    }
    public static void main (String [] arguments) {
        Segiempat rect = new Segiempat();
        System.out.println("Buat segiempat dengan koordinat(25,25)
                               dan (50,50)");
        rect.buatSegiemapt(25,25,50,50);
        rect.cetakSegiempat();
        System.out.println();
        System.out.println("Buat segiempat dengan point (10,10) dan
                               point (20,20) : ");
        rect.buatSegiempat(new Point(10,10), new Point(20,20));
        rect.cetakSegiempat();
        System.out.println();
        System.out.print("Buat segiempat dengan 1 point (10,10),
                               koordinat (50,50)");
        rect.buatSegiempat(new Point(10,10),50,50);
        rect.cetakSegiempat();
    }
}
```

Ketika program tersebut dijalankan, akan tampak hasil seperti dibawah ini :

Buat segiempat dengan koordinat (25,25) dan (50,50)

Segiempat: <25, 25, 50, 50>



Buat segiempat dengan point (10,10) dan point (20,20):

segiempat: <10, 10, 20, 20>

Buat segiempat dengan 1 point (10,10), koodinat (50,50)

segiempat:<10, 10, 60, 60>

### C. Overriding

*Overriding* adalah suatu keadaan dimana *method* pada *subclass* menolak *method* pada parent class-nya. *Overriding* mempunyai ciri-ciri sebagai berikut :

1. Nama *method* harus sama
2. Daftar parameter harus sama
3. *return type* harus sama

Berikut ini contoh terjadinya *Overriding* dimana *method* `Info()` pada class `Child` meng-override *method* `Info()` pada class `parent` :

```
class Parent {  
    public void Info() {  
        System.out.println("Ini class Child");  
    }  
}  
  
class Child extends Parent {  
    public void Info() {  
        System.out.println("Ini class Child");  
    }  
}
```

*Method* yang terkena override (*overridden method*) diharuskan tidak boleh mempunyai *modifier* yang lebih luas aksesnya dari *method* yang meng-override (*Overriding method*).



## BAB VI. *Exception Handling*

Ulasan	Tujuan
Bab ini akan menjelaskan tentang <i>Exception</i> di Java, apa yang dimaksud dengan <i>Exception</i> , bagaimana menangkap <i>Exception</i> . Serta akan dijelaskan <i>throwing exception</i> dan membuat <i>user define exception</i> .	<ol style="list-style-type: none"><li>1. Mengetahui tentang <i>Exception</i> di Java.</li><li>2. Mengetahui kelas-kelas <i>exception</i> di Java</li><li>3. Mengetahui bagaimana <i>Exception</i> muncul dan cara menangkap <i>Exception</i></li><li>4. Mengetahui cara melempar <i>Exception</i></li><li>5. Mengetahui tentang <i>User Define Exception</i></li></ol>

### A. Pendahuluan

Program yang handal dapat menghindari keadaan tertentu tanpa mengalami kerusakan. Untuk mencapai program yang handal adalah dengan mengantisipasi permasalahan yang mungkin muncul dan melakukan testing untuk semua kemungkinan permasalahan yang akan muncul. Salah satu langkah dalam membuat program handal adalah dengan meyakinkan bahwa indeks yang digunakan berada didalam *range* yang telah ditentukan. Tetapi terdapat permasalahan dimana cara tersebut sulit dan kadang tidak memungkinkan untuk mengantisipasi semua hal tersebut, Tidak selalu jelas apa yang harus dilakukan ketika sebuah *error* terdeteksi.



Gambar VI-1 Kesalahan pengkodean

Di dalam Java terdapat sebuah exception dimana exception mengindikasikan sebagai event yang *exceptional* (tidak biasa). Event ini merupakan event abnormal



yang muncul selama eksekusi program dan mengganggu alur instruksi. *Event* abnormal dapat berupa *error* di dalam program.

Di Java terdapat 2 tipe *error*, yaitu *compile time error* dan *runtime error*. *Compile time error* muncul ketika Sintaks pemrograman yang dibuat tidak sesuai dengan ketentuan yang ada dalam pemrograman Java (*error* Sintaks). *Runtime error* muncul selama eksekusi program.

## **B. Kelas *Exception***

Kata "*exception*" lebih umum dibandingkan dengan "*error*". Di dalam *Exception* termasuk semua keadaan yang muncul di dalam program yang sedang dieksekusi. Sebuah *exception* dapat berupa *error*, atau hanya kasus khusus.

Ketika sebuah *exception* muncul selama eksekusi program, dikatakan bahwa *exception* telah dilempar (*exception is thrown*). Ketika hal ini terjadi program diberhentikan dan program bisa rusak. Bagaimanapun, kegagalan program dapat dihindari jika *exception* yang dilempar dapat ditangkap dan dikelola sebagaimana mestinya. Sebuah *exception* dapat dilempar di suatu bagian program dan dapat ditangkap di bagian lain dari program tersebut. *Exception* yang tidak ditangkap akan membuat program gagal. Contoh : Jika terjadi pembagian dengan nol atau membuka file yang tidak ada, maka *exception* akan muncul.

Ketika sebuah *exception* dilempar, hal yang sebenarnya dilempar adalah sebuah obyek. Obyek ini membawa informasi dari tempat dimana *exception* di tangkap dan dikelola. Biasanya sebuah obyek *exception* mengandung pesan *error* yang mendeskripsikan apa yang terjadi sehingga menyebabkan *exception* dan dapat berisi data lain.

Semua obyek *exception* merupakan *subclass* dari kelas standar `java.lang.Throwable`, dan *subclass* ini diatur dalam hirararki kelas yang kompleks

Class `Throwable` mempunyai turunan Class yaitu class *Error* dan *Exception*. Kedua kelas ini nanti yang akan mempunyai beberapa turunan. *Subclass* yang ada di bawah *Error* merupakan kelas yang merepresentasikan *error* serius didalam *Java Virtual Machine* yang membuat program di terminasi karena terdapat masalah. Umumnya, sebaiknya tidak dilakukan *try catch* pada *error*. *Subclass Exception* merepresentasikan *exception* yang biasanya disebut *error* tetapi *error* ini merupakan *error* yang dapat diatasi dengan cara tertentu.

Kelas *Exception* mempunyai beberapa *subclass* seperti `RuntimeException`, `InterruptedException` dan `IOException`. *Exception RuntimeException* mengelola *exception* yang muncul di dalam program ketika *runtime*. Untuk lebih



lengkap, dapat dibaca di javadoc manual. Berikut beberapa penyebab exception muncul :

Tabel VI-1 Contoh *Exception*

Exception	Penyebab
<code>IllegalArgumentException</code>	Muncul ketika memberikan argumen yang tidak sesuai dengan tipe data pada suatu <i>method</i>
<code>ArrayIndexOutOfBoundsException</code>	Muncul ketika mengakses elemen <i>array</i> yang tidak terdapat pada <i>array</i> tersebut (melampaui batas indeks)
<code>NumberFormatException</code>	Muncul ketika ingin mengubah sebuah <i>string</i> ke format numerik
<code>ArithmeticException</code>	Muncul ketika membuat sebuah <i>error</i> dalam aritmatika seperti pembagian dengan nol
<code>NullPointerException</code>	Muncul ketika suatu aplikasi berusaha menggunakan obyek tanpa mengalokasikan memori

### C. Mengimplementasikan *Exception Handling*

Ketika sebuah *error* muncul di dalam *method*, Java membuat sebuah obyek *Exception*. Setelah membuat obyek *exception*, Java mengirimkannya ke program dengan melemparkan *exception* (*throwing exception*). *Exception* yang dilempar perlu ditangani menggunakan *exception handler* dan diproses. Untuk mengimplementasikan *Exception handling* digunakan *keyword* berikut :

- o *Try*
- o *Catch*
- o *Finally*
- o *Throw*
- o *Throws*

#### 1. Statement *Try* dan *Catch*

Untuk menangkap *exception* di dalam program java, diperlukan *Statement try*. Blok *Statement try* membingkai kode yang kemungkinan akan memunculkan *exception* dan mendefinisikan *exception handler* yang dapat menanganinya. Blok *catch* digunakan sebagai *exception handler*. Blok *try* harus mempunyai minimal satu buah blok *catch* yang mengikutinya. Klausula *catch* menspesifikasikan tipe *exception* yang akan ditangani. Lingkup dari blok *catch* dibatasi oleh *Statement* yang ada di dalam blok *try*. *Statement* untuk mendeklarasikan blok *try catch*:

```
try {  
    //Statements that cause an exception
```



```
}  
catch(<exception name> <object name>) {  
    //error handling code  
}
```

Komputer akan mengeksekusi *statement* yang ada di dalam blok *try*. Jika tidak ada *exception* yang muncul ketika dieksekusi maka *Statement catch* tidak akan di jalankan. Jika sebuah *exception* muncul maka blok *catch* akan langsung dijalankan. Contoh :

```
public class ArithmeticException {  
    public static void main (String[] args) {  
        int num1=5, num2=0;  
        int num3=num1/num2;  
        Sistem.out.println("The num3 = "+num3);  
    }  
}
```

Jika program diatas di *compile* maka tidak ada *error*, tetapi setelah di eksekusi, *handler* pada Java *runtime* sistem akan melempar sebuah *exception* ketika terjadi pembagian dengan nol sehingga akan dibuat sebuah obyek dari kelas *exception* yang dilempar yaitu *ArithmeticException*.

Jika sebuah *exception* muncul didalam blok *try* maka *exception* handler yang berkaitan akan menangani *exception* tersebut, pada contoh diatas karena muncul *exception* *ArithmeticException* maka akan dibuat *exception* handler untuk *ArithmeticException*

```
public class ArithmeticException {  
    public static void main (String[] args)  
    {  
        int num1=5, num2=0;  
        try {  
            int num3=num1/num2;  
            System.out.println("The num3 = "+num3);  
        }  
        catch(ArithmeticException obj) {  
            System.out.println("Division by zero");  
        }  
    }  
}
```

Dengan menangani *exception* maka program terhindar dari kerusakan. Sebelum isi dari *catch* dieksekusi, obyek yang merepresentasikan *exception* diletakkan ke dalam variabel *obj*, yang berisi data tentang *exception*.

## 2. Multiple Catch

Sebuah blok *try* dapat diikuti oleh banyak blok *catch*. Hal ini penting ketika sebuah blok *try* mempunyai *Statement* yang mungkin akan memunculkan berbagai tipe *exception*. Contoh :



```
public class MultipleCatch {  
    public static void main (String[] args) {  
        int num1=5, num2=0;  
        int arrNum[]= {1,2,3 };  
        try {  
            int num3=num1/num2;  
            System.out.println("The num3 = "+num3);  
            System.out.println("The 3 elemen is "+ArrNum[3]);  
        }  
        catch(ArithmeticException obj){  
            System.out.println("Division by zero");  
        }  
        catch(ArrayIndexOutOfBoundsException obj) {  
            System.out.println("Error...Out Of Bounds");  
        }  
        catch(Exception obj) {  
            System.out.println("Other error");  
        }  
    }  
}
```

Pada kode diatas terdapat tiga *catch* untuk menangani exception yaitu *ArithmeticException*, *ArrayIndexOutOfBoundsException*, dan *Exception*. Blok *catch* yang pertama akan dieksekusi jika didalam blok *try* melempar *ArithmeticException*, blok *catch* yang kedua akan dieksekusi jika di dalam blok *try* melempar *ArrayIndexOutOfBoundsException*, dan blok *catch* yang terakhir akan menangkap *exception* yang lain. Ketika program mengeksekusi `int num3=num1/num2;`, maka *ArithmeticException* akan muncul dan akan ditangkap oleh blok *catch* yang pertama (*ArithmeticException obj*).

Kelas *Exception* merupakan *superclass* dari semua tipe kelas *exception*. Jika blok *catch* yang pertama berisi obyek dari kelas *Exception* maka blok *catch* berikutnya tidak akan dieksekusi, hal ini disebut *unreachable code*. Untuk menghindari *error unreachable code*, maka *catch* yang didalamnya mengandung kelas *Exception* diletakkan pada urutan blok *catch* yang terakhir.

### 3. **Statement Finally**

*Statement* yang berada di dalam blok *finally* akan dieksekusi paling akhir ketika sedang mengeksekusi blok *try*, meskipun tidak ada *exception* yang muncul atau ada *exception* yang muncul. *Statement finally* digunakan untuk melakukan sesuatu yang penting ketika terdapat *exception* atau tidak. Sintaks *try-catch-finally*:

```
try {  
    //block of code  
}  
catch (<exception name> <object name>) {
```





```
        //block of exception code
    }
    Finally {
        //block of code that is always executed
    }
```

Jika sebuah *exception* dilempar atau tidak dilempar, blok *finally* akan dieksekusi meskipun tidak ada *Statement catch* yang cocok dengan *exception* yang muncul. Untuk menghindari duplikasi kode maka dapat diletakkan di dalam blok *finally*. Blok *finally* merupakan pilihan, boleh ada atau tidak di dalam *try catch blok*, tetapi hanya satu blok *finally* saja yang dapat dibuat dalam satu buah blok *try*.

#### 4. **Throwing Exception**

Suatu saat dibutuhkan untuk melempar *exception* pada program yang dibuat. Hal ini terjadi jika di dalam program ditemukan beberapa *exceptional* atau kondisi *error* tetapi tidak ada alasan untuk menangani *error* tersebut dimana permasalahan ditemukan. Program dapat melempar sebuah *exception* dengan harapan dapat ditangkap dan ditangani oleh blok *catch*. Hal ini dapat dilakukan dengan melempar *exception* secara eksplisit dengan menggunakan *Statement throw*. Contoh: dibutuhkan untuk melempar *exception* ketika user salah memasukkan nilai.

*Statement throw* akan menyebabkan terminasi dari aliran control program Java dan menghentikan eksekusi ketika *Statement* tersebut dieksekusi. *Exception* yang dilempar oleh *Statement throw* akan ditangkap oleh blok *catch* yang ada, jika tidak ada blok *catch* yang sesuai dengan *exception* yang dilempar maka program akan diterminasi. Contoh penggunaan *Statement throw*:

```
public class ThrowingStatement {
    static double root(double A,double B, double C)
    {
        double disc=0;
        try {
            if(A==0) {
                throw new IllegalArgumentException("A can't be zero");
            }
            else {
                disc = (B*B)-(4*A*C);
                if (disc<0)
                    throw new IllegalArgumentException("Discriminant< 0");
            }
        } //endtry
        catch(IllegalArgumentException obj) {
            Sistem.out.println("Illegal Argument in : "+obj);
        }
    }
}
```



```
    }  
    return ((-B + Math.sqrt(disc))/(2*A));  
}  
public static void main(String[] args)    {  
    double Ans = root(0,2,4);  
}  
}
```

Program diatas mempunyai kondisi  $A \neq 0$  dan  $B^2 - 4AC \geq 0$ . *Method* akan melempar sebuah exception `IllegalArgumentException` jika kondisi tersebut tidak terpenuhi. Ketika sebuah kondisi illegal ditemukan dalam *method*, melempar sebuah *exception* merupakan cara yang masuk akal. Jika program yang memanggil *method* mengetahui cara menangani *error*, maka dapat menangkap *exception*, jika tidak maka program akan rusak dan harus diperbaiki.

*Statement Throws* digunakan oleh *method* untuk menspesifikasikan tipe-tipe exception yang akan dilempar di dalam *method*. Jika *method* yang melempar *exception* tidak dapat menanganinya maka *exception* harus ditangani oleh bagian yang memanggil *method* tersebut. Hal ini dilakukan dengan *Statement throws*. *Statement throws* digunakan untuk menuliskan tipe-tipe *exception* yang terdapat di dalam *method*. Contoh :

```
class ThrowingStatement {  
    static double root(double A, double B, double C) throws  
        IllegalArgumentException {  
        double disc=0;  
        if(A==0) {  
            throw new IllegalArgumentException("A can't be zero");  
        }  
        else {  
            disc = (B*B)-(4*A*C);  
            if (disc<0)  
                throw new IllegalArgumentException("Discriminant< 0");  
        }  
        return ((-B + Math.sqrt(disc))/(2*A));  
    }  
    public static void main (String[] args)    {  
        double Ans;  
        try {  
            Ans = root(0,2,4);  
        }  
        catch(IllegalArgumentException obj)    {  
            System.out.println("Illegal Argument in : "+obj);  
        }  
    }  
}
```



#### D. User defined Exception

*User defined exception* merupakan exception yang dibuat oleh programmer sesuai dengan permintaan pada aplikasi. Tiap aplikasi yang dibuat kemungkinan mempunyai batasan yang spesifik. *Statement throw, throws, try, catch, dan finally* digunakan dalam mengimplementasikan *user define exception*. Kelas *Exception* menuruni semua *method* yang ada di dalam kelas *Throwable*. *User defined exception* juga menuruni semua *method* yang didefinisikan di dalam kelas *Throwable*. Contoh :

```
class MyException extends Exception {
    public String toString() {
        return "MyException caught : The Input value cannot less than 0";
    }
}
public class UserException {
    static int TestUAQ, TestUTQ;
    static void setValue(int UAQ,int UTQ) throws MyException {
        if ((UAQ<0)|| (UTQ<0))
            throw new MyException();
        TestUAQ=UAQ;
        TestUTQ=UTQ;
        Sistem.out.println("The value UAQ : "+TestUAQ+", UTQ : "+TestUTQ);
    }
    public static void main (String[] args) {
        try {
            setValue(80,90);
            setValue(-70,60);
        }
        catch(MyException obj) {
            System.out.println("The Exception raised : "+obj);
        }
    }
}
```

## Rangkuman

1. *Exception* merupakan event abnormal yang muncul selama eksekusi program dan mengganggu alur instruksi.
2. Di Java terdapat 2 tipe *error*, yaitu *compile time error* dan *runtime error*.
3. Ketika sebuah *exception* dilempar, hal yang sebenarnya dilempar adalah sebuah obyek yang membawa informasi dari tempat dimana exception di tangkap dan dikelola.
4. Class *Throwable* mempunyai turunan *Class* yaitu class *Error* dan *Exception*.
5. *Exception Handling* diimplementasikan dengan menggunakan *keyword Try, Catch, Finally, Throw* dan *Throws*.



6. Blok *Statement* **try** membingkai kode yang kemungkinan akan memunculkan *exception* dan mendefinisikan *exception handler* yang dapat menanganinya sedangkan blok **catch** digunakan sebagai *exception handler*.
7. Blok *finally* akan dieksekusi paling akhir ketika sedang mengeksekusi blok *try*, ketika ada *exception* atau tidak.
8. Satu blok *try* mempunyai satu atau lebih blok *catch* dan satu blok *finally* yang optional.
9. *Statement* **throw** digunakan untuk melempar *Exception* secara eksplisit.
10. *User defined exception* merupakan *exception* yang dibuat oleh programmer sesuai dengan permintaan pada aplikasi.

## Latihan

1. Apakah hasil dari Sintaks di bawah

```
public class MultipleCatch {
    public static void main (String[] args) {
        int num1=5, num2=0;
        int arrNum[] = {1,2,3 };
        try {
            int num3=num1/ArrNum[3];
            int num4=num1/num2;
            System.out.println("The num3 = "+num3);
            System.out.println("The 3 elemen is "+ArrNum[3]);
        }
        catch(ArithmeticException obj) {
            System.out.println("Division by zero");
        }
        catch(ArrayIndexOutOfBoundsException obj) {
            System.out.println("Error...Out Of Bounds");
        }
        catch(Exception obj) {
            System.out.println("Other error");
        }
    }
}
```

2. Sebutkan hasil yang terjadi dari program di bawah

```
public class MyApp {
    String name="Mark";
    int nilai[]={100,0};
    void getNilai() {
        Sistem.out.println("Nilai rata-rata : "+((nilai[1]+nilai[2])/2));
    }
    public static void main (String[] args) {
        try {
            MyApp App=new MyApp();
            Sistem.out.println(App.name);
            App.getNilai();
            MyApp App2;
        }
    }
}
```



```
    }  
    catch(ArithmeticException obj) {  
        Sistem.out.println("Error, ada pembagian dengan 0 ");  
    }  
    catch(ArrayIndexOutOfBoundsException obj) {  
        Sistem.out.println("Error, pengakses array diluar batas");  
    }  
    finally {  
        Sistem.out.println("Finally here");  
    }  
}
```

3. Terdapat sebuah sistem supermarket yang akan digunakan untuk mengolah data pembelian. Pada aplikasi tersebut disimpan data barang yang dijual, meliputi kode barang, nama barang, pabrik, dan harga. Harga barang harus lebih besar 0. Ketika terjadi pembelian, jumlah pembelian barang harus lebih besar dari 0. Buatlah user defined *exception* yang akan mengeluarkan *exception* jika harga barang yang dimasukkan  $\leq 0$  dan jika jumlah barang yang di beli  $< 0$ .



## BAB VII. *Input/Output*

Ulasan	Tujuan
<b>Dalam bab ini akan dibahas tentang bagaimana cara melakukan pengaksesan data kedalam file, baik melakukan proses pembacaan ataupun penulisan. Akan dibahas tentang pengaksesan data menggunakan <i>stream</i> dan <i>random-access file</i></b>	<ol style="list-style-type: none"><li>1. Mengetahui pembacaan dan penulisan kedalam sebuah file.</li><li>2. Mengetahui penggunaan file menggunakan kelas File.</li><li>3. Mengetahui penggunaan <i>byte stream</i>.</li><li>4. Mengetahui penggunaan <i>character stream</i>.</li><li>5. Mengetahui penggunaan <i>random-access file</i></li></ol>

### A. Pendahuluan

Sebuah aplikasi menyimpan data dalam sebuah variabel, yang sebenarnya data tersebut disimpan dalam sebuah alamat memori. Data dalam variabel tersebut akan hilang begitu keluar dari cakupan program (jika berupa variabel lokal) atau ketika program ditutup. Bagi sebuah aplikasi yang membutuhkan sekian banyak data dalam pemrosesannya, maka akan membutuhkan waktu *runtime* yang sangat besar karena harus memasukkan data lagi dari awal. Untuk menghindari hal itu, diperlukan sebuah mekanisme yang bisa digunakan untuk menyimpan data-data dari variabel yang sebelumnya telah kita masukkan dan pergunakan. Dengan mekanisme tersebut, maka data tidak akan hilang jika aplikasi ditutup dan bisa dipergunakan lagi begitu program dijalankan lagi.

Komputer akan menyimpan data yang ada dalam sebuah file kedalam tempat penyimpanan sekunder seperti *harddisk*, *optical disk* atau pita magnetik. Dalam bab ini akan dibahas tentang bagaimana Java membuat, meng-*update* dan memproses data pada sebuah file.

Pemrosesan file merupakan salah satu kemampuan penting dalam sebuah bahasa pemrograman yang digunakan dalam pengembangan sebuah aplikasi, yang biasanya digunakan untuk menyimpan data secara terus menerus. Dalam bab ini akan dibahas tentang fitur pengaksesan data pada file dan *stream* input/output. Pemrosesan file merupakan salah satu subset dari kemampuan pemrosesan *stream* dalam Java, yang memungkinkan sebuah program untuk



membaca dan menulis data kedalam memori, file atau melalui jaringan. Disini akan kita bahas tentang tiga bentuk pemrosesan file, pemrosesan file dengan *stream* dan pemrosesan *random-access file*.

## B. Implementasi Kelas File

File merupakan tempat penyimpanan utama data yang dimiliki oleh sebuah program. Java menyediakan sebuah package `java.io` untuk melakukan pemrosesan input dan output. Salah satu kelas yang ada dalam *package* `java.io` adalah kelas `File`, yang digunakan untuk melakukan pembuatan file dan direktori.

Kelas `file` dalam package `java.io` menyediakan berbagai macam *method* untuk mengakses atribut-atribut dari file dan direktori, seperti hak akses dari, tanggal pembuatan dan akses, alamat dari direktori dimana file tersebut berada, dan lain-lain. Sebuah obyek yang dibuat dari kelas `File` yang akan merepresentasikan file atau direktori. Terdapat beberapa konstruktor yang bisa kita gunakan untuk membuat *instance* pada kelas `File`:

- `File(File obyekDirektori, String namaFile):`  
Membuat sebuah *instance* pada kelas `File` dimana `obyekDirektori` adalah sebuah obyek dari kelas `File` yang menunjuk sebuah direktori, sedang `namaFile` adalah file yang akan diakses.
- `File (String alamatDirektori):`  
Membuat sebuah obyek dari kelas `File`. `alamatDirektori` merupakan alamat di direktori mana file yang akan diakses berada.
- `File(String alamatDirektori, String namaFile):`  
Membuat sebuah obyek dari kelas `File`. `alamatDirektori` menentukan alamat direktori dari file sedangkan `nama file` menentukan file mana yang akan diakses.

Keterangan : Alamat disini bisa berupa alamat absolut atau alamat relatif.

Terdapat beberapa *method* yang bisa digunakan didalam sebuah obyek dari kelas `File`, antara lain:

Tabel 7-1 Method dalam kelas `File`

Method	Keterangan
<code>boolean canRead()</code>	Akan mengembalikan nilai <i>true</i> jika file yang sedang diakses saat ini dapat dibaca ( <i>readable</i> ). Jika tidak akan mengembalikan nilai <i>false</i> .
<code>boolean canWrite()</code>	Akan mengembalikan nilai <i>true</i> jika file bisa ditulis ( <i>writable</i> ). Jika tidak akan mengembalikan nilai <i>false</i>



Method	Keterangan
<code>boolean exists()</code>	Akan mengembalikan nilai <i>true</i> jika argumen yang ditunjuk pada konstruktor instantiasi obyek merupakan sebuah file atau direktori. Jika bukan file atau direktori akan mengembalikan nilai <i>false</i> .
<code>boolean isFile()</code>	Akan mengembalikan nilai <i>true</i> jika argumen yang ditulis dalam konstruktor merupakan sebuah file. Selain itu akan bernilai <i>false</i> .
<code>boolean isDirectory()</code>	Akan mengembalikan nilai <i>true</i> jika argumen yang ditulis dalam konstruktor adalah sebuah direktori. Selain itu akan mengembalikan nilai <i>false</i> .
<code>boolean isAbsolute()</code>	Akan mengembalikan nilai <i>true</i> jika argument yang ditulis dalam konstruktor adalah sebuah alamat absolut dari file atau direktori. Selain itu akan mengembalikan nilai <i>false</i> .
<code>String getAbsolutePath()</code>	Akan mengembalikan sebuah string alamat absolut dari sebuah file atau direktori.
<code>String getName()</code>	Akan mengembalikan sebuah string nama dari file atau direktori.
<code>String getPath()</code>	Akan mengembalikan sebuah string alamat dari dimana file atau direktori tersebut.
<code>String getParent()</code>	Akan mengembalikan sebuah string alamat induk dari file atau direktori tersebut. (direktori dimana file atau direktori tersebut disimpan)
<code>long length()</code>	Mengembalikan panjang dari file, dalam ukuran byte. Jika berupa sebuah direktori akan mengembalikan nilai 0 (nol).
<code>long lastModified()</code>	Akan mengembalikan sebuah representasi waktu kapan terakhir kali file atau direktori tersebut dimodifikasi.
<code>String[] list()</code>	Akan mengembalikan sebuah <i>array</i> berbentuk string yang akan menyimpan daftar isi (file dan direktori) dari sebuah direktori. Jika bukan sebuah direktori akan mengembalikan nilai <i>null</i> .

Berikut adalah contoh penggunaan kelas File.

```
import java.io.File;
import java.io.IOException;
class AksesFile {
    String daftar[] = new String[20];
    public void displayData() {
        try {
            File obj1 = new File("D:\\IO", "KelasFile.java");
            File obj2 = new File("D:\\IO");
            System.out.println("Sebuah obyek file");
            System.out.println("Nama File      : "+obj1.getName());
            System.out.println("getAbsolutePath : "+obj1.getAbsolutePath());
```





```
System.out.println("getPath           : "+obj1.getPath());
System.out.println("Lokasi           : "+obj1.getParent());
System.out.println("Panjang           : "+obj1.length());
System.out.println("Last Modif        : "+obj1.lastModified());
System.out.println();
System.out.println("Sebuah obyek Direktori");
System.out.println("Nama Direktori : "+obj2.getName());
System.out.println("getAbsolutePath : "+obj2.getAbsolutePath());
System.out.println("getPath         : "+obj2.getPath());
System.out.println("Lokasi         : "+obj2.getParent());
System.out.println("Panjang         : "+obj2.length());
System.out.println("last Modif      : "+obj2.lastModified());
daftar = obj2.list();
System.out.println("Isi dari direktori : ");
for (int i=0;i<daftar.length;i++)
{
    System.out.println("\t"+daftar[i]);
}
}
catch (Exception e){
    System.out.println(e);
}
}
}

class KelasFile {
    public static void main(String[] args) {
        AksesFile aksesFile = new AksesFile();
        aksesFile.displayData();
    }
}
```

Keluaran dari program diatas adalah:

```
D:\IO>java KelasFile
Sebuah obyek file
Nama File       : KelasFile.java
getAbsolutePath : D:\IO\KelasFile.java
getPath         : D:\IO\KelasFile.java
Lokasi          : D:\IO
Panjang         : 1479
Last Modif      : 1237535716687

Sebuah obyek Direktori
Nama Direktori  : IO
getAbsolutePath : D:\IO
getPath         : D:\IO
Lokasi          : D:\
Panjang         : 0
last Modif      : 1237534976734
Isi dari direktori :
    AksesFile.class
    file.class
    KelasFile.class
    KelasFile.java
    KelasFile.java.bak
```



### C. Penggunaan *Stream* dalam Java

*Stream* dalam sebuah program java digunakan untuk melakukan pembacaan atau penulisan kedalam sebuah asal (*source*) atau tujuan (*destination*). Operasi *Input output* (I/O) terdiri dari dua bagian, yaitu *input* yang berasal dari sebuah sumber (misal sebuah keyboard, jaringan komputer, file) kemudian data masukan tadi akan diproses dan akan dikeluarkan menuju sebuah tujuan (misal monitor, jaringan komputer, file).

Didalam Java, package `java.io` terdiri dari berbagai macam kelas dan *interface stream* I/O yang mendukung operasi I/O. *Stream* dibagi menjadi dua jenis, *byte stream* dengan unit dasar satu *byte* data dan *character stream* dengan dasar satu karakter *Unicode*.

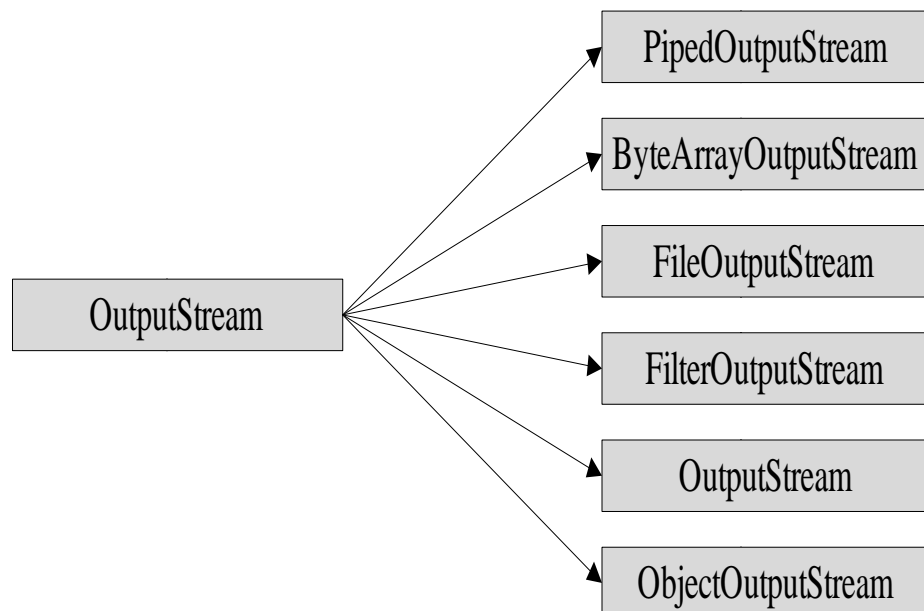
#### 1. Implementasi *Byte Stream*

*Byte stream* menggunakan *byte* sebagai unit dasar operasi baca dan tulis terhadap *stream*nya. Terdapat dua Kelas *byte stream* yaitu `InputStream` dan `OutputStream`.

##### a. *Output Stream*

Kelas `OutputStream` merupakan *stream* yang digunakan untuk menulis data dalam bentuk *byte*. Beberapa contohnya adalah digunakan untuk menulis kedalam layar monitor atau file.

Berikut adalah hirarki dari kelas `OutputStream`.



Gambar 7-1 Kelas `OutputStream`



Beberapa *method* yang digunakan dalam kelas `OutputStream`

Tabel 7-2 Method pada kelas `OutputStream`

Method	Keterangan
<code>write(byte b)</code>	Menulis sekian <code>b</code> byte kedalam sebuah file.
<code>write(byte b[])</code>	Menulis <i>array</i> berisi byte kedalam sebuah file.
<code>write(byte b[], int offset, int length)</code>	Menulis kedalam byte dimana datanya berasal dari sebuah <i>array</i> <code>b</code> , dimulai dari lokasi <code>offset</code> sebanyak sekian <code>length</code> byte.
<code>close()</code>	Menutup <i>stream</i> output.
<code>flush()</code>	Menghapus buffer <i>stream</i> output.

### 1) Obyek `System.out`

Kelas `System` dalam package `java.lang` memiliki sebuah variabel statik yang digunakan untuk merepresentasikan sebuah layar monitor. Variabel tersebut adalah `out`, yang merupakan *instance* dari kelas `PrintStream` yang bisa menggunakan *method* `print()` dan `println()` untuk menulis kedalam layar monitor.

Berikut adalah contoh penggunaan `System.out`.

```
public class CobaSistemIn {  
    public static void main(String args[]) {  
        byte buffer[]=new byte[80];  
        System.out.print("Ketik sesuatu, ");  
        System.out.print("akhiri dengan ");  
        System.out.println("menekan tombol enter");  
        try {  
            Sistem.in.read(buffer);  
        }  
        catch(Exception e) {  
            System.out.println(e);  
        }  
        System.out.println();  
        String str=new String(buffer);  
        System.out.println("isi Teks:");  
        System.out.print(str);  
    }  
}
```

Berikut adalah hasil dari kode diatas.



```

C:\WINDOWS\system32\cmd.exe
D:\IO>java CobaSystemIn
Ketik sesuatu, akhiri dengan menekan tombol enter
contoh System.out

isi Teks:
contoh System.out

D:\IO>_
```

## 2) File *OutputStream*

Kelas *FileOutputStream* digunakan untuk melakukan operasi penulisan kedalam sebuah file. Beberapa konstruktor yang bisa digunakan dalam kelas *FileOutputStream* adalah :

- *FileOutputStream*(*File* objFile):  
Membuat sebuah obyek *File stream* yang menghubungkan dengan sebuah file. objFile merupakan sebuah file obyek yang merepresentasikan sebuah file.
  - *FileOutputStream*(*String* alamatFile):  
Membuat sebuah obyek *File stream* yang menghubungkan dengan sebuah file. *String* alamatFile merupakan alamat dari file yang akan dijadikan tujuan penulisan.
  - *FileOutputStream*(*File* objFile, *boolean* bool):  
Membuat sebuah obyek *File stream* yang menghubungkan dengan sebuah file. objFile merupakan sebuah file obyek yang merepresentasikan sebuah file, sedang *boolean* bool jika bernilai *true* menunjukkan bahwa penulisan file dilakukan dalam mode *append* (penambahan data di akhir file)
  - *FileOutputStream*(*String* s, *boolean* bool):  
Membuat sebuah obyek *File stream* yang menghubungkan dengan sebuah file. *String* alamatFile merupakan alamat dari file yang akan dijadikan tujuan penulisan, sedang *boolean* bool jika bernilai *true* menunjukkan bahwa penulisan file dilakukan dalam mode *append* (penambahan data di akhir file)
- Beberapa *method* untuk melakukan penulisan kedalam *stream* dengan kelas *FileOutputStream*



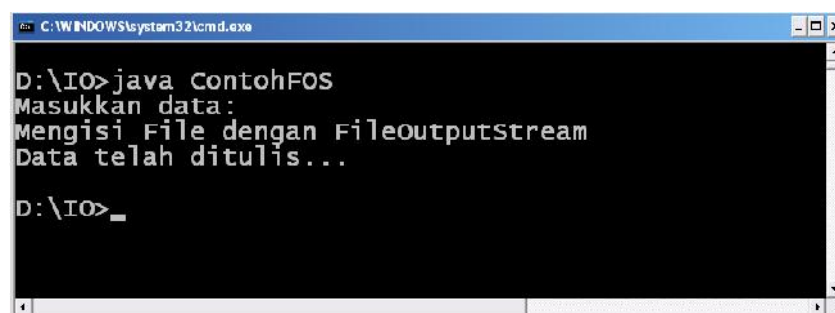
Tabel 7-3 Method dalam kelas FileOutputStream

Method	Keterangan
write(byte b)	Menulis byte b kedalam file <i>stream</i> .
write(byte b[], int offset, int length)	Menulis sejumlah length byte dari posisi offset kedalam file <i>stream</i> , sebanyak length byte.

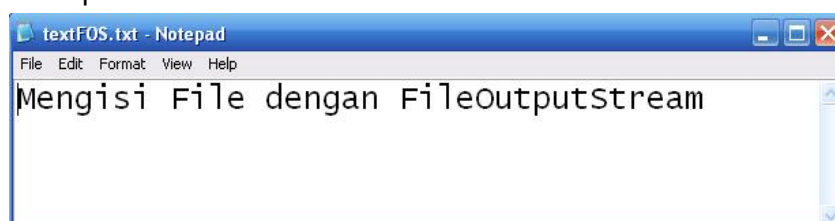
Berikut adalah contoh dari penggunaan FileOutputStream.

```
import java.io.*;
public class ContohFOS {
    public static void main(String args[]) {
        byte buffer[]=new byte[100];
        try {
            System.out.println("Masukkan data:");
            System.in.read(buffer, 0, 100);
        }
        catch(IOException e) {
            System.out.println(e);
        }
        try {
            FileOutputStream objFOS;
            objFOS= new FileOutputStream("textFOS.txt");
            objFOS.write(buffer);
            System.out.println("Data telah ditulis...");
        }
        catch(FileNotFoundException f) {
            System.out.println(f);
        }
        catch(IOException i) {
            System.out.println(i);
        }
    }
}
```

Runtime dari kode diatas



Hasil penulisan dalam file textFOS.txt





### 3) **BufferedOutputStream**

Kelas `BufferedOutputStream` membuat sebuah buffer dalam memori dan menampilkan buffer tersebut kedalam *stream* output. Buffer akan menyimpan byte-byte dalam dan mengirimkannya ke tujuan hanya dalam 1 kali operasi I/O saja. Berikut beberapa konstruktor yang digunakan dalam kelas `BufferedOutputStream`.

- `BufferedOutputStream(OutputStream objOS)` :  
Membuat sebuah *stream* output dan menambahkan buffer kedalamnya. Ukuran buffer secara defaultnya adalah 512 byte.
- `BufferedOutputStream(OutputStream os, int besarBuffer)` :  
Membuat sebuah *stream* output dan menambahkan buffer kedalamnya dengan ukuran buffer yang telah ditentukan sebesar `besarBuffer`.

Tabel 7-4 Method dalam kelas `BufferedOutputStream`

Method	Keterangan
<code>public void write(byte[] b)</code>	Menulis sejumlah <code>b.length</code> byte kedalam <i>stream</i> output.
<code>public void write(byte[] b, int offset, int len)</code>	Menulis isi dari <i>array</i> <code>b</code> kedalam <i>stream</i> output sejumlah <code>len</code> dimulai dari posisi <code>offset</code>
<code>public void flush()</code>	Menulis semua byte yang ada didalam buffer kedalam tujuan penyimpanan, misal file

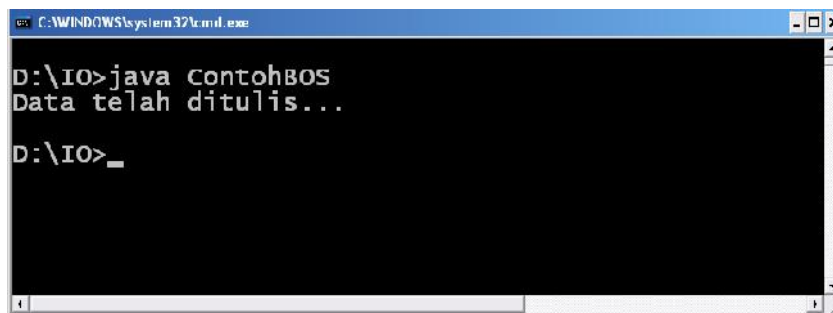
Berikut adalah contoh penggunaan `BufferedOutputStream`.

```
import java.io.BufferedOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
public class ContohBOS {
    public void writeToFile(String filename) {
        BufferedOutputStream BOS = null;
        try {
            BOS = new BufferedOutputStream(new
FileOutputStream(filename));
            BOS.write("Contoh BufferedOutputStream".getBytes());
            System.out.println("Data telah ditulis...");
        }
        catch (FileNotFoundException e) {
            System.out.println(e);
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```



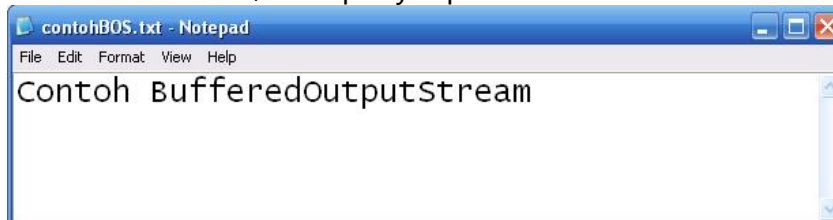
```
    }  
    finally {  
        try {  
            if (BOS != null) {  
                BOS.flush();  
                BOS.close();  
            }  
        }  
        catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
}  
public static void main(String[] args) {  
    new ContohBOS().writeToFile("contohBOS.txt");  
}  
}
```

Eksekusi kode diatas



```
C:\WINDOWS\system32\cmd.exe  
D:\IO>java contohBOS  
Data telah ditulis...  
D:\IO>
```

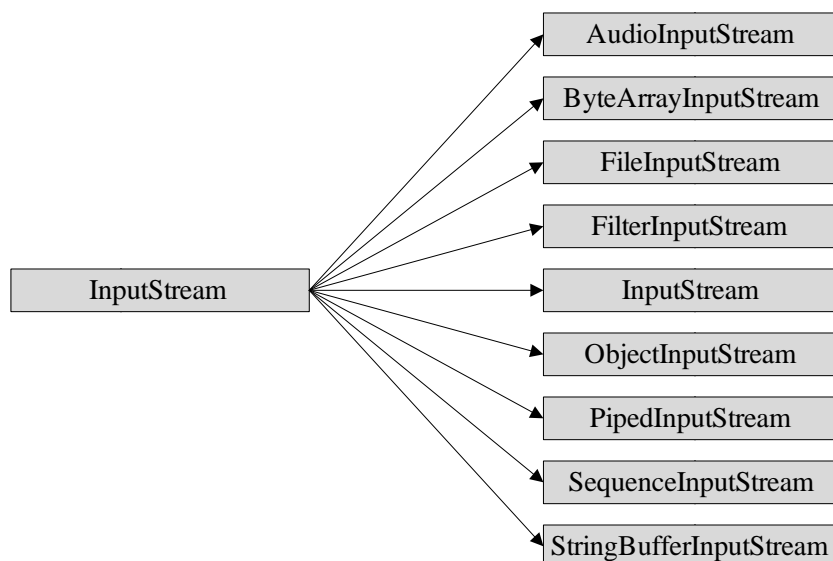
File contohBOS.txt, hasil penyimpanan dari contoh diatas.



```
contohBOS.txt - Notepad  
File Edit Format View Help  
Contoh BufferedOutputStream
```



## b. Input Stream



Gambar 7-2 Hirarki kelas InputStream

Input *stream* adalah *stream* yang membaca data dalam bentuk byte. Java menyediakan kelas `InputStream` yang merupakan kelas abstrak untuk melakukan proses input untuk membaca data dari berbagai macam sumber (seperti file, keyboard dan jaringan) dan menampilkan datanya kedalam layar monitor.

Beberapa *method* yang digunakan dalam kelas `InputStream`

Tabel 7-5 Method `InputStream`

Method	Keterangan
<code>int read()</code>	Membaca satu byte data dari input <i>stream</i> .
<code>int read(byte b[])</code>	Membaca beberapa byte dari input <i>stream</i> dan memasukkannya kedalam <i>array</i> b
<code>int read(byte b[], int offset, int length)</code>	Membaca sebanyak length byte dari <i>stream</i> dan memasukkannya kedalam <i>array</i> b. Jika memasuki akhir dari <i>stream</i> akan mengembalikan nilai -1
<code>available()</code>	Mengembalikan jumlah byte yang bisa dibaca dari sebuah <i>stream</i> .
<code>long skip(long n)</code>	Melewatkan sekian n byte dari input <i>stream</i> .
<code>mark(int nbyte):</code>	Menandai posisi yang ada pada input <i>stream</i> saat ini.
<code>reset():</code>	Mengembalikan posisi kepada lokasi yang telah ditandai dari <i>method</i> mark.
<code>void close()</code>	Menutup input <i>stream</i> dan melepaskan sumber daya yang terikat pada <i>stream</i> tersebut.





### 1) Obyek `System.in`

Kelas `System` dalam package `java.lang` memiliki satu anggota static yang merupakan referensi dari sebuah keyboard. Variabel dengan nama `in` merupakan sebuah *instance* dari kelas `InputStream` yang dapat digunakan untuk membaca dari keyboard. Berikut adalah contoh penggunaan `System.in` yang akan digunakan untuk membaca inputan user dari keyboard.

```
public class CobaSistemIn {  
    public static void main(String args[]) {  
        byte buffer[]=new byte[80];  
        System.out.print("Ketik sesuatu, ");  
        System.out.print("akhiri dengan ");  
        System.out.println("menekan tombol enter");  
        try {  
            System.in.read(buffer);  
        }  
  
        catch(Exception e) {  
            Sistem.out.println(e);  
        }  
        System.out.println();  
        String str=new String(buffer);  
        System.out.println("isi Teks:");  
        System.out.print(str);  
    }  
}
```

Berikut adalah hasil dari kode diatas:

```
C:\WINDOWS\system32\cmd.exe  
D:\IO>java CobaSystemIn  
Ketik sesuatu, akhiri dengan menekan tombol enter  
belajar OOP dengan java  
isi Teks:  
belajar OOP dengan java  
D:\IO>
```

### 2) Kelas `FileInputStream`

Kelas `FileInputStream` digunakan untuk melakukan operasi input pembacaan data dari dalam file. Berikut adalah beberapa konstruktor yang bisa digunakan dalam kelas `FileInputStream`

- `FileInputStream(File namaFile):`  
Membuat sebuah obyek yang akan menggunakan file dengan nama `namaFile` sebagai sumber inputannya.
- `FileInputStream(String alamatFile):`



Membuat sebuah obyek yang akan menggunakan sebuah file dengan alamat `alamatFile` (berbentuk `string`) sebagai sumber inputannya.

- `FileInputStream(FileDescriptor obyekFile):`  
Membuat sebuah obyek yang akan menggunakan sebuah `obyekfile` yang sudah ditentukan sebelumnya sebagai sumber inputannya.

Beberapa *method* yang digunakan dalam kelas `FileInputStream`

Tabel 7-6 Method dalam kelas `FileInputStream`

Method	Keterangan
<code>read()</code>	Membaca satu byte data dari <i>stream</i> inputan.
<code>read(byte b[], int offset, int length):</code>	Membaca sekian <code>length</code> byte data dari <i>stream</i> input dan memasukkannya kedalam sebuah <i>array</i> <code>b</code> .
<code>long skip(long n)</code>	Melewatkan sekian <code>n</code> byte data dari <i>stream</i> input.

Berikut adalah contoh penggunaan `FileInputStream`.

```
import java.io.*;
public class ContohFIS {
    public static void main(String args[])
    {
        byte buffer[]=new byte[100];
        try {
            FileInputStream file=new FileInputStream("dokumen.txt");
            file.read(buffer,0,50);
        }
        catch(Exception e) {
            System.out.println(e);
        }
        System.out.println("Isi file dokumen.txt adalah:");
        String str=new String(buffer);
        System.out.println(str);
    }
}
```

Berikut adalah hasil dari kode diatas

```
C:\WINDOWS\system32\cmd.exe
D:\IO>java ContohFIS
Isi file dokumen.txt adalah:
ini adalah isi file dokumen

D:\IO>
```



### 3) Kelas `BufferedInputStream`

Kelas `BufferedInputStream` menerima inputan dari *stream* input dan menyimpannya kedalam buffer memori komputer, ukuran buffer secara defaultnya adalah 512 byte. Berbeda dengan kelas `ByteArrayInputStream` yang hanya membaca data dalam bentuk byte saja, kelas `BufferedInputStream` akan membaca data dalam berbagai jenis tipe data. Beberapa konstruktor dari kelas `BufferedInputStream` adalah:

- `BufferedInputStream(InputStream isObj) :`  
Membuat sebuah obyek *stream* input dan menambahkan buffer kedalam obyek tersebut. `isObj` merupakan obyek yang digunakan untuk menentukan dari sumber mana inputan berasal.
- `BufferedInputStream(InputStream isObj, int ukuranBuffer) :`  
Membuat sebuah *stream* input dan menambahkan ukuran buffer sebesar `ukuranBuffer`. `isObj` merupakan obyek yang digunakan untuk menentukan dari sumber mana inputan berasal.

Contoh penggunaan `BufferedReader`.

```
import java.io.*;
public class ContohBSR {
    public static void main(String args[]) {
        InputStreamReader is = new InputStreamReader(Sistem.in);
        BufferedReader dataIn = new BufferedReader(is);
        String data="";
        try {
            System.out.print("Ketik sesuatu, ");
            System.out.println("akhiri dengan enter :");
            data=dataIn.readLine();
        }
        catch(Exception e) {
            Sistem.out.println(e);
        }
        Sistem.out.println();
        Sistem.out.println("Hasil output data:");
        Sistem.out.println(data);
    }
}
```

Hasil dari kode diatas adalah:

```
C:\WINDOWS\system32\cmd.exe
D:\IO>java contohBIS
Ketik sesuatu, akhiri dengan enter :
contoh BufferedInputStream

Hasil output data:
contoh BufferedInputStream

D:\IO>
```

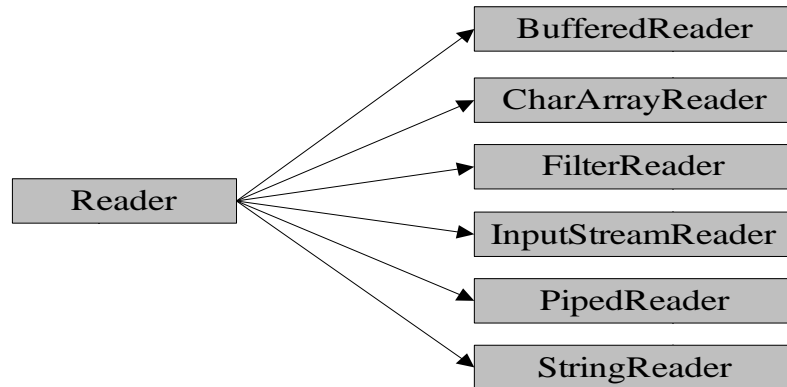


## 2. Implementasi *Character Stream*

Kelas-kelas *stream* karakter menangani proses I/O dengan basis unit adalah sebuah karakter dengan menggunakan representasi *Unicode* yang disimbolkan dengan 16 bit. *Stream* karakter terdiri dari kelas-kelas *Reader* dan *Writer* dalam package *java.io*.

### a. Kelas *Reader*

Kelas *java.io.Reader* merupakan sebuah kelas abstrak yang menyediakan berbagai macam *method* untuk melakukan proses pembacaan dalam bentuk karakter *Unicode* dari berbagai sumber data seperti harddisk, keyboard atau memori. Karena merupakan sebuah kelas abstrak, maka kelas ini tidak bisa diinstantiasi, tetapi sub-kelasnya lah yang akan digunakan untuk proses pembacaan data. Berikut adalah kelas-kelas turunan dari kelas *Reader*.



Gambar 7-3 Hirarki Kelas *Reader*

Terdapat berbagai macam *method* dalam kelas *Reader*, tetapi sebenarnya hanya *method* *read()* dan *close()* saja yang kita perlukan.

Tabel 7-7 Method dalam kelas *Reader*

Method	Keterangan
<code>int read()</code>	Membaca satu karakter dan mengembalikan sebuah nilai integer dari karakter yang dibaca. Jika yang dibaca adalah EOF maka akan dikembalikan nilai -1.
<code>int read(char buffer[])</code>	Membaca karakter dan memasukkannya kedalam sebuah <i>array</i> buffer dan mengembalikan sebuah bilangan dari karakter yang dibaca. Jika EOF akan mengembalikan nilai -1
<code>abstract void close()</code>	Menutup <i>stream</i> .



### 1) Kelas `FileReader`

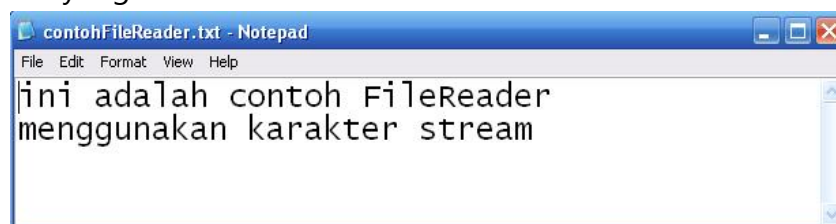
Kelas `FileReader` digunakan untuk membaca karakter dari dalam file, tetapi kelas ini tidak memiliki *method* didalamnya. *Method*-nya sendiri diturunkan dari kelas induknya yaitu `Reader` dan `InputStreamReader`. Terdapat beberapa konstruktor dala kelas `FileReader` ini.

- `FileReader(File objFile)`  
Membuat sebuah kelas `FileReader` dengan inputan sebuah obyek `objFile`.
- `FileReader(String namaFile)`  
Membuat sebuah kelas `FileReader` dengan inputan sebuah nama `namaFile`.

Berikut adalah contoh penggunaan `FileReader`

```
import java.io.*;
public class contohFR {
    public static void main(String args[]) {
        try {
            File file=new File("cthFileReader.txt");
            FileReader f=new FileReader(file);
            int ch;
            while((ch=f.read())!=-1) {
                Sistem.out.print((char)ch);
            }
        }
        catch(FileNotFoundException fnf) {
            System.out.println(fnf);
        }
        catch(IOException io) {
            System.out.println(io);
        }
    }
}
```

File yang akan dibaca dari kode diatas berisi hal berikut.



Berikut adalah hasil eksekusi dari kode diatas



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(c) Copyright 1985-2001 Microsoft Corp.

D:\IO>java contohFR
ini adalah contoh FileReader
menggunakan karakter stream
D:\IO>
```

## 2) Kelas `BufferedReader`

Kelas `BufferedReader` membaca teks dalam bentuk *stream* karakter dan memasukkannya kedalam sebuah buffer didalam *stream* input. Dengan menggunakan buffer, maka proses pembacaan bisa dilakukan lebih dari satu kali pada saat yang sama, sehingga bisa lebih meningkatkan performansi dari aplikasi. Contoh penggunaan dari kelas ini adalah untuk mendapatkan inputan dari user melalui konsole. Beberapa konstruktor dari kelas `BufferedReader` ini.

- `BufferedReader(Reader objStream)`  
Membuat sebuah obyek `BufferedReader` dengan ukuran buffer default.
- `BufferedReader(Reader objStream, int ukuranBuffer)`  
Membuat sebuah obyek `BufferedReader` dengan ukuran buffer sebesar `ukuranBuffer`.

Berikut adalah contoh dari `BufferedReader`

```
import java.io.*;
public class contohBR {
    public static void main(String args[])
    {
        System.out.println("Ketik :");

        BufferedReader br = new BufferedReader
                               (new InputStreamReader(System.in));
        String str=new String();
        try
        {
            str=br.readLine();
        }
        catch(IOException io)
        {
            System.out.println(io);
        }
        System.out.println();
        System.out.println("Isi ketikan: ");
    }
}
```



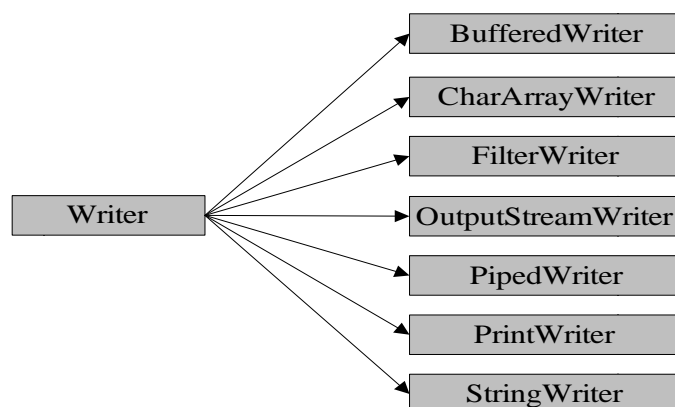
```
        System.out.println(str);  
    }  
}
```

Hasil dari kode diatas adalah

```
C:\WINDOWS\system32\cmd.exe  
D:\IO>java contohBR  
Ketik :  
contoh dari kelas BufferedReader  
  
Isi ketikan:  
contoh dari kelas BufferedReader  
D:\IO>_
```

### b. Kelas Writer

Kelas `java.io.Writer` merupakan sebuah kelas abstrak yang menyediakan berbagai macam *method* untuk melakukan proses penulisan dalam bentuk karakter *Unicode* ke berbagai tujuan data seperti harddisk dan monitor. Karena merupakan sebuah kelas abstrak, maka kelas ini tidak bisa diinstantiasi, tetapi sub-kelasnya lah yang akan digunakan untuk proses pembacaan data. Berikut adalah kelas-kelas turunan dari kelas `Writer`.



Gambar 7-4 Hirarki kelas `Writer`

Terdapat banyak *method* dalam kelas `Writer`, tetapi hanya *method* `write()`, `flush()` dan `close()` saja yang kita butuhkan.

Tabel 7-8 Method dalam kelas `Writer`

Method	Keterangan
<code>int write(int ch)</code>	Menulis sebuah karakter kedalam <i>stream</i> output-karakter.
<code>int write(char buffer[])</code>	Menulis sebuah <i>array</i> karakter kedalam <i>stream</i> output-karakter.
<code>abstract void close()</code>	Menutup <i>stream</i> .
<code>Void flush()</code>	Mem-flush buffer output



## 1) Kelas FileWriter

Kelas `FileWriter` digunakan untuk menulis data kedalam sebuah file, tetapi kelas ini tidak memiliki *method* didalamnya. *Method*nya sendiri diturunkan dari kelas induknya yaitu `Writer` dan `OutputStreamReader`. Terdapat beberapa konstruktor dala kelas `FileWriter` ini.

- `FileWriter(String namaFile)`  
Membuat sebuah obyek `FileWriter` dengan parameter inputan nama file yang akan ditulis.
- `FileWriter(String namaFile, boolean boolAppend)`  
Membuat sebuah obyek `FileWriter` dengan parameter inputan nama file yang akan ditulis, sedangkan `boolAppend` digunakan untuk menentukan bahwa penulisan akan dilakukan secara menambahkan data di bagian paling bawah jika `boolAppend` bernilai `true`.
- `FileWriter(File objFile)`  
Membuat sebuah obyek `FileWriter` dengan parameter inputan sebuah obyek dari kelas `File`.

Berikut adalah contoh dari penggunaan kelas `FileWriter`.

```
import java.io.*;
public class ContohFW {
    public static void main (String args[]) {
        String str="contoh dari";
        String str2="penggunaan kelas FileWriter";
        try {
            FileWriter fout = new FileWriter("cthFileWriter.txt" );
            fout.write(str, 0, str.length() );
            fout.write(str2, 0, str2.length() );
            System.out.println("Data sudah ditulis...");
            fout.close();
        }
        catch (IOException io) {
            System.out.println(io);
        }
    }
}
```

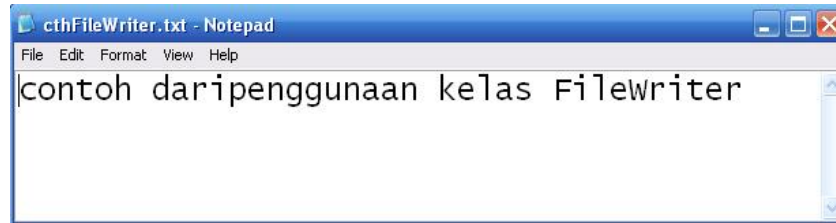
Hasil dari kode diatas

```
ca C:\WINDOWS\system32\cmd.exe
D:\IO>java ContohFW
Data sudah ditulis...
D:\IO>
```





File cthFileWriter.txt akan berisi.



## 2) Kelas **BufferedWriter**

Kelas `BufferedWriter` menulis teks dalam bentuk *stream* karakter dan memasukkannya kedalam sebuah buffer didalam *stream* output. Dengan menggunakan buffer, maka proses penulisan bisa dilakukan lebih dari satu kali pada saat yang sama, sehingga bisa lebih meningkatkan performansi dari aplikasi.

`BufferedWriter` memiliki beberapa konstruktor

- `BufferedWriter(Writer streamOutput)`  
Membuat sebuah obyek `BufferedWriter` dengan ukuran buffer default.
- `BufferedWriter(Writer streamOutput, int ukuranBuffer)`  
Membuat sebuah obyek `BufferedWriter` dengan ukuran buffer sebesar `ukuranBuffer`

Berikut adalah contoh penggunaan `BufferedWriter`

```
import java.io.*;
public class contohBW {
    public static void main(String args[]) {
        int ch;
        BufferedWriter bw=null;
        System.out.println("Ketik kalimat");
        System.out.println("akhiri dengan [ctrl + z]:");

        try {
            Bw = new BufferedWriter(new
                                   FileWriter("cthBuffWriter.txt"));
            while((ch=Sistem.in.read())!=-1)
            {
                bw.write(ch);
            }
            bw.flush(); //penulisan kedalam file
        }
        catch(IOException io) {
            System.out.println(io);
        }
    }
}
```

Proses eksekusi dari contoh diatas.



```
C:\WINDOWS\system32\cmd.exe
D:\IO>java contohBW
Ketik kalimat
akhiri dengan [ctrl + z]:
Mengetikkan sebuah kalimat

diakhiri dengan menekan tombol kontrol z
^Z
D:\IO>
```

Isi file cthBuffWriter.txt adalah

```
cthBuffWriter.txt - Notepad
File Edit Format View Help
Mengetikkan sebuah kalimat

diakhiri dengan menekan tombol kontrol z
```

### 3) Kelas `PrintWriter`

Kelas `PrintWriter` merupakan turunan dari kelas `Writer` dan akan menuliskan karakter yang telah terformat kedalam *stream* karakter output. Didalam kelas ini terdapat beberapa *method* yang akan digunakan yaitu `print()` dan `println()` terdapat beberapa konstruktor dalam kelas ini.

- `PrintWriter(OutputStream streamObj)`  
Membuat sebuah obyek dari kelas `PrintWriter` dengan menggunakan obyek dari kelas `OutputStream`.
- `PrintWriter(OutputStream streamObj, boolean autoflush)`  
Membuat sebuah obyek dari kelas `PrintWriter` dengan menggunakan obyek dari kelas `OutputStream`. Parameter `autoflush` menandakan bahwa *method* `println()` akan memflush kedalam buffer output.
- `PrintWriter(Writer outWriter, boolean autoflush)`  
Membuat sebuah obyek dari kelas `PrintWriter` dengan menggunakan obyek dari karakter *stream* output. Parameter `autoflush` menandakan bahwa *method* `println()` akan memflush kedalam buffer output.

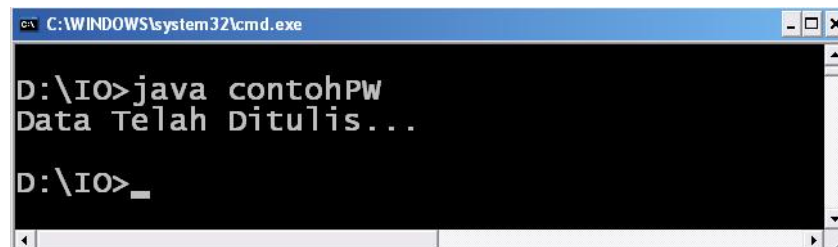
Contoh penggunaan kelas `PrintWriter`.

```
import java.io.*;
public class contohPW {
    public static void main(String args[]){
        PrintWriter pw=null;
        try {
            FileWriter file=new FileWriter("cthPrintWriter.txt");
```

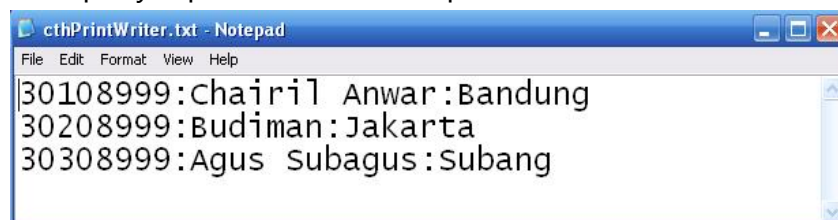


```
        pw = new PrintWriter(file);
    }
    catch(IOException io) {
        System.out.println(io);
    }
    pw.print("30108999:");
    pw.print("Chairil Anwar:");
    pw.print("Bandung");
    pw.println();
    pw.print("30208999:");
    pw.print("Budiman:");
    pw.print("Jakarta");
    pw.println();
    pw.print("30308999:");
    pw.print("Agus Subagus:");
    pw.print("Subang");
    pw.println();
    System.out.println("Data Telah Ditulis...");
    pw.close();
}
}
```

Eksekusi dari kode diatas



Hasil penyimpanan data diatas pada file cthPrintWriter.txt



#### D. Implementasi *RandomAccessFile*

*Random access file* (pengaksesan file secara acak) memungkinkan kita untuk mengakses isi dari sebuah file secara tidak berurutan. Dengan cara ini kita bisa membaca dan menulis data teks dan byte pada lokasi manapun dalam sebuah file. Ambil sebuah kasus ketika yang kita inginkan tidak semua teks dari sebuah file kita butuhkan, tetapi hanya pada posisi tertentu saja, misal pada baris ke-sekian. Maka dengan menggunakan *Random access file* ini kita bisa langsung menuju ke posisi yang diinginkan dan mengambil teks yang kita butuhkan saja.



Kelas `java.io.RandomAccessFile` mengimplementasikan sebuah interface `DataInput` dan `DataOutput` yang bisa digunakan untuk melakukan proses baca dan tulis kedalam file. Didalam kelas ini, kita harus mendefinisikan proses apa yang akan kita lakukan, apakah membaca atau juga akan menulis kedalam file tersebut.

Terdapat beberapa konstruktor yang bisa digunakan dalam kelas `RandomAccessFile`, antara lain:

- `RandomAccessFile(File obyekFile, String mode):`

Membuat sebuah obyek dari kelas `RandomAccessFile`, dimana argumen `obyekFile` adalah obyek yang menentukan file mana yang akan dibuka, sedangkan argumen `mode` menentukan jenis hak akses pembukaan file tersebut.

- `RandomAccessFile(String namaFile, String mode):`

Membuat sebuah obyek dari kelas `RandomAccessFile`, dimana argumen `namaFile` adalah `nama` dari file yang akan dibuka, sedang argumen `mode` menentukan jenis hak akses pembukaan file tersebut.

Terdapat beberapa jenis hak akses pembukaan file, antara lain:

Tabel 7-9 Jenis akses pembukaan file

Nilai	Keterangan
"r"	Membuka file hanya untuk dilakukan proses pembacaan saja. Jika dilakukan proses penulisan pada obyek ini maka akan menghasilkan sebuah <code>IOException</code>
"rw"	Membuka sebuah file untuk dilakukan proses baca dan tulis. Jika file yang ditunjuk tidak ada, maka akan dibuat sebuah file baru.
"rws"	Membuka sebuah file untuk dilakukan proses baca dan tulis, sebagaimana mode "rw", untuk mode "s" juga akan dilakukan proses update terhadap isi dan metadata dari file tersebut, yang akan dituliskan secara tersinkronisasi.
"rwd"	Membuka sebuah file untuk dilakukan proses baca dan tulis, sebagaimana mode "rw", untuk mode "s" juga akan dilakukan proses update terhadap isi dari file tersebut, yang akan dituliskan secara tersinkronisasi.



Terdapat beberapa *method* yang bisa digunakan didalam sebuah obyek dari kelas File, antara lain :

Tabel 7-10 Method dalam kelas RandomAccessFile

Method	Keterangan
void close()	Menutup sebuah obyek RandomAccessFile dan melepas semua sumber daya yang digunakan, seperti <i>stream</i> dan pointer yang digunakan dalam file tersebut.
long getFilePointer()	Mendapatkan posisi pointer saat ini didalam sebuah file.
long length()	Mendapatkan panjang dari sebuah file.
void seek(long position)	Menentukan posisi pointer kepada lokasi yang ditentukan (posisi sebelum lokasi tersebut).
int skipBytes(int n)	Akan melewati sekian n byte karakter dari posisi sekarang.

Berikut adalah contoh penggunaan RandomAccessFile.

```
import java.io.RandomAccessFile;
import java.io.IOException;
class RAC
{
    RandomAccessFile file=null;
    public void aksesFile()
    {
        try
        {
            /* membuka file work.txt, dengan mode akses read dan write jika
            file tidak ditemukan, maka akan dibuat sebuah file baru */
            file=new RandomAccessFile("work.txt","rw");
            file.writeChar('A');
            file.writeChar('K');
            file.writeInt(4);
            file.writeInt(6);
            file.seek(0);

            System.out.println(file.readChar());
            System.out.println(file.readChar());
            System.out.println(file.readInt());
            System.out.println(file.readInt());
            file.close();
        }
        catch(IOException e)
        {
            System.out.println("Exception: " +e);
        }
    }
}
```



Class driver dari kelas RAC

```
class ContohRandomAccessFile {  
    public static void main(String[] args) {  
        RAC obj = new RAC();  
        obj.aksessFile();  
    }  
}
```

Ouput dari contoh diatas

```
C:\WINDOWS\system32\cmd.exe  
D:\IO>java ContohRandomAccessFile  
A  
K  
4  
6  
D:\IO>
```

## Rangkuman

- Penggunaan kelas `File` untuk mengakses atribut yang dimiliki oleh sebuah file atau direktori.
- Penggunaan *stream* dalam Java untuk melakukan penulisan dan pembacaan data.
- Terdapat dua jenis *stream* dalam Java
  - *Byte Stream*
  - *Character Stream*
- Kelas `InputStream` digunakan untuk melakukan proses pembacaan data berbentuk *byte stream* dari berbagai macam sumber, misal keyboard dan file.
- Kelas `InputStream` terdiri dari kelas `AudioInputStream`, `ByteArrayInputStream`, `FileInputStream`, `FilterInputStream`, `InputStream`, `ObjectInputStream`, `PipedInputStream`, `SequenceInputStream`, `StringBufferInputStream`
- Kelas `OutputStream` digunakan untuk melakukan prose penulisan data berbentuk *byte stream* ke berbagai macam tujuan, misal monitor dan file.
- Kelas `OutputStream` terdiri dari kelas turunan `ByteArrayOutputStream`, `FileOutputStream`, `FilterOutputStream`, `ObjectOutputStream`, `OutputStream`, `PipedOutputStream`.
- Kelas `Reader` digunakan untuk melakukan proses pembacaan data berbentuk *character stream* dari berbagai macam sumber, misal keyboard dan file.



- Kelas `Reader` terdiri dari kelas turunan `BufferedReader`, `CharArrayReader`, `FilterReader`, `InputStreamReader`, `PipedReader`, `StringReader`
- Kelas `Writer` digunakan untuk melakukan prose penulisan data berbentuk character stream ke berbagai macam tujuan, misal monitor dan file.
- Kelas `Writer` terdiri dari kelas turunan `BufferedWriter`, `CharArrayWriter`, `FilterWriter`, `OutputStreamWriter`, `PipedWriter`, `PrintWriter`, `StringWriter`.
- Pembacaan dan penulisan kedalam file dengan akses secara acak menggunakan kelas `RandomAccessFile`.

## Latihan

1. Buatlah sebuah aplikasi yang digunakan untuk melakukan proses administrasi data mahasiswa yang terdiri dari.
  - a. Input data mahasiswa.
  - b. Cari Data mahasiswa
  - c. Hapus Data Mahasiswa
  - d. Simpan data mahasiswa kedalam sebuah fileDi awal eksekusi, aplikasi akan mengambil data-data mahasiswa yang sebelumnya telah tersimpan.
2. Seperti soal diatas, Buat sebuah aplikasi yang akan digunakan untuk memproses nilai mahasiswa. Data yang disimpan adalah data nilai beserta indeksnya.



## **Daftar Pustaka**

Deitel, H.M. Java How To Program, sixth Edition. 2004, Prentice Hall.

Exercise 10, Introduction to Computer Science Januari 2003, TU Darmstadt

H.M Deitel, Java How to Program, Sixth Edition. Prentice Hall, August 2004

[http://cs.bilgi.edu.tr/pages/standards\\_project/java\\_CodingStyle.pdf](http://cs.bilgi.edu.tr/pages/standards_project/java_CodingStyle.pdf)

<http://cs-education.stanford.edu/class/cs107/handouts/42SimpleInheritance.pdf>

<http://faculty.ksu.edu.sa/Shammami/Documents/DocCSC113/Lectures%20Notes/ChapterRelationship.pdf>

<http://poss.ipb.ac.id/files/JENI-Intro1-Bab07-Java%20Array.pdf>

[http://www.cs.brandeis.edu/~cs21a/Lectures/Lec1\\_cs22.pdf](http://www.cs.brandeis.edu/~cs21a/Lectures/Lec1_cs22.pdf)

<http://www.cs.nott.ac.uk/~dgc/G51PRG/Using%20Inheritance%20in%20Java.pdf>

<http://www.javapassion.com/javase/>

J.E.N.I Pengenalan Pemrograman 1 Versi 1.2 juni 2007, Joyce Avestro, JARDIKNAS

Java Fundamental NIIT India

Object Oriented Programming using Java Anban Pillay, Shcool of Computer Science, University of KwaZulu-Natal Durban, Februari 2207

Poo, Danny. Kiong, Derek. Ashok, Swarnalatha. "Object Oriented Programming and Java, second Edition", 2008. Springer.

Programming in Java NIIT India

Puji Hartono, ST. Modul Praktikum Pemrograman Berorientasi Objek, versi 1.0

The Java Reference Library, 1996,1997. O'Reilly & Associates

Weisfeld, Matt. "The Object-Oriented Thought Process, Developer's Library. 2009, Addison-Wesley.