

**Nama Kelompok:**

Anggi Wijaya 0606104201  
Evi Yulianti 0606031465  
Syarief Abdurrahman 0606101950

**Review:****Bab 4. Bahasa Java****Komentar Umum :**

Bab ini sungguh amat sangat menarik untuk dibahas, karena bahasa Java saat ini sudah sangat mengglobalisasi dalam dunia programming. Bahkan bahasa java pun sudah dipakai untuk membuat Operating System, contohnya yaitu Sistem Operasi Solaris. Apalagi saat ini bahasa java telah menjadi open source dan keuntungan dalam penggunaan bahasa Java yaitu bahasa multiplatform sehingga bahasa Java dapat diaplikasikan di semua sistem operasi dengan menggunakan Java Virtual Machine. Bahkan di Indonesia pun telah berdiri komunitas Java yang bernama JENI ( Java for Edu Network in Indonesia ) <http://jeni.jardiknas.org/web/guest/home>.

**Hubungan dengan bab sebelumnya/selanjutnya**

Kami rasa bab ini tidak memiliki hubungan dengan bab sebelumnya tetapi bab ini memiliki kaitan dengan bab sesudahnya, yakni bab 25, 26, dan 27. Bahasa Java digunakan sebagai alat bantu menjelaskan konsep interaksi proses (Reader/Writer, Producer/Consumer, dll), melalui contoh-contoh programnya.

**Komentar kelengkapan per bagian****Bag 4.1**

Bagian ini sudah cukup baik tentang sejarah bahasa Java. Mungkin bisa diberikan tokoh/institusi penting yang berperan dalam sejarah perkembangan Java.

**Bag 4.2**

Bagian ini sudah cukup baik, mungkin perlu dijelaskan sedikit tentang kelebihan dan kekurangan bahasa Java dibandingkan dengan bahasa pemrograman yang lain.

**Bag 4.3**

Bagian ini sudah cukup mewakili serta memberikan penjelasan yang tepat tentang bagian dari Java API.

**Bag 4.4**

Bagian ini sudah cukup menjelaskan dan memberikan gambaran yang tepat tentang JVM (Java Virtual Machine).

**Bag 4.5**

Bagian ini kurang memberikan contoh sistem operasi dalam Java. Mungkin bisa dituliskan salah satu contohnya yaitu Solaris 10 (versi terbaru) yang dirilis pada 31 Januari 2005.

**Bag 4.6 dan 4.7**

Bagian ini sudah cukup baik dan jelas. Namun penjelasan tentang istilah-istilah yang digunakan terkesan minimalis.

**Bag 4.8, 4.9, 4.10, 4.11**

Bagian ini sebaiknya digabung, karena bagian tersebut menjelaskan tentang atribut beserta hak aksesnya. Tujuannya adalah agar bab ini bisa dijelaskan secara lebih ringkas namun tetap jelas,

karena walaupun dalam 1 bagian, kami akan membandingkan penggunaan atribut berdasarkan hak aksesnya. Jadi, hanya dibutuhkan 1 contoh program saja yang mencakup semua atribut tersebut.

Bag 4.12, 4.13, 4.14, 4.15

Bagian ini sudah cukup jelas, hanya saja untuk bag.14 source code Java nya agak sulit dibaca. karena tidak langsung memakai sintaks bahasa Java.

Bagian 4.16, 4.17, 4.18

Bagian ini sudah cukup jelas,tetapi untuk bagian 4.16 bisa ditambah contoh package yang ada di library Java, seperti *java.util,javax.swing*.

Bagian 4.18

Bagian ini perlu merangkum konsep penting Java secara singkat seperti *object, class, interface, method, inheritance*.

**Usulan kelengkapan :**

- Diagram proses eksekusi java (write->compile->load->verify->execute)

# Bab 4. Bahasa Java

## 4.1. Pendahuluan

Java adalah sebuah teknologi yang diperkenalkan oleh Sun Microsystems pada pertengahan tahun 1990. Menurut definisi dari Sun, Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan. Kita lebih menyukai menyebut Java sebagai sebuah teknologi dibanding hanya sebuah bahasa pemrograman, karena Java lebih lengkap dibanding sebuah bahasa pemrograman konvensional. Teknologi Java memiliki tiga komponen penting, yaitu:

- *Programming-language specification*
- *Application-programming interface*
- *Virtual-machine specification*

## 4.2. Bahasa Pemrograman Java

Bahasa Java dapat dikategorikan sebagai sebuah bahasa pemrograman berorientasi objek, pemrograman terdistribusi dan bahasa pemrograman *multithreaded*. Objek Java dispesifikasi dengan membentuk kelas. Untuk masing-masing kelas Java, kompiler Java memproduksi sebuah file keluaran arsitektur netral yang akan jalan pada berbagai implementasi dari *Java Virtual Machine (JVM)*. Awalnya Java sangat digemari oleh komunitas pemrograman internet, karena Java mendukung untuk *applets*, dimana program dengan akses sumber daya terbatas yang jalan dalam sebuah *web browser*. Java juga menyediakan dukungan level tinggi untuk *networking* dan objek terdistribusi.

Java juga dianggap sebagai sebuah bahasa yang aman. Tampilan ini pada khususnya penting menganggap bahwa sebuah program Java boleh mengeksekusi silang sebuah jaringan terdistribusi.

Sehingga bahasa Java saat ini termasuk bahasa pemrograman yang relatif mudah untuk dipelajari karena banyaknya contoh dan konsep yang beredar luas, baik berupa buku maupun di internet. Tetapi program yang dibuat dalam bahasa Java juga relatif lebih banyak membutuhkan waktu saat di eksekusi (lebih lama) dikarenakan untuk menjalankan programnya dibutuhkan JVM (perantara antar program dan sistem operasi).

## 4.3. Java API

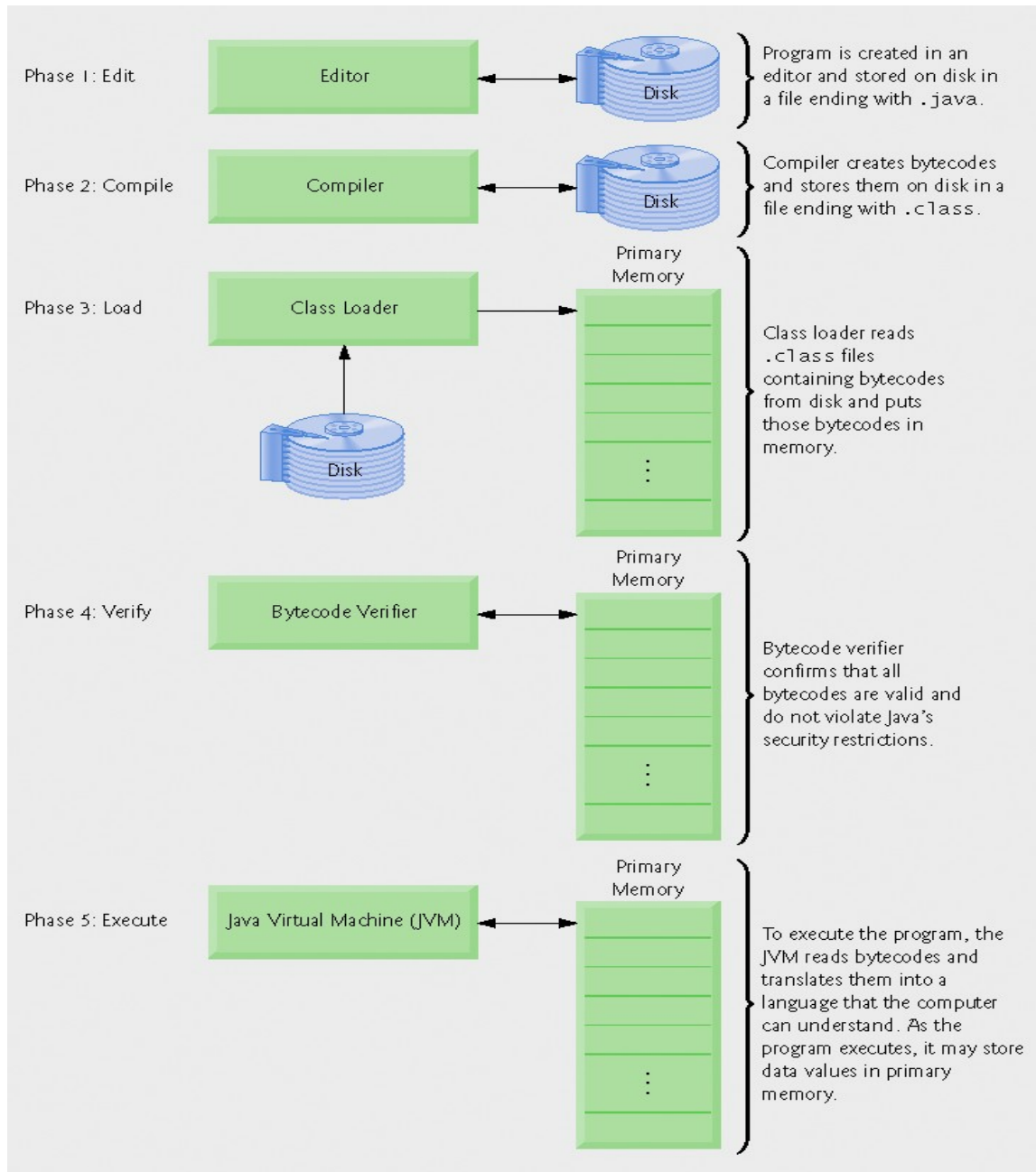
Java API terdiri dari tiga bagian utama:

- **Java Standard Edition (SE)**, sebuah standar API untuk merancang aplikasi desktop dan *applets* dengan bahasa dasar yang mendukung grafis, M/K, keamanan, konektivitas basis data dan jaringan.
- **Java Enterprise Edition (EE)**, sebuah inisiatif API untuk merancang aplikasi server dengan mendukung untuk basis data.
- **Java Micro Edition (ME)**, sebuah API untuk merancang aplikasi yang jalan pada alat kecil seperti telepon genggam, komputer genggam dan pager.

## 4.4. Java Virtual Machine

Java Virtual Machine (JVM) adalah sebuah spesifikasi untuk sebuah komputer abstrak. JVM terdiri dari sebuah kelas pemanggil dan sebuah interpreter Java yang mengeksekusi kode

arsitektur netral. Kelas pemanggil memanggil file **.class** dari kedua program Java dan Java API untuk dieksekusi oleh interpreter Java. Interpreter Java mungkin sebuah perangkat lunak interpreter yang menterjemahkan satu kode byte pada satu waktu, atau mungkin sebuah *just-intime* (JIT) kompiler yang menurunkan *bytecode* arsitektur netral kedalam bahasa mesin untuk *host computer*. Berikut adalah bagan tahap proses eksekusi Java:



## 4.5. Sistem Operasi Java

Sistem operasi biasanya ditulis dalam sebuah kombinasi dari kode bahasa C dan assembly, terutama disebabkan oleh kelebihan performa dari bahasa tersebut dan memudahkan komunikasi dengan perangkat keras.

Satu kesulitan dalam merancang sistem basis bahasa adalah dalam hal proteksi memori, yaitu memproteksi sistem operasi dari pemakai program yang sengaja memproteksi pemakai

program lainnya. Sistem operasi tradisional mengharapkan pada tampilan perangkat keras untuk menyediakan proteksi memori. Sistem basis bahasa mengandalkan pada tampilan keamanan dari bahasa. Sebagai hasilnya, sistem basis bahasa menginginkan pada alat perangkat keras kecil, yang mungkin kekurangan tampilan perangkat keras yang menyediakan proteksi memori.

Versi terakhir dari Sistem Operasi Java saat ini bernama Solaris 10 yang dirilis pada tanggal 31 Januari 2005. Hingga saat ini telah ada versi *Solaris 10 update 4*.

## 4.6. Dasar Pemrograman

Java2 adalah generasi kedua dari Java *platform* (generasi awalnya adalah Java Development Kit). Java berdiri di atas sebuah mesin interpreter yang diberi nama JVM. JVM inilah yang akan membaca *bytecode* dalam file *.class* dari suatu program sebagai representasi langsung program yang berisi bahasa mesin. Oleh karena itu, bahasa Java disebut sebagai bahasa pemrograman yang *portable* karena dapat dijalankan pada berbagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM.

*Platform* Java terdiri dari kumpulan *library*, JVM, kelas- kelas *loader* yang dipaket dalam sebuah lingkungan rutin Java, dan sebuah *compiler*, *debugger*, dan perangkat lain yang dipaket dalam Java Development Kit (JDK). Java2 adalah generasi yang sekarang sedang berkembang dari *platform* Java. Agar sebuah program Java dapat dijalankan, maka file dengan ekstensi *.java* harus dikompilasi menjadi file *bytecode*. Untuk menjalankan *bytecode* tersebut dibutuhkan JRE (*Java Runtime Environment*) yang memungkinkan pemakai untuk menjalankan program Java, hanya menjalankan, tidak untuk membuat kode baru lagi. JRE berisi JVM dan *library* Java yang digunakan.

*Platform* Java memiliki tiga buah edisi yang berbeda, yaitu J2EE (*Java2 Enterprise Edition*), J2ME (*Java2 Micro Edition*) dan J2SE (*Java2 Second Edition*). J2EE adalah kelompok dari beberapa API (*Application Programming Interface*) dari Java dan teknologi selain Java. J2EE sering dianggap sebagai *middleware* atau teknologi yang berjalan di *server*, namun sebenarnya J2EE tidak hanya terbatas untuk itu. Faktanya J2EE juga mencakup teknologi yang dapat digunakan di semua lapisan dari sebuah sistem informasi. Implementasi J2EE menyediakan kelas dasar dan API dari Java yang mendukung pengembangan dari rutin standar untuk aplikasi klien maupun *server*, termasuk aplikasi yang berjalan di *web browser*. J2SE adalah lingkungan dasar dari Java, sedangkan J2ME merupakan edisi *library* yang dirancang untuk digunakan pada *device* tertentu seperti *pagets* dan *mobile phone*.

Java merupakan bahasa pemrograman yang bersifat *case sensitive* yang berarti penulisan menggunakan huruf besar ataupun huruf kecil pada kode program dapat berarti lain. Misalnya penulisan "System" akan diartikan berbeda dengan "system" oleh interpreter. Java tidak seperti C++, Java tidak mendukung pemrograman prosedural, tapi mendukung pemrograman berorientasi objek sehingga ada sintaks *class* pada kode programnya.

## 4.7. Objek dan Kelas

Sebuah kelas menyerupai sebuah struktur yang merupakan tipe data sendiri, misalkan tipe data titik yang terdiri dari koordinat x dan y. Bahasa Java telah menggunakan sebuah kelas untuk menyatakan tipe data titik karena bahasa pemrograman Java merupakan bahasa pemrograman berorientasi objek murni sehingga tidak mengenal struktur tapi mengenal apa yang disebut dengan kelas.

Perbedaan sebuah kelas dengan sebuah struktur adalah sebuah kelas dapat berdiri sendiri dan dapat digunakan untuk berbagai keperluan kelas-kelas yang lain, sedangkan sebuah struktur tidak dapat berdiri sendiri. Sebuah kelas lebih fleksibel untuk digunakan oleh kelas lain tanpa harus membongkar kode program utama, sedangkan jika digunakan struktur maka kode program harus dibongkar untuk disalin bagian strukturnya ke kode program utama yang lain. Sebuah *file* dapat terdiri dari berbagai kelas, namun biasanya pada bahasa pemrograman Java sebuah *file* hanya terdiri dari satu kelas yang disimpan dengan nama kelas, misal *file* List.java berisi kelas *List*. Namun jika kelas yang dibuat misalnya `public class nama_kelas`, maka kelas itu harus disimpan dalam satu *file* hanya untuk satu kelas. Setelah dilakukan kompilasi maka pada Java akan ada sebuah *file* .class yang berisi *bytecode* dari setiap kelas. Jika sebuah *file* terdiri dari dua kelas maka setelah dikompilasi akan dihasilkan dua buah *file* .class yang nantinya akan dibaca oleh interpreter Java saat program dieksekusi.

Sebuah kelas saat program dieksekusi dan perintah `new` dijalankan, maka akan dibuat sebuah objek. Objek adalah elemen pada saat *runtime* yang akan diciptakan, dimanipulasi dan dihancurkan saat eksekusi sehingga sebuah objek hanya ada saat sebuah program dieksekusi, jika masih dalam bentuk kode, disebut sebagai kelas jadi pada saat *runtime* (saat sebuah program dieksekusi), yang kita punya adalah objek, di dalam teks program yang kita lihat hanyalah kelas.

## 4.8. Atribut

Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas, Atribut dapat memiliki hak akses `private`, `public` maupun `protected`. Sebuah atribut yang dinyatakan sebagai *private* hanya dapat diakses secara langsung oleh kelas yang membungkusnya, sedangkan kelas lainnya tidak dapat mengakses atribut ini secara langsung. Sebuah atribut yang dinyatakan sebagai *public* dapat diakses secara langsung oleh kelas lain di luar kelas yang membungkusnya. Sebuah atribut yang dinyatakan sebagai *protected* tidak dapat diakses secara langsung oleh kelas lain di luar kelas yang membungkusnya, kecuali kelas yang mengaksesnya adalah kelas turunan dari kelas yang membungkusnya. Contoh penggunaan atribut-atribut tersebut ada pada kelas di bawah ini:

```
class Elemen
{
    private Elemen next1;
    public Elemen next2;
    protected Elemen next3;
    Elemen()
    {}
}
```

Jika kelas lain mempunyai objek dari kelas `Elemen`, dengan dilakukan pembuatan objek sbb:

```
Elemen e = new Elemen()
```

maka :

- atribut `next1`, tidak dapat diakses melalui `e.next1` karena atribut yang hak aksesnya `private` hanya dapat diakses oleh kelas yang membungkusnya. Jadi pengaksesannya dapat dilakukan dengan membuat method pada kelas `Elemen` yang akan mengembalikan nilai atribut `next1` tersebut. Contoh:

```
public getNext1()
{
```

```
        return next1;
    }
```

Kelas lain yang akan mengakses atribut `next1` pada kelas `Elemen` harus memanggil method `getNext1()`. Izin akses `private` ini ditujukan untuk melindungi atribut - atributnya agar tidak dapat diakses oleh kelas lain

- atribut `next2`, dapat diakses kelas lain melalui `e.next2` karena bersifat `public` yang memungkinkan kelas lain untuk mengakses secara langsung terhadap kelas tersebut. Jika sebuah atribut tidak ditulis izin aksesnya, maka interpreter Java akan menganggap atribut tersebut memiliki izin akses `public`.
- Atribut `next3`, dapat diakses secara langsung oleh kelas lain yang merupakan turunan kelas `Elemen`. Izin akses `protected` ini dimaksudkan untuk melindungi atribut agar tidak diakses secara langsung oleh sembarang kelas lain, kecuali oleh kelas turunannya.

## 4.9. Metode

Metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada pemrograman prosedural. Pada sebuah metode di dalam sebuah kelas juga memiliki izin akses seperti halnya atribut pada kelas, izin akses itu antara lain *private*, *public* dan *protected* yang memiliki arti sama pada izin akses atribut yang telah dibahas sebelumnya. Sebuah kelas boleh memiliki lebih dari satu metode dengan nama yang sama asalkan memiliki parameter masukan yang berbeda sehingga kompiler atau interpreter dapat mengenali metode mana yang dipanggil. Hal ini dinamakan *overloading*.

Di dalam sebuah kelas, terdapat juga yang disebut sebagai metode atau atribut statis yang memiliki kata kunci `static`. Maksud dari statis di sini adalah metode yang dapat diakses secara berbagi dengan semua objek lain tanpa harus membuat objek yang memiliki metode statis tadi (tanpa proses `new`), tapi sebuah metode statis mempunyai keterbatasan yaitu hanya dapat mengakses atribut atau metode lain di dalam kelas yang membungkusnya yang juga bersifat statis. Metode statis biasanya diimplementasikan untuk metode main.

## 4.10. Konstruktor

Sebuah kelas harus memiliki sebuah metode yang disebut sebagai konstruktor. nama sebuah konstruktor harus sama dengan nama dari sebuah kelas, misalkan kelas `Elemen` maka konstruktornya adalah `Elemen()`. Sebuah konstruktor juga dapat menerima sebuah masukan seperti halnya prosedur pada pemrograman prosedural. Fungsi dari sebuah konstruktor adalah: mengalokasikan sebuah objek saat program dieksekusi, memberikan nilai awal sebagai inisialisasi dari semua atribut yang perlu diinisialisasi dan mengerjakan proses-proses yang diperlukan saat sebuah objek dibuat.

Namun pada kenyataannya sebuah konstruktor dapat tidak berisi apa-apa, hal ini jika memang tidak diperlukan adanya inisialisasi atau proses yang dikerjakan ketika sebuah objek dibuat. Konstruktor tersebut disebut sebagai konstruktor *default*. Konstruktor harus bersifat *public* karena sebuah konstruktor akan diakses oleh kelas lain untuk membuat objek suatu kelas.

Sebuah kelas dapat memiliki konstruktor lebih dari satu. Pada saat eksekusi program, kompiler atau interpreter akan mencari konstruktor mana yang sesuai dengan konstruktor yang dipanggil, hal ini juga merupakan *overloading*.

## 4.11. Inheritance

*Inheritance* atau pewarisan pada pemrograman berorientasi objek merupakan suatu hubungan dua buah kelas atau lebih. Dalam hal ini ada kelas yang memiliki atribut dan metode yang sama dengan kelas lainnya beserta atribut dan metode tambahan yang merupakan sifat khusus kelas yang menjadi turunannya. Sebagai contoh, misalkan ada sebuah kelas Titik yang mempunyai kelas turunan Titik3D:

```
class Titik
{
    private Integer x;
    private Integer y;

    Titik()
    {
        x = 0;
        y = 0;
    }

    public int getX()
    {
        return x;
    }

    public int getY()
    {
        return y;
    }
}

public class Titik3D extends Titik
{
    private Integer z;

    public Titik3D()
    {
        z = 0;
    }

    public int getZ()
    {
        return z;
    }
}
```

Keterkaitan antara kelas Titik dan Titik3D adalah kelas Titik3D merupakan kelas turunan dari kelas Titik. Dalam hal ini kelas Titik disebut dengan kelas dasar atau *super class* atau *base class* sedangkan kelas Titik3D disebut sebagai kelas turunan atau *derived class* atau *subclass* .

Pada contoh di atas, ketika kelas Titik3D dibuat objeknya maka objek tersebut dapat menggunakan metode yang ada pada kelas Titik walau pada kode programnya metode itu tidak dituliskan, misalkan sebagai berikut:

```
Titik3D p = new Titik3D();

Integer x = p.getX();

Integer y = p.getY();

Integer z = p.getZ();
```

Keuntungan dari pewarisan adalah tidak perlu mengotak atik kode kelas yang membutuhkan tambahan atribut atau metode saja, karena tinggal membuat kelas turunannya tanpa harus mengubah kode kelas dasarnya. Kelas dasar akan mewariskan semua atribut dan kodenya kecuali konstruktor dan destruktur yang memiliki izin akses *public* dan *protected* ke kelas turunannya dengan izin akses yang sama dengan pada kelas dasar.

Ketika sebuah kelas turunan dibuat objeknya saat eksekusi, maka secara implisit konstruktor kelas dasar dipanggil terlebih dahulu baru kemudian konstruktor kelas turunan dijalankan. Begitu juga saat objek dimusnahkan maka secara destruktur kelas turunan akan dijalankan baru kemudian destruktur kelas dasar dijalankan.

## 4.12. *Abstract*

Pada bahasa pemrograman Java juga ada sebuah kata kunci *abstract* yang dapat digunakan pada sebuah metode, namun jika digunakan pada sebuah metode, maka metode tersebut harus berada di dalam sebuah kelas yang juga menggunakan kata kunci *abstract*. Metode *abstract* tidak boleh memiliki badan program, badan program metode ini dapat diimplementasikan pada kelas turunannya.

Fungsi dari kelas atau metode *abstract* pada bahasa pemrograman Java adalah menyediakan sebuah abstraksi kelas atau metode sehingga dapat dilihat metode apa saja yang ada di dalam kelas tanpa harus melihat isi badan program dari metode-metode itu. Prinsipnya sama dengan fungsi sebuah daftar isi pada sebuah buku, dengan melihat daftar isi bisa diketahui isi sebuah buku tanpa harus membaca semua isi buku terlebih dahulu.

## 4.13. *Package*

*Package* adalah sebuah kontainer atau kemasan yang dapat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam *package* yang berbeda. Sebuah *package* pada Java dapat digunakan oleh *package* yang lain ataupun kelas-kelas di luar *Package*. Jika dalam bahasa pemrograman Java terdapat kode import `example.animal.Mamalia`; maka program tersebut memakai kelas `Mamalia` yang ada pada *package* `example.animal`. Jika terdapat kode import `example.animal.*`; maka program tersebut memakai semua kelas yang ada pada *package* `example.animal`.

*Package* pada bahasa pemrograman Java dinyatakan dengan kode: `package nama_package`;

Misalnya: `package example.animal`;

yang ditulis pada bagian atas kode program kelas anggota *package*. Misal sebuah kelas dengan nama `Mamalia` ada di dalam *package* dengan nama `example.animal` maka file yang menyimpan kode program kelas `Mamalia` dimasukkan dalam direktori. Contoh *package* yang ada dalam library java antara lain `java.util`, `javax.swing`, dll.

## 4.14. *Interface*

*Interface* atau antar muka pada bahasa pemrograman Java sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah *interface* dapat diimplementasikan oleh kelas lain. Sebuah kelas dapat mengimplementasikan lebih dari satu *interface* bahwa kelas ini akan mendeklarasikan metode pada *interface* yang dibutuhkan kelas itu sekaligus mendefinisikan isinya pada kode

program kelas itu. Metode pada *interface* yang diimplementasikan pada suatu kelas harus sama persis dengan yang ada pada *interface*. Misalnya pada *interface* terdapat deklarasi `void printAnimal()`; maka pada kelas yang mengimplementasikan metode itu harus ditulis sama yaitu:

```
void printAnimal() {  
.....  
}
```

Sebuah *interface* dideklarasikan dengan kode:

```
interface nama_antarmuka{  
    metode_1  
    metode_2  
    .....  
    metode_n  
}
```

misalnya:

```
interface Animal{  
    void printAnimal();  
}
```

Adapun deklarasi kelas yang mengimplementasikan interface sebagai berikut:

```
class nama_kelas implements interface_1, interface_2, ...,  
    interface_n{  
    metode_1  
    metode_2  
    .....  
    metode_n  
}  
}
```

misalnya:

```
class Mamalia implements Animal{  
    Mamalia () {  
    }  
  
    void printAnimal() {  
        system.out.println("printAnimal dalam kelas Mamalia");  
    }  
}
```

## Rangkuman

Java adalah sebuah teknologi yang diperkenalkan oleh Sun Microsystems pada pertengahan tahun 1990. Menurut definisi dari Sun, Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan. Teknologi Java memiliki tiga komponen penting, yaitu: *Programming-*

*language specification, Application-programming interface, Application-programming interface.*

Java2 adalah generasi kedua dari Java *platform* (generasi awalnya adalah Java Development Kit). Java berdiri di atas sebuah mesin interpreter yang diberi nama JVM. JVM inilah yang akan membaca *bytecode* dalam file *.class* dari suatu program sebagai representasi langsung program yang berisi bahasa mesin. Oleh karena itu, bahasa Java disebut sebagai bahasa pemrograman yang *portable* karena dapat dijalankan pada berbagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM.

## Rujukan

- [Silberschatz2005] Avi Silberschatz, Peter Galvin, dan Rag Gagne. 2005. *Operating Systems Concepts*. Seventh Edition. John Wiley & Sons.
- [KuliahDPBO] Yova Ruldeviyani. 2007. *Unit 1 Introduction to Object*. Fakultas Ilmu Komputer Universitas Indonesia.
- [WEBWiki2008] Wikipedia, The Free Encyclopedia. 2008. *Solaris (operating system)* [http://en.wikipedia.org/wiki/Solaris\\_%28operating\\_system%29](http://en.wikipedia.org/wiki/Solaris_%28operating_system%29). Diakses 10 April 2008.