Pemrograman Berorientasi Objek

LABORATORIUM SISTEM INFORMASI

2008

KATA PENGANTAR

Java adalah salah satu bahasa pemrograman berorientasi objek (OOP – *Object Oriented Programming*). Paradigma OOP menyelesaikan masalah dengan mempresentasikan masalah ke model objek. Java memiliki keutamaan dibanding bahasa pemrograman lain yaitu : *Cross Platform* dengan adanya *Java Virtual Machine* (JVM), pengembangannya didukung oleh programmer secara luas dan *Automatic Garbage Collection* yang membebaskan programmer dari tugas manajemen memori.

Melalui praktikum Java, diharapkan mahasiswa dapat merasakan kemudahan dan kelebihan dalam membuat suatu program dengan bahasa pemrograman berorientasi objek. Software Java yang digunakan adalah Java SDK 1.4.2 yang berjalan dalam sistem operasi linux.

Modul java ini berisi tentang pengertian bahasa Java, kemudian diawali mulai dari awal penulisan program, fungsi – fungsi dan metode yang terdapat dalam bahasa Java hingga dihasilkan output dari program tersebut. Di awal modul juga diberikan tips-tips dalam pembuatan program menggunakan bahasa Java.

DAFTAR ISI

PENDAHULUAN

SEJARAH JAVA

Proyek Java dimulai pada tahun 1991, ketika sejumlah insinyur perusahaan Sun yang dimotori oleh ***James Gosling*** mempunyai keinginan untuk mendesain sebuah bahasa komputer kecil yang dapat dipergunakan untuk peralatan konsumen seperti kotak tombol saluran TV. Proyek ini kemudian diberi nama sandi ***Green***.

Keharusan untuk membuat bahasa yang kecil , dan kode yang ketat mendorong mereka untuk menghidupkan kembali model yang pernah dicoba oleh bahasa UCSD Pascal, yaitu mendesain sebuah bahasa yang portable yang menghasilkan kode intermediate. Kode intermediate ini kemudian dapat digunakan pada banyak komputer yang interpreternya telah disesuaikan.

Karena orang-orang Sun memiliki latar belakang sebagai pemakai unix sehingga mereka lebih menggunakan C++ sebagai basis bahasa pemrograman mereka, maka mereka secara khusus mengembangkan bahasa yang berorientasi objek bukan berorientasi prosedur. Seperti yang dikatakan Gosling ”*Secara keseluruhan, bahasa hanyalah sarana, bukan merupakan tujuan akhir”.* Dan Gosling memutuskan menyebut bahasanya dengan nama “Oak” (diambil dari nama pohon yang tumbuh tepat diluar jendela kantornya di Sun), tetapi kemudian nama Oak diubah menjadi java, karena nama Oak merupakan nama bahasa komputer yang sudah ada sebelumnya.

Pada tahun 1994 sebagian besar orang menggunakan mosaic, browser web yang tidak diperdagangkan yang berasal dari pusat *Supercomputing* Universitas Illinois pada tahun 1993.(Mosaic sebagian ditulis oleh Marc Andreessen dengan bayaran $6.85 per jam, sebagai mahasiswa yang melakukan studi praktek. Di kemudian hari ia meraih ketenaran sebagai salah seorang pendiri dan pemimpin teknologi di netscape)

Browser yang sesungguhnya dibangun oleh **Patrick Naughton** dan **Jonathan Payne** dan berkembang ke dalam browser HotJava yang kita miliki saat ini. Browser HotJava ditulis dalam Java untuk menunjukkan kemampuan Java. Tetapi para pembuat juga memiliki ide tentang suatu kekuatan yang saat ini disebut dengan applet, sehingga mereka membuat browser yang mampu penerjemahkan kode byte tingkat menengah. "Teknologi yang Terbukti" ini diperlihatkan pada SunWorld '95 pada tanggal 23 mei 1995, yang mengilhami keranjingan terhadap Java terus berlanjut.

Kriteria "Kertas Putih" Java

Penulis Java telah menulis pengaruh "Kertas Putih" yang menjelaskan tujuan rancangan dan keunggulannya. Kertas mereka disusun lewat 11 kriteria berikut :

Sederhana (Simple)

Syntax untuk Java seperti syntax pada C++ tetapi syntax Java tidak memerlukan header file, pointer arithmatic (atau bahkan pointer syntax), struktur union, operator overloading, class virtual base, dan yang lainnya. Jika anda mengenal C++ dengan baik, maka anda dapat berpindah ke syntax Java dengan mudah tetapi jika tidak, anda pasti tidak berpendapat bahwa Java sederhana.

Berorientasi Objek (Object Oriented)

Rancangan berorientasi objek merupakan suatu teknik yang memusatkan rancangan pada data (objek) dan interface. Fasilitas pemrograman berorientasi objek pada Java pada dasarnya adalah sama dengan C++. Feature pemrograman berorientasi objek pada Java benar-benar sebanding dengan C++, perbedaan utama antara Java dengan C++ terletak pada penurunanberganda (multiple inheritance), untuk ini Java memiliki cara penyelesaian yang lebih baik.

Terdistribusi (Distributed)

Java memiliki library rutin yang luas untuk dirangkai pada protokol TCP/IP sepetrti HTTP dan FTP dengan mudah. Aplikasi Java dapat membuka dan mengakses objek untuk segala macam NET lewat URL sama mudahnya seperti yang biasa dilakukan seorang programmer ketika mengakses file sistem secara lokal.

Kuat (Robust)

Java dimaksudkan untuk membuat suatu program yang benar-benar dapat dipercaya dalam berbagai hal. Java banyak menekankan pada pengecekan awal untuk kemungkinan terjadinya masalah, pengecekan pada saat run0time dan mengurangi kemungkinan timbulnya kesalahan (error). Perbedaan utama antara Java dan C++ adalah Java memiliki sebuah model pointer yang mengurangi kemungkinan penimpaan (overwriting) pada memory dan kerusakan data (data corrupt).

Aman (Secure)

Java dimaksudkan untuk digunakan pada jaringan terdistribusi. Sebelum sampai pada bagian tersebut, penekanan terutama ditujukan pada masalah keamanan. Java memungkinkan penyusunan program yang bebas virus, sistem yang bebas dari kerusakan.

Netral Arsitektur (Architecture Neutral)

Kompiler membangkitkan sebuah format file dengan objek arsitektur syaraf, program yang di kompile dapat dijalankan pada banyak prosesor, disini diberikan sistem run time dari Java. Kompiler Java melakukannya dengan membangkitkan instruksi-instruksi kode byte yang tidak dapat dilakukan oleh arsitektur komputer tertentu. Dan yang lebih baiik Java dirancang untuk mempermudah penterjemahan pada banyak

komputer dengan mudah dan diterjemahkan pada komputer asal pada saat run-time.

Portabel (Portable)

Tidak seperti pada C dan C++, di Java terdapat ketergantungan pada saat implementasi (implement dependent). ukuran dari tipe data primitif ditentukan, sebagaimana kelakuan aritmatik padanya. Librari atau pustaka merupakan bagian dari sistem yang mendefinisikan interface yang portabel.

Interpreter

Interpreter Java dapat meng-eksekusi kode byte Java secara langsung pada komputer-komputer yang memiliki interpreter. Dan karena proses linking dalam Java merupakan proses yang kenaikannya tahap demi tahapdan berbobot ringan, maka proses pengembangan dapat menjadi lebih cepat dan masih dalam penelitian.

Kinerja Yang Tinggi (High Performance)

Meskipun kinerja kode byte yang di interpretasi biasanya lebih dari memadai, tetapi masih terdapat situasi yang memerlukan kinerja yang lebih tinggi. Kode byte dapat diterjemahkan (pada saat run-time) de dalam kode mesin untuk CPU tertentu dimana aplikasi sedang berjalan.

Multithreaded

Multithreading adalah kemampuan sebuah program untuk melakukan lebih dari satu pekerjaan sekaligus. Keuntunga dari multithreading adalah sifat respons yang interaktif dan real-time.

Dinamis

Dalam sejumlah hal, Java merupakan bahasa pemrograman yang lebih dinamis dibandingkan dengan C atau C++. Java dirancang untuk

beradaptasi dengan lingkungan yang terus berkembang. Librari dapat dengan mudah menambah metode dan variabel contoh yang baru tanpa banyak mempengaruhi klien. Informasi tipr run-time dalam Java adalah langsung (straigtforward).

- **EDISI JAVA**

  Java adalah bahasa yang dapat di jalankan dimanapun dan disembarang platform apapun, diberagam lingkungan : internet, intranet, consumer Electronic products dan computer Applications.

  The Java 2 platform tersedia dalam 3 edisi untuk keperluan berbeda. Untuk beragam aplikasi yang dibuat dengan bahasa java, java dipaketkan dalan edisi2 berikut :

  1. **Java 2 Standard Edition (J2SE)**
  2. **Java 2 Enterprise Edition (J2EE)**
  3. **Java 2 Micro Edition (J2ME)**

  Masing – masing edisi berisi java 2 Software Development Kit (J2SDK) untuk mengembangkan aplikasi dan java 2 Runtime Environment (J2RE) untuk menjalankan aplikasi.

- **Kelebihan Java dibandingkan dengan C++**
  - Pembuat program java telah merancang java untuk menghilangkan pengalokasian dan dealokasi memori secara manual, karena java memiliki Garbage Collection.
  - Diperkenalkannya deklarasi array yang sebenarnya dan menghilangkan aritmatika pointer. Hal ini yang sering menyebabkan memori overwrite.
  - Di hilangkannya multiple inheritance, mereka menggantinya dengan interface.

KONSEP OBJEK ORIENTASI PROGRAM

1

Obyektif :

1. Mengerti maksud inheritance
2. Mengerti dan memahami encapsulation
3. Mengerti dan dapat menjelaskan mengenai polymorphism
4. Dapat membuat program paling sederhana dari java

Pengenalan Java

Apa itu Java ?

Java adalah salah satu bahasa pemrograman berorientasi objek **(OOP-Object Oriented Programming)**. Paradigma OOP menyelesaikan masalah dengan merepresentasikan masalah ke model objek.

Keutamaan Java disbanding bahasa pemrograman lain:

- *Cross platform*, dengan adanya Java Virtual Machine(JVM)
- Pengembangannya didukung oleh programmer secara luas
- *Automatic Garbage Collection*, membebaskan programmer dari tugas manajemen memori

Pemrograman Berorientasi Obyek (OOP)

Pemisalan Objek dalam OOP

Objek-objek dalam dunia nyata, mempunyai 2 karakteristik khusus : Status dan Perilaku. Contohnya, sepeda punya status(jumlah gir, jumlah pedal, dua buah ban) dan perilaku(mengerem, mempercepat, ubah gir).

Bahasa yang berorientasi pada objek pun mempunyai karakteristik yang sama dengan objek-objek di dunia nyata. Yaitu status yang dalam bahasa

pemrograman biasanya disimpan sebagai *Variabel* dan perilaku yang diimplementasikan sebagai *Method*.

Karakteristik OOP

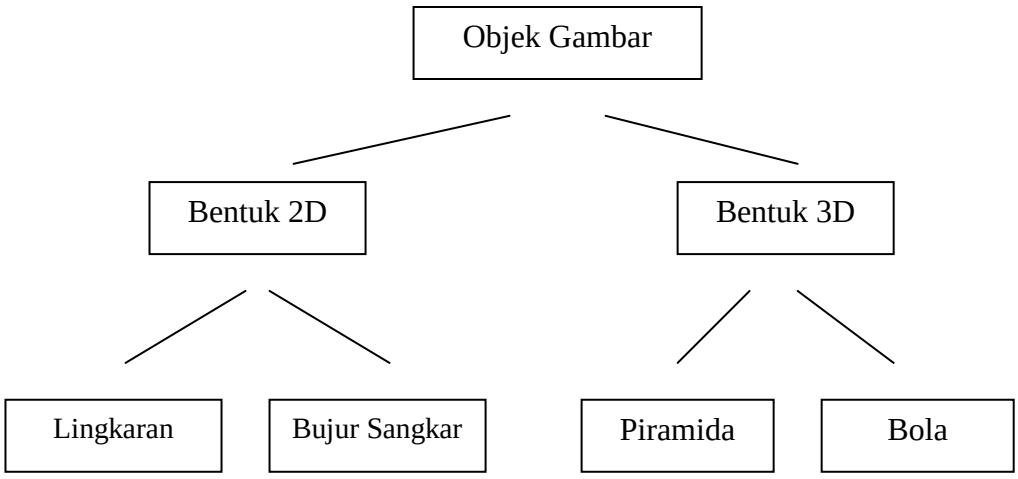
1. **Enkapsulasi(Pembungkusan)**

   Enkapsulasi adalah pelindung program dan data yang sedang diolah. Enkapsulasi mendefinisikan perilaku dan melindungi program dan data agar tidak diakses secara sembarangan oleh program lain.

   Dalam Java, dasar enkapsulasi adalah *class*. Anda membuat suatu *class* yang menyatakan bahwa variable atau method sebuah *class* tidak dapat diakses oleh *class* lain dengan menjadikan *class* tersebut private, atau menjadikan *class* tersebut protected – yaitu hanya bisa diakses oleh turunannya, atau menjadikan *class* tersebut public – yaitu bisa diakses oleh sembarang *class*.

2. **Inheritansi**

   Objek-objek yang berada di sekitar kita adalah objek-objek yang saling terhubung secara hirarkis. Misalnya :



   Lingkaran dan Bujur Sangkar adalah turunan dari bentuk 2D dan Bentuk 2D adalah turunan dari Objek Gambar

Lingkaran dan Bujur Sangkar mewarisi(inherit) sifat-sifat dari bentuk 2D, juga mewarisi sifat-sifat dari objek gambar

Lingkaran dan Bujur Sangkar dapat dikatakan subclass dari bentuk 2D. Bentuk 3D adalah superclass dari Bola dan Piramida, dan seterusnya.

3. **Polimorfisme**

   Walaupun Lingkaran dan Bujur Sangkat sama-sama turunan dari Bentuk 2D, tetapi cara menubah ukuran masing-masing berbeda, untuk lingkaran anda harus merubah besar jari-jarinya, sedang untuk bujur sangkar anda harus mengubah panjang sisinya.

Dalam Java implementasi, method suatu parent-class dapat diubah oleh sub-class, hal ini dikenal dengan overriding method. Deklarasi method sama tetapi implementasi atau definisinya berbeda(Method atau perilaku yang sama tapi implementasinya/caranya yang berbeda-beda inilah yang disebut dengan Polimorfisme).

Menulis Program Java

Aturan penulisan program di Java

- Java adalah turunan dari C, sehingga Java memiliki sifat C yaitu *Case sensitive*, yaitu membedakan antara huruf besar dan kecil
- Dalam sebuah file program di Java, hanya diijinkan memiliki 1 buah class yang bersifat *public*
- Dalam sebuah file program Java, hanya ada satu method main(method yang pertama kali dibaca oleh interpreter Java)
- Nama sebuah file program Java harus sama dengan nama class yang memiliki method main() di dalam tubuhnya. Perhatikan bahwa tulisan nama file dengan nama class (huruf besar maupun kecilnya) haruslah persis sama. .

Berikut adalah contoh membuat program dengan menggunakan Java. Pada program akan ditampilkan tulisan “Hello World !“

```
//Nama File Hello.java
class Hello
{
    public static void main(String args[])
    {
        System.out.println(“Hello World !”);
    }
}
```

Langkah selanjutnya :

1. Simpan dengan nama: Hello.java
2. compile Hello.java : javac Hello.java
3. hasilnya akan menghasilkan : Hello.class
4. jalankan Hello.class: java Hello.class atau java Hello
5. akan keluar hasil :

```
Hello world!
```

Programming Tip :

Dalam penulisan bahasa program, disarankan :

1. Huruf depan dari sebuah class atau method menggunakan huruf besar
2. Menulis Komentar pada sebuah class atau method untuk memudahkan debug(pencarian kesalahan). Serta mempermudah orang lain membaca program kita. Ingat !!! Dalam dunia nyata, programmer bekerja secara team, jadi usahakan partner team mengerti apa yang kita buat dengan memberikan komentar(tentang pembuatan komentar akan dibahas dibawah)
3. Membuat indentasi(jarak antara induk perintah dan anak perintah). Identasi sebisa mungkin dibuat standard, semisal pada contoh diatas, jarak antara tulisan “class Hello” sebagai induk perintah dengan tulisan

“public…..” sebagai anak perintah adalah 5 spasi. Sekali lagi, ini untuk mempermudah dalam pengertian program.

OPERATOR-OPERATOR DALAM JAVA

2

Obyektif :

1. Memahami tentang operator-operator (aritmatic, logical, relational, assigment, bitwise)
2. Dapat membuat program sederhana dengan menggunakan operator-operator

---

STATEMENT DAN IDENTIFIER

1. Statement

Bentuk statement atau pernyataan dalam satu program di Java adalah sebagai berikut :

```
Int i=1;
String teman = “Iman Rochdilianto”;
import java.awt.Font;
System.out.println(“Selamat Datang “ + teman +  “di Praktikum SBP”);
pegawai.tetap=true;
        total= a + b + c + d + e;
```

Setiap statement selalu diakhiri dengan titik koma (;)

Blok adalah 2 tanda kurung kurawal ({}) yang menyatukan statemen

```
{
  x = x + 1;
  y = y * 3;
}
```

Java memperbolehkan spasi dalam jumlah berapa saja (Spasi, tab, baris baru)

```
class Hello
{
        public static void main(String args[])
        {
                System.out.println(“Hello World”);
        }
    }
```

bisa ditulis dalam bentuk seperti dibawah ini :

```
class Hello ( public static void main(String args[]) {
System.out.println(“Hello World!”); } }
```

2. **Identifier**

   Dalam Java, identifier adalah nama yang diberikan untuk variable, class, atau method. Identifier boleh dimulai dengan huruf, underscore(_) atau tanda dollat($).

   Identifier adalah *case sensitive*(membedakan huruf besar/kecil) dan tak ada batas maksimum.

   Contoh :

   username
   user_name
   _sys_var1
   $change

Variabel dan Tipe Data

1. **Variabel**

   Variabel adalah suatu item dari data yang diberi nama identifikasi(identifier), variable dapat diartikan lokasi di dalam memori yang mana suatu nilai(value) dapat disimpan.

2. Tipe Data

Java membagi tipe data menjadi 2 bagian :

(1) Tipe data primitive

Keyword	Size	Range
		Bilangan Integer
Byte	8 bits	-128 s/d 127
Short	16 bits	-32768 s/d 32767
Int	32 bits	-2.147.483.648 s/d 2.147.483.647
Long	64 bits	9223372036854775808 s/d 9223372036854775808
		Bilangan Real
Float	32 bits	Single Precision
Double	64 bits	Double Precision
		Tipe Data Lain
Char	16 bits	Single Characte
Boolean	True / false	Nilai Boolean

Contoh cara pendeklarasian dan inisialisasi tipe data primitive sebagai berikut :

```
char ch;              // deklarasi variable
ch = “R’;             // inisialisasi variable
char ch1= “S”;        // delarasi dan inisialisasi variable
int x,y,z;            // deklarasi 3 variabel integer
boolean tetap= true;
```

(2) Tipe data reference

Reference adalah pointer ke tipe data atau penyimpan alamat data.

Terdapat tiga data reference yaitu : *array, class,* dan *interface* (mengenai tipe data reference akan diuraikan dalam bab selanjutnya)

Komentar

Berikut cara menyisipkan komentar pada program

```
class Hello
{       // kalimat ini adalah komentar
        // yang tak akan dieksekusi
        public static void main(String args[])
        {
                System.out.println(“Hello World!”);
        }
        /*      Kalimat ini adalah komentar
                Yang tidak akan dieksekusi
        */
}
```

Literal

Karakter literal adalah karakter yang ditulis diantara kutip tunggal : ‘r’,’#’,’14’ dan sebagainya. Karakter ini disimpan sebagai 16 bit Unicode Characters. Berikut daftar special kode yang merepresentasikan karakter-karakter yang tidak dapat di print(non-printable characters)

Escape	Meaning
\n	Newline
\t	Tab
\b	Backspace
\r	Carriage Return
\f	Formfeed
\\	Backslash
\’	Single Quote
\”	Double Quote
\ddd	Octal

\xdd	Hexadecimal
\udddd	Unicode Character

Contoh :

"Trade Mark dari Java \u212"

Hasil output diatas adalah :

Trade Mark dari Java ™

Operator dan Ekspresi

Ekspresi : adalah statement yang mengembalikan suatu nilai

Operator : suatu symbol yang biasanya digunakan dalam ekspresi

Operator Aritmatika

Operator	Meaning	Example
+	Addition	3 + 4
-	Substraction	5 – 7
*****	Multiplication	5 * 5
/	Division	14 / 7
%	Modulus	20 % 7

Contoh :

```
// Nama File Aritmatika.java
class Aritmatika {
        public static void main(String args[])        {
                short x = 10'
                int y = 4;
                float a = 12.5f;
                float b = 7f;
```

```
        System.out.println("X = " + x + ", Y = " +y);
        System.out.println("X +Y = " + (x +y));
        System.out.println("X -Y = " +(x-y));
        System.out.println("X / Y = " +(x/y));
        System.out.println("X % Y = " +(x%y));

        System.out.println("A = " + a + ", B = " +b);
        System.out.println("A / B = " + (a / b));
        }
}
```

Lebih jauh dengan Assignment

Variabel assignment adalah suatu bentuk ekspresi :

```
x = y = z = 0;
```

pada contoh diatas variable x,y,z bernilai 0.

Assignment Operator

Ekspression	Meaning
x += y	x = x + y
x -= y	X = x – y
x *= y	x = x * y
x /= y	x = x / y

Operator Perbandingan

Java mempunyai beberapa ekspresi untuk menguji hasil suatu perbandingan :

Operator	Meaning	Example
==	Equal	x== 3
!=	Not Equal	x != 3
<	Less Than	x < 3

>	Greater Than	x > 3
<=	Less Than Or Equal To	x < = 3
>=	Greater Than Or Equal To	x > = 3

OBJECT , CLASS DAN METHOD

3

Obyektif :

1. Mengetahui pengertian dari objek & class
2. Dapat membuat program sederhana dari java dengan menggunakna objek dan class

---

Object dan Class

class

Dalam dunia nyata, kita sering berinteraksi dengan banyak object. Kita tinggal di rumah, rumah adalah suatu object, dalam terminology OOP rumah kita adalah instance dari suatu class rumah.

Misal kita tinggal dalam suatu komplek perumahan, sebelum membangun rumah, developer akan berpanduan pada rancang bangun rumah (blue print) yang telah dibuat seorang arsitek. Blue print dari rumah adalah class, sedang rumah yang kita tinggal (rumah-rumah dalam komplek) disebut instance.

Manusia adalah sebuah class ; anda, saya, kita adalah instance dari class manusia.

Object

Object adalah instance dari class. Jika class secara umum merepresentasikan (template) sebuah object, sebuah instance adalah representasi nyata dari class itu sendiri.

Bekerja dengan Object

Ketika anda membuat program dengan Java, anda akan mendefinisikan beberapa class, anda juga akan menggunakan class untuk membuat

suatu instance dan tentu saja akan bekerja dengan instance-instance tersebut.

Membuat Object

Untuk membuat object, kita menggunakan perintah *new* dengan sebuah nama class yang akan dibuat sebagai instance dari class tersebut.

```
String str = new String();
Random r = new Random();
Pegawai p2 = new PEgawai();
Date hari = new Date();
```

hari adalah object reference dari class Date yang akan digunakan untuk mengakses class Date.

Sedangkan operator *new* adalah operator yang akan menghasilkan hari sebagai reference ke instance dari class Date().

Contoh :

Kita akan menggunakan class Date untuk membuat suatu object Date.

```
import java.util.Date;
class CreateDates {
    public static void main(String args[]){
            Date d1,d2,d3;
            d1 = new Date();
            System.out.println("Hari 1 : " + d1);
        d2 = new Date(71,4,14,8,35);
            System.out.println("Hari 2 : " + d2);
        d3 = new Date("September 3 1976 2:25 PM");
            System.out.println("Hari 3 : " + d3);
    }
}
```

Ketika anda memanggil operator new terjadi beberapa hal :

1. Instance baru yang telah diberikan oleh class dicipakan

2. Memori dialokasikan untuk instance tersebut
3. Special Method didefinisikan pada class (Konstruktor)

Konstruktor : Suatu method tertentu untuk membuat dan menginsialisasi sebuah instance baru dari class. Konstruktor menginisialiasasi object-object baru dan variable-variabel. Pemberian nama method Konstruktor harus sama dengan nama classnya. (Penjelasan tentang Konstruktor akan dibahas dalam pertemuan berikutnya)

Akses dan Setting Class dan Variabel Instance

Akses Variable Instance

Untuk mengambil value dari suatu variable instance kita gunakan notasi titik(.)

Dengan notasi titik, sebuah instance atau variable class dibagi dua bagian. Object berada di kiri titik dan variable berada di kanan titik.

```
Pegawai.tugas;
```

Pegawai adalah object, tugas adalah variable. Misalkan tugas adalah object yang mempunyai variable instance sendiri yaitu status, penulisan dapat ditulis sebagai berikut

```
Pegawai.tugas.status;
```

Memberi Nilai Variabel

Untuk memberi nilai variable kita gunakan operator sama dengan(=) disebelah kanan ekspresi.

```
Pegawai.tugas.status = SELESAI; // SELESAI==true
```

Contoh :

```
//  Nama File : Testpoint.java
import java.awt.font;
```

```
class Testpoint {
        public static void main(String args[]) {
                Point poin = new Point(10,10);
                System.out.println(“X = “ + point.x);
                System.out.println(“Y = “ + point.y);
                System.out.println(“Setting X = 6 “);
                poin.x = 6;
                System.out.println(“Setting Y = 14“);
                poin.y = 14;
                System.out.println(“X = “ + point.x);
                System.out.println(“Y = “ + point.y);
        }
}
```

Memanggil Method

Untuk memanggil method didalam object, sama seperti memanggil variable instance; yaitu dengan dengan menggunakan notasi titik(.)

Object berada disebelah kiri titik, dan method beserta argumen-argumen berada di kanan titik.

```
ObjectSatu.methodDua(arg1, arg2, arg3);
```

Method tanpa argument :

```
ObjectSatu.methodNoArg();
```

Jika method yang dipanggil mempunyai object yang mempunyai method tersendiri.

```
ObjectSatu.GetObjectLain().getNama();
```

Method dengan kombinasi memanggil variable instance

```
Pegawai.golongan.gaji(arg1, arg2);
```

Contoh :

```
//Nama File : TestString.java
classTestString {
        public static void main(String args[]) {
        String str=”Awalilah segala sesuatu pekerjaan dengan
                Bismillah”;

                System.out.println(“Kalimat bijak : “ +str);
                System.out.println(“Panjang Kalimat  : “ +str.length());
                System.out.println(“Character pada posisi 4 adalah : “
                        +str.charAt(4));
        }
}
```

Object Reference

Ketika bekerja dengan object-object, salah satu hal yang penting untuk dimengerti adalah bagaimana menggunakan reference ke suatu object.

Ketika kita meng-assign suatu object ke variable, atau menjadikan object-object sebagai argument pada suatu method, sesungguhnya kita telah membuat reference ke object-object tersebut, bukan object atau duplikasi(copy) dari object yang membuat suatu reference

Contoh berikut akan membuat kita jelas :

```
// Nama file : ReferencesTest.java
import java.awt.font;
class ReferenceTest {
        public static void main(String args[]) {
```

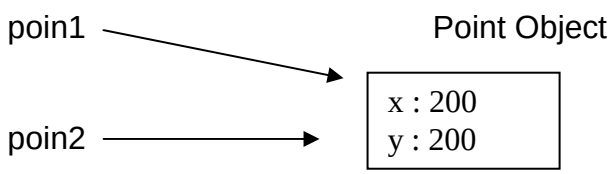
```
        Point poin1,poin2;
        poin1 = new Point(100,100);
        poin2 = poin1;
        poin1.x = 200;
        poin2.y = 200;
        System.out.println("Point 1 : "+poin1.x+","+poin1.y);
        System.out.println("Point 1 : "+poin1.x+","+poin1.y);
    }
}
```

Dalam program diatas, kita mendeklarasikan dua variable bertipe Point, dan meng-assign suatu Point baru ke poin1. Kemudian meng-assign poin2 dengan nilai dari poin1.

Output yang terjadi adalah :

Point 1 : 200,200
Point 2 : 200,200

Terlihat poin2 juga berubah. Ketika kita meng-assign suatu nilai dari poin1 ke poin2, sesungguhnya kita menciptakan sebuah reference dari point2 menunjuk ke suatu object yang sama dengan poin1.



Mendeklarasikan Class dan Variabel

Class adalah kumpulan kode atau cetak biru(blue print) dari suatu object. Didalam cetak biru menerangkan sifat dari objek dan identitas suatu variable.

Sintax untuk mendeklarasikan class :

```
['public'] [('abstract' | 'final')] class nama_class
{
        // sifat dari object dan identitas suatu variable dideklarasikan diantara {}
}
```

Menggunakan keyword public berarti class tersebut bisa di akses oleh class-class di seluruh package. Perlu diingat jangan menggunakan public jika class dibuat hanya di akses oleh class-class dalam satu package dengan class itu sendiri.

Gunakan keyword abstract untuk mendefinisikan suatu class abstract dimana object tidak bisa diciptakan dari class abstract, kelas abstract dibuat untuk diturunkan(di subclass) bukan untuk diinstansiasi langsung.

Gunakan keyword final untuk mendefinisikan suatu class yang tidak dapat diturunkan.

Penamaan class biasanya menggunakan huruf capital untuk karakter pertamanya.

Contoh :

```
class Mhs {
}
```

Jika class adalah sub class dari class lain gunakan keyword extends.

```
class Mhs extends Mahasiswa {
```

```
        }
```

Dapat diartikan Mhs adalah subclass dari Mahasiswa.

Deklarasi Variabel

Sintax Deklarasi variable :

```
[(public | private | protected)]
    [(final | volatile)]
       [static][transient]
          Tipe_data Nama_variabel [=ekspresi];
```

Contoh :

```
class Mahasiswa {
        String npm;
        int nilai;
}
```

Akses variable

Untuk akses dan scope variable digunakan keyword : public, private atau protected.

1. **Tanpa keyword**

   Contoh dibawah ini bila tidak menggunakan keyword :

   ```
   class MyClass
   {
       int nama;
   }
   ```

   Berarti : Hanya kode-kode yang terdapat dalam MyClass dan class-class lain yang dideklarasikan dalam package dimana MyClass dideklarasikan yang dapat mengakses variable nama.

2. **private**

   ```
   class Pegawai
   {
       private double gaji
   ```

```
}
```

Berarti : Hanya kode-kode yang terdapat dalam class Pegawai yang dapat mengakses variable gaji.

3. **public**

```
public class Pegawai
{
    public String nama;
}
```

Berarti : Kode-kode yang terdapat didalam class Pegawai dan class yang terdapat di dalam package lain dapat mengakses variable nama(class Pegawai harus dideklarasikan public juga agar dapat diakses class-class dalam package lain.

4. **protected**

```
public class Pegawai
{
    protected String nama;
}
```

Berarti : Hanya kode-kode yang terdapat dalam class Pegawai dan class-class lain dalam satu package yang sama dengan class Pegawai dan seluruh sub class dari class Pegawai(yang dideklarasikan dalam package lain) dapat mengakses variable nama.

METHOD

Metode menentukan perilaku objek,yakni apa yang terjadiketika objek itu dibuat serta berbagai operasi yang dapat dilakukan objek sepanjang hidupnya.

Metode memiliki 4 (empat) bagian dasar :

- Nama metode
- Tipe Objek atau tipe primitive yang dikembalikan metode.
- Daftar parameter.

- Badan atau isi metode.

Tiga bagian pertama dari definisi metode membentuk apa yang disebut sebagai penanda (signature) metode dan mengindikasikan informasi penting tentang metode itu sendiri. Dengan kata lain, nama metode tersebut dari metode=-metode lain dalam program. Dalam java kita dapat memiliki metode-metode berbeda yang memiliki nama sama tetapi berbeda tipe kembalian atau daftar argumennya, sehingga bagian-bagian definisi metode ini menjadi penting. Ini disebut *overloading* metode.

Definisi dasar metode adalah sebagai berikut :

```
   Tipekembalian namametode (type1 arg1, type2 arg2, type3 arg3 ..)
{
…….
}
```

dalam contoh diatas tipe kembalian adalah tipe nilai yang dikembalikan oleh metode. Ini bias berupa salah satu tipe primitive,nama kelas, atau void bila metode tidak mengembalikan nilai sama sekali.

Contoh Program :

```
class KelasRentang {
int [] buatRentang(int lower, int upper) {
     int arr[] = new int [ {upper – lower ) + 1 ];
    for (int I = 0 ; i< arr.length;i++) {
         arr[i] = lower++;
      }
       return arr;
 }
public static void main(String [] args) {
int inilarik[];
KelasRentang iniRentang = new KelasRentang ();
Inilarik = iniRentang.buatRentang(5,20);
System.out.print(“Lariknya adalah : [ “ );
For (int i = 0; i < inilarik.length;i++) {
```

```
        System.out.print(inilarik[i] + " ");
    }
    System.out.println("] ");
    }
    }
```

Pada bagian definisi metode terkadang kita ingin merujuk objek saat ini (yakni objek dimana metode berada untuk pertama kalinya). Untuk itu digunakanlah kata kunci **this**. Kata kunci this dapat digunakan dimanapun objek saat ini berada pada notasi titik untuk merujuk variabel *instance* objek, sebagai argument ke metode, sebagai nilai kembalian untuk metode saat ini, san sebagainya :

```
t  = this.x      // x variabel instance objek
return this   // mengembalikan objek saat ini.
```

KONSTRUKTOR

Metode konstruktor digunakan untuk menginisialisasi objek baru ketika metode-metode itu dibuat. Tidak seperti metode biasa, kita tidak dapat memanggil metode konstruktor dengan memanggilnya langsung. Metode konstruktor dipanggil oleh java secara otamatis ketika kita membauat objek baru.

Jika kita menggunakan **new** untuk membuat objek baru, java melakukan 3(tiga) hal :

- Mengalokasikan memori untuk objek baru
- Menginisialisasi variabel *instance* objek tersebut, baik dengan nilai awal maupun dengan nilai default (0 untuk bilangan, null untuk objek, false untuk Boolean, '\0' untuk objek, false untuk Boolean, '\0' untuk karakter).
- Memanggil Metode konstruktor kelas tersebut (mungkin satu dari beberapa metode)

Dengan mendefinisikan metode konstruktor pada kelas yang kita buat, kita dapat mengatur nilai awal variabel instance, memanggil metode berdasar

variabel tersebut atau objek lain, atau menghitung property awal objek, kita juga dapat melakukan overloading konstruktor sebagaimana yang biasa kita lakukanh terhadap metode regular, juga membuat objek yang memiliki properti khusus berdarkan argumen yang kita berikan dalam ekspresi **new.**

Konstruktor mirip dengan metode regular, hanya saja ada dua perbedaan utama yaitu :

- Konstruktor selalu memiliki nama yang sama dengan class.
- Konstruktor tidak memiliki nilai kembalian.

Contoh Program :

```
class  Asisten {
     String nama ;
     int  umur;
     Asisten(String n, int u) {
       nama = n;
     umur = u;
    }
  void tampilAsisten () {
       System.out.print(“Hallo, namaku “ +  nama );
       System.out.println(“ Umurku “ + umur + “ tahun “);
    }
  public ststic void main(String [] args) {
     Asisten a;
     System.out.println(“ “);
     a = new Asisten(“Widy Marinto Jati “ ,20);
     a.tampilAsisten();
     System.out.println(“--------------------“ );
     a = new Asisten(“Iman R, ST “ ,22);
     a.tampilAsisten();
     System.out.println(“----------------------“);
   }
```

```
}
```

INHERITANCE, ARRAY DAN STRING

4

Obyektif :

1. Mengetahui dan memahami penurunan sifat dalam java
2. Mengetahui peggunaan array dan string
3. Dapat membuat program dengan menggunakan inheritance, array dan string dalam java secara sederhana

---

ARRAY

Array merupakan suatu struktur data yang sangat penting dalam suatu bahasa pemrograman . Suatu larik atau array merupakan suatu kumpulan data yang memiliki tipe yang sama. Misal suatu array bertipe string maka tidak boleh ada tipe lain didalamnya.

Ada 2 cara untuk mendeklarasikan array dalam bahasa java yaitu :

a. Dengan menggunakan operator **New :**

   Bentuk Umum : ***tipe-array nama-array[ ] = new tipe-array[ukuran array]***

   Contoh : int nama [] = new int[10]

b. Dengan memberikan nilai awal array dengan menggunakan “{“ dan “}”

   Contoh :

   String [ ] jurusan = {“ MI ”,” TI “,” TK “,” AKUNTANSI “ }

Untuk bentuk pertama ini array belum memiliki nilai awal sehingga sebelum digunakan kita harus memberikan nilai awal kepada array tersebut.

Contoh Program :

```
class  larik {
    String [] nama = {“JATI”,”SANTI”,”NISA”,”RINTO”};
    String [] kelas = new String[nama.length];
Void cetakNama() {
    Int I = 0;
   System.out.println(nama[i] + ” “ + kelas[i];
i++;
   System.out.println(nama[i] + ” “ + kelas[i];
i++;
   System.out.println(nama[i] + ” “ + kelas[i];
i++;
   System.out.println(nama[i] + ” “ + kelas[i];
 }
public static void main (String[] args) {
   larik a = new larik ();
   System.out.println(“ “);
   System.out.println(“------------------------“);
   a.cetakNama();
   System.out.println(------------------------“);
     a.kelas[0] = “3IA03”;
     a.kelas[1] = “4IA01”;
     a.kelas[2] = “2IA01”;
     a.kelas[0] = “3IA07”;
  a.cetakNama();
 System.out.println(“--------------------“);
 }
}
```

Untuk deklarasi array multi dimensi :

Type array nama array [] [] = new type array[ukuran-array] [ukuran array]

Contoh program :

```
import java.io.*;
class Array2D
{
 public static void main(String[]args) {
     DataInputStream entry = new DataInputStream(System.in);
     try {
         int[][]angka = new int[3][3];
         for(int i = 0;i<angka.length;i++) {
             for(int j = 0;j<angka[i].length;j++) {
                 System.out.print("Matrik [ " + (i+1)+ "] [ " + (j+1) + " ] = ");
                  angka[i][j] = Integer.parseInt(entry.readLine());
             }
         }
         System.out.println("Data array 2 Dimaensi : ");
         for(int i = 0; i<angka.length;i++) {
             for(int j = 0;j<angka[i].length;j++) {
                  System.out.print(angka[i][j]+ " ");
             }
             System.out.println();
         }
      }
      catch(Exception e) {
          System.out.println("Wah salah input tuh ");
      }
   }
}
```

INHERITANCE (PENURUNAN SIFAT)

Inheritance adalah pewarisan atribut-atribut dan method pada sebuah class yang diperoleh dari class yang telah terdefinisi tersebut.

Contoh Program :

```
// contoh inheritanace sederhana
// file disimpan dengan nama penurunansederhana.java
class A {
    int i;
    int j;
    void show_ij() {
        System.out.println("I dan j = " + I + "" + j);
    }
}
class B extends A {
    int K ;
    void show_k () {
        System.out.println("k = " +k);
    }
   void sum_all() {
       System.out.println("I + j + k = " + (i+j+k));
    }
}
class penurunansederhana {
    public static void main (String args[]) {
    A objekBapak = new A();
    B objekAnak = new B();
    objekBapak.i = 13;
    objekBapak.j = 17;
    System.out.println("Objek A -> objek superclass dari B : ")
```

```
        objekBapak.show_ij();
        objekAnak.i = 9;
       objekAnak.j = 10;
       objekAnak.k = 11;
       System.out.println("Objek A -> objek superclass dari B : ")
       objekAnak.show_ij();
       objekAnak.show_k();
       objekAnak.sum_all();
    }
  }
```

STRING

Java string merupakan salah satu kelas dasar yang disediakan oleh java untuk manipulasi karakter. Kelas string diguinakan untuk mendefinisikan string yang constant(tidak bias berubah).

Contoh Program :

```
      class  panjang_string {
          public static void main(String [] args) {
              String s1 = "Perkenalan ";
              String s2 = new String (" Pertama " );
               Int pjg;
               Pjg = s1.length;
               System.out.println("panjang String s1 = \" +s1+"\"= " + pjg );
               Pjg = s2.length();
               System.out.println("panjang String s2 = \" +s2+"\"= " + pjg );
          }
       }
```

KONTROL ALUR PROGRAM

5

Obyektif :

1. Mengetahui dan memahami tentang percabangan (seleksi)
2. Mengetahui dan memahami tentang perulangan (iterasi)
3. Dapat membuat program tentang control alur program

PERCABANGAN

- **If – Else**

  Bentuk if-else menyebabkan eksekusi dijalankan melalui sekumpulan keadaan boolean sehingga hanya bagian tertentu program yang dijalankan. Bentuk umum pernyataan if-else :

  if (boolean expression) statement 1; [else statement 2;]

  Klausa else bersifat optional, setiap statement dapat berupa satu statement tunggal atau dapat berupa satu blok statement yang ditandai dengan tanda {} (kurung kurawal). Boolean expression dapat berupa sembarang pernyataan boolean yang menghasilkan besaran boolean.

- **Break**

  Java tidak memiliki pernyataan goto. Penggunaan goto adalah untuk membuat percabangan secara sembarang yang membuat program sulit dimengerti dan mengurangi optimasi compiler tertentu. Pernyataan break pada Java dirancang untuk mengatasi semua kasus tersebut. Istilah break mengacu kepada proses memecahkan blok program. Proses tersebut memerintahkan runtime untuk menjalankan program dibelakang blok tertentu. Untuk dapat ditunjuk blok diberi

nama/label. Break juga dapat digunakan tanpa label untuk keluar dari suatu loop dan pernyataan switch. Penggunaan break menunjukkan bahwa kita akan keluar dari sutu blok program.

- **Switch**

  Pernyataan switch memberiikan suatu cara ubtuk mengirimkan bagian program berdasarkan nilai suatu variabel atau pernyataan tunggal. Bentuk umum pernyataan switch :

```
switch (expression)
{       case value1  :
                Statement;
                break;
        case value2  :
                Statement;
                break;
        case valueN  :
                Statement;
                break;
        default;
}
```

  Expression dapat menghasilkan suatu tipe sederhana, dan setiap value yang disebutkan pada pernyataan case harus berupa tipe yang cocok. Pernyataan switch bekerja dengan cara membandingkan nilai expression dengan setiap nilai pada pernyataan case. Jika ada yang cocok maka urutan program yang ada di pernyataan case tersebut akan dijalankan, jika tidak ada yang cocok, program akan menjalankan default

- **Return**

  Java menggunakan bentuk sub-routine yang disebut method untuk mengimplementasikan antarmuka prosedural ke class objek. Setiap

saat dalam method dapat digunakan pernyataan return yang menyebabkan eksekusi mencabang kembali ke pemanggil method.

PERULANGAN

Loop atau sering disebut juga sebagai *iterasi* adalah pengualangan suatu eksekusi dari suatu kode program. Pengulangan ini akan terus dilakukkan sampai sebuah kondisi dicapai atau perulangan tersebut telah diulang sebanyak n kali .

Didalam bahasa java terdapat beberapa macam perulangan yaitu :

a. While

Statemen while digunakan untuk mengeksekusi sebuah blok secara berulang selama memenuhi kondisi tertentu..

Bentuk Umum :

while(ekspresi) {

…… statemen …….. }

Contoh Program :

```
class ulang1 {
    public static void main (String []args ) {
        System.out.println("Masukkan angka kamu : ");
        char c = (char) System.in.read();
        while (c <> '7' ) {
            System.out.println("Please try again ! ");
            System.out.println("Masukkan angka kamu : ");
            char c = (char) System.in.read();
        }
        System.out.println("Anda Benar !!!!! ");
    }
}
```

b. Do….While

Sama halnya dengan *while*, statemen *do-while* digunakan untuk mengeksekusi sebuah blok secara berulang sampai tidak memenuhi kondisi tertentu. Pada penggunaan *while*,ekspresi diperiksa pada saat awa;,jadi kemungkinan blok *statemen* dalam *while* tidak pernah dieksekusi. Pada penggunaan *do-while*, ekspresi tidak diperiksa pada saat awal eksekusi, jadi minimal blok statemen do-while akan di eksekusi sekali.

Bentuk Umum :

do {

……. statemen…….

}while(ekspresi)

Contoh Program :

```
class DoWhile {
    public static void main (String args[]) {
        int n = 10 ;
       do {
          System.out.println("tick tick " + n);
           n--;
        } while(n > 0);
     }
  }
```

c. For

Statemen for digunakan untuk mengeksekusi sebuah blok secara berulang dalam sebuah range tertentu.

Bentuk Umum :

for(inisialisai;terminasi;increment){

…….. statemen ……. }

Contoh Program :

```
class ForTick {
    public static void main (String []args) {
        int n;
        for(n=10;n>0;n--)
        System.out.println(“tick tick “ + n);
    }
}
```

EXCEPTION HANDLING

6

Obyektif :

1. Mampu menangani eksepsi
2. Mengetahui dan memahami tentang multithreading
3. Dapat membuat program tentang exception handling

PENANGANAN EKSEPSI

Eksepsi adalah keadaan tidak normal yang muncul pada suatu bagian program pada saat dijalankan. Penanganan eksepsi pada java membawa pengelolaan kesalahan program saat dijalankan kedalam orientasi-objek. Eksepsi java adalah objek yang menjelaskan suatu keadaan eksepsi yang muncul pada suatu bagian program.

Saat suatu keadaan eksepsi muncul, suatu objek *exception* dibuat dan dimasukkan ke dalam method yang menyebabkan eksepsi. Method tersebut dapat dipilih untuk menangani eksepsi berdasarkan tipe tertentu. Method ini juga menjaga agar tidak keluar terlalu dini melalui suatu eksepsi, dan memiliki suatu blok program yang dijalankan tepat sebelum suatu eksepsi menyebabkan metodenya kembali ke pemanggil.

Eksepsi dapat muncul tidak beraturan dalam suatu method, atau dapat juga dibuat secara manual dan nantinya melaporkan sejumlah keadaan kesalahan ke method yang memanggil.

Dasar-dasar Penanganan Eksepsi

Penanganan eksepsi pada java diatur dengan lima kata kunci : *try, catch, throw, throws* dan *finally.* Pada dasarnya *try* digunakan untuk

mengeksekusi suatu bagian program, dan jika muncul kesalahan, sistem akan melakukan *throw* suatu eksepsi yang dapat anda *catch* berdasarkan tipe eksepsinya, atau yang anda berikan *finally* dengan penangan default. Berikut ini bentuk dasar bagian penanganan eksepsi :

```
try {
        // Block of Code
    }
catch (ExceptionType1 e) {
        // Exception Handler for ExceptionType1
}
catch (ExceptionType2 e) {
        // Exception Handler for ExceptionTYpe2
        throw (e); // re-throw the Exception…
}
finally {
}
```

Tipe Eksepsi

Dipuncak hirarki class eksepsi terdapat satu class yang disebut *throwable*. Class ini digunakan untuk merepresentasikan semua keadaan ekasepsi. Setiap *ExceptionType* pada bentuk umum diatas adalah subclass dari *throwable*.

Dua subclass langsung *throwable* didefinisikan untuk membagi class *throwable* menjadi dua cabang yang berbeda. Satu, class *Exception*, digunakan untuk keadaan eksepsi yang harus ditangkap oleh program yang kita buat. Sedangkan yang lain diharapkan dapat menangkap class yang kita subclasskan untuk menghasilkan keadaan eksepsi.

Cabang kedua *throwable* adalah class *error*, yang mendefinisikan keadaan yang tidak diharapkan untuk ditangkap dalam lingkungan normal.

Eksepsi yang Tidak Dapat Ditangkap

Obyek eksepsi secara otomatis dihasilkan oleh runtime java untuk menanggapi suatu keadaan eksepsi. Perhatikan contoh berikut :

```
class Exc0 {
    public static void main (Stinr args[]) {
        int d = 0;
        int a = 42 / d;
    }
}
```

Saat runtime java mencoba meng-eksekusi pembagian, akan terlihat bahwa pembaginya adalah nol, dan akan membentuk objek eksepsi baru yang menyebabkan program terhenti dan harus berurusan dengan keadaan kesalahan tersebut. Kita belum mengkodekan suatu penangan eksepsi, sehingga penanganan eksepsi default akan segera dijalankan. Keluaran dari program di atas :

```
java.lang.ArithmeticExpression : /by zero
at Exc0.main (Exc0.java:4)
```

Berikut adalah contoh lainnya dari eksepsi :

```
class Exc1 {
    static void subroutine() {
        int d = 0;
        int a = 42 / d;
    }
    public static void main (Stinr args[]) {
        Exc1.subroutine();
    }
}
```

Output-nya :

```
java.lang.ArithmeticException : / by zero
        at Exc1.subroutine(Exc1.java :4)
        at Exc1.main(Exc1.java : 7)
```

Try dan Catch

Kata kunci try digunakan untuk menentukan suatu blok program yang harus dijaga terhadap semua eksepsi, setelah blok try masukkan bagian catch, yang menentukan tipe eksepsi yang akan ditangkap. Perhatikan contoh berikut :

```
class Exc2 {
    public static void main (String args[]) {
        try {
            int d = 0;
            int a = 42 / d;
        }
        catch (ArithmeticException e) {
             System.out.println(“Division By Zero);
        }
    }
}
```

Throw

Pernyataan throw digunakan untuk secara eksplisit melemparkan suatu eksepsi. Pertama kita harus mendapatkan penanganan dalam suatu instance throwable, melalui suatu parameter kedalam bagian catch, atau dengan membuatnya menggunakan operator new. Bentuk umum pernyataan throw :

throw ThrowableInstance;

Aliran eksekusi akan segera terhenti setelah pernyataan throw, dan pernyataan selanjutnya tidak akan dicapai. Blok try terdekat akan diperiksa untuk melihat jika telah memiliki bagian catch yang cocok dengan tipe instance Throwable. Jika tidak ditemukan yang cocok, maka pengaturan dipindahkan ke pernyataan tersebut. Jika tidak, maka blok pernyataan try selanjutnya diperiksa, begitu seterusnya sampai penanganan eksepsi terluar menghentikan program dan mencetak penelusuran semua tumpukan sampai pernyataan throw. Contoh :

```
class throwDemo {
static void demoProc() {
        try {
                throw new NullPointerException("demo");
        }
        catch (NullPointerException e) {
                System.out.println("caught inside demoproc…");
                throw e;
        }
}
public static void main (String args[]) {
        try {
                demoproc();
        }
        catch (NullPointerException e) {
                System.out.println("recaugt : " + e);
        }
        }
}
```

Output :

```
caught inside demoproc
```

recaught : java.lang.NullPointerException : demo

Throws

Kata kunci throws digunakan untuk mengenali daftar eksepsi yang mungkin di-throw oleh suatu method. Jika tipe eksepsinya adalah error, atau RuntimeException, atau suatu subclassnya, aturan ini tidak berlaku, karena tidak diharapkan sebagai bagian normal dari kerja program.

Jika suatu method secara eksplisit men-throws suatu intans dari Exception atau subclassnya, diluar RuntimeException, kita harus mendeklarasikan tipenya dengan pernyataan throws. ini mendefinisikan ulang deklarasi method sebelumnya dengan sintaks sebagai berikut :

type method-name (arg-list) throws exception-list { }

Contoh :

```
class ThrowsDemo {
    static void procedure () thorws IllegalAccessException {
        System.out.println(“Inside Procedure”);
        throw new IllegalAccessException(“demo”);
    }
    public static void main(String args[]) {
        try {
                procedure();
        }
        catch (IllegalAccessException e) {
                System.out.println(“caught “+ e);
        }
    }
}
```

Output :

Inside procedure

caught java.lang.IllegalAccessException : demo

Finally

Saat suatu eksepsi dilemparkan, alur program dalam suatu method membuat jalur yang cenderung tidak linier melalui method tersebut, melompati baris-baris tertentu, bahkan mungkin akan keluar sebelum waktunya pada kasus dimana tidak ada bagian catch yang cocok. Kadang-kadang perlu dipastikan bahwa bagian program yang diberikan akan berjalan, tidak perduli eksepsi apa yang terjadi dan ditangkap. Kata kunci finally dapat digunakan untuk menentukan bagian program itu.

Setiap try membutuhkan sekurang-kurangnya satu bagian catch atau finally yang cocok. Jika kita tidak mendapatkan bagian catch yang cocok, maka bagian finally akan dieksekusi sebelum akhir program, atau setiap kali suatu method akan kembali ke pemanggilnya, melalui eksepsi yang tidak dapat ditangkap, atau melalui pernyataan return, bagian finally akan dieksekusi sebelum kembali ke method kembali.

Berikut adalah contoh program yang menunjukkan beberapa method yang keluar dengan berbagai cara, tidak satupun tanpa mengeksekusi bagian finally-nya.

```
class finallyDemo {
        static void proA() {
                try {
                        System.out.println(“Inside procA..”);
                        throw new RuntimeException(“Demo”);
                }
                finally {
                        System.out.println(“procA is finally”);
                }
        }
        static void proB() {
                try {
```

```
                System.out.println(“Inside procB..”);
                return;
        }
        finally {
                System.out.println(“procB is finally”);
        }
        }
    public static void main(String args[]) {
        try {
                procA{};
        }
        catch (Exception e);
        procB();
    }
}
```

Output :

```
Inside procA..
procA is finally
Inside procB..
procB is finally
```

Multithreading

Banyak persoalan dalam pemrograman membutuhkan kemampuan suatu program untuk melakukan beberapa hal sekaligus, atau memberikan penanganan segera terhadap suatu kejadian/ event tertentu dengan menunda aktivitas yang sedang dijalankan untuk menangani event tersebut dan akhirnya kembali melanjutkan aktivitas yang tertunda.

Contoh, dalam sistem aplikasi jaringan, kita dapat membuat suatu program melakukan komputasi lokal dengan data yang sudah didapat dari

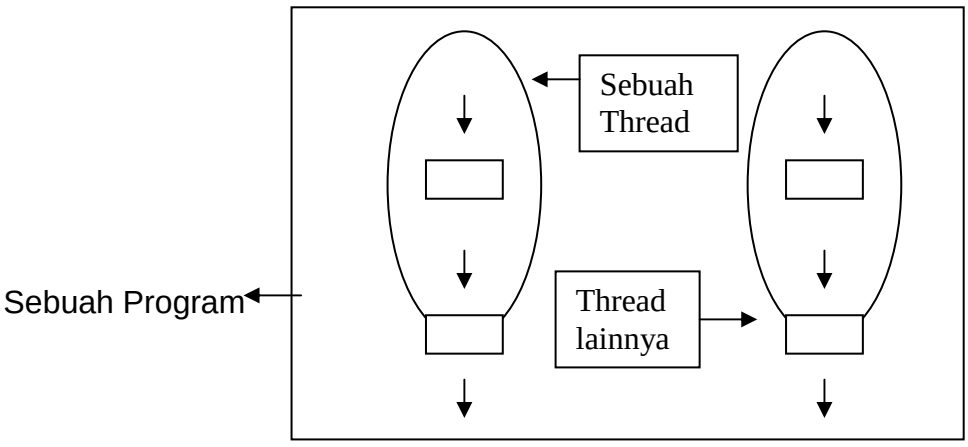
jaringan, pada saat program tersebut menunggu datangnya tambahan data dari jaringan. Tanpa multithreading, program tersebut harus melakukannya secara sekuensial dalam sebuah alur program tunggal (yaitu alur control utama), yang diawali dengan penantian tibanya keseluruhan data, baru kemudian komputasi. Pada masa penantian tersebut, komputer berada pada keadaan idle yang menyebabkan ketidakefisienan pada keseluruhan program.

Dengan multithreading kita dapat menciptakan dua thread secara dinamis, yaitu thread yang berjaga dipintu gerbang, menunggu masuknya data., dan thread yang melakukan komputasi lokal atas data yang sudah tersedia.

Multithreading dan Java

Thread (seringkali disebut juga *lightweight process* atau *execution context*) adalah sebuah *singlesequentialflow of control* didalam sebuah program. Secara sederhana, thread adalah sebuah subprogram yang berjalan didalam sebuah program.

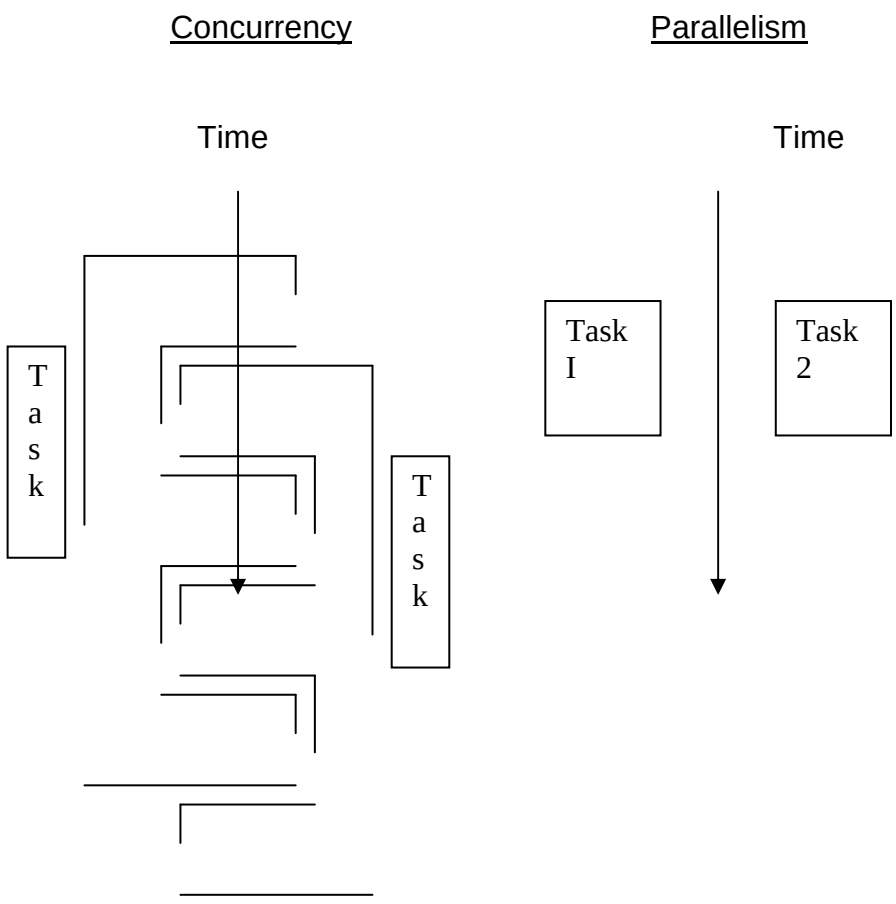
Seperti halnya sebuah program, sebuah thread mempunyai awal dan akhir. Sebuah program dapat mempunyai beberapa thread di dalamnya. Jadi perbedaannya program yang *multithreaded* mempunyai beberapa flow of control yang berjalan secara konkuren atau paralel sedangkan program yang *singlethreaded* hanya mempunyai satu flow of control.



Gb.1. Dua Thread dalam Satu Program

Dua program yang dijalankan secara terpisah (dari command line secara terpisah), berada pada dua address space yang terpisah. Sebaliknya, kedua thread pada gambar diatas berada pada address space yang sama (address space dari program dimana kedua thread tersebut dijalankan).

Kalau program itu berjalan diatas mesin dengan single processor, maka thread-thread itu dijalankan secara konkuren(dengan mengeksekusi secara bergantian dari satu thread ke thread yang lainnya). Jika program itu berjalan diatas mesin dengan multiple processor, maka thread-thread itu bisa dijalankan secara paralel (masing-masing thread berjalan di processor yang terpisah).

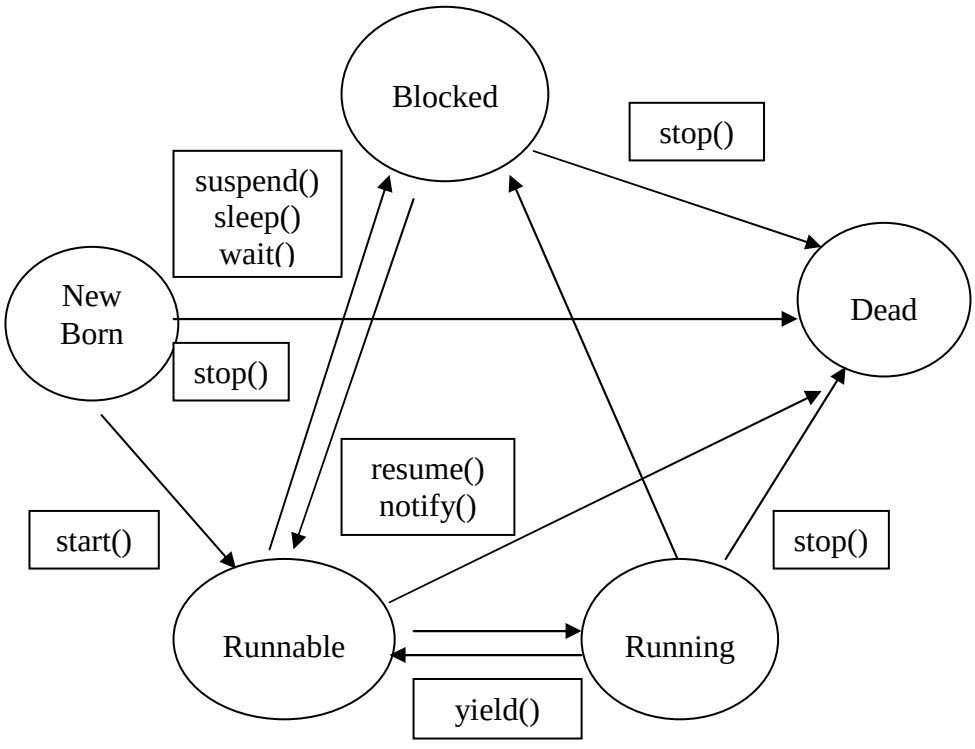


Gb.2. Konkurensi dan parallelism

Gambar 2 dapat menjelaskan perbedaan antara konkurensi dan parallelism. Bahasa Java mempunyai kemampuan multithreading built-in, pada Java Virtual Macjine terdapat thread scheduler yang menentukan thread mana yang beraksi pada selang waktu tertentu. Scheduler pada JVM mendukung *preemptive multithreading*, yaitu suatu thread dengan prioritas tinggi dapat menyeruak masuk dan menginterupsi thread yang sedang beraksi, kemampuan ini sangat menguntungkan dalam membuat aplikasi real-time.

Scheduler pada JVM juga mendukung *non-preemptive multithreading* I(atau sering disebut juga cooperative multithreading), yaitu thread yang sedang beraksi tidak dapat diinterupsi, ia akan menguasai waktu CPU, sampai menyelesaikan tugasnya atau secara eksplisit merelakan diri untuk berhenti dan memberi kesempatan bagi thread lain.

Daur Hidup sebuah Thread



Gb.3. State-state dari thread

Newborn

Sevbuah thread berada pada state ini ketika dia di instantiasi. Sebuah ruangan dimemori telah dialokasikan untuk thread itu,dan telah menyelesaikan tahap inisialisasinya.

```
……
Thread timerThread = new TimerThread();
……..
```

Pada state ini, timeThread belum masuk dalam skema penjadwalan thread scheduler.

Runnable

Pada state ini, sebuah thread berada dalam skema penjadwalan, akan tetapi dia tidak sedang beraksi. Kita bisa membuat timerThread yang kita buat sebelumnya masuk ke state runnable dengan :

```
…..
timerThread.start();
……
```

Kapan tepatnya timerThread beraksi, ditentukan oleh thread scheduler.

Running

Pada state ini, thread sedang beraksi. Jatah waktu beraksi bagi thread ini ditentukan oleh thread scheduler. Pada kasus tertentu, thread scheduler berhak meng-interupsikegiatan dari thread yang seddang beraksi (misalnya ada thread lainnya dengan prioritas yang lebih tinggi).

Thread dalam keadaan running bisa juga lengser secara sukarela, dan masuk kembali ke state runnable, sehingga thread lain yang sedang menunggu giliran(runnable) memperoleh kesempatan untuk beraksi. Tindakan thread yang lengser secara sukarela, biasanya disebut yield-ing.

```
public void run() {
```

…….
Thread.yield();
…….
}
Blocked

Pada tahap ini thread sedang tidak beraksi dan diabaikan dalam penjadwalan thread scheduler. Thread yang sedang terblok menunggu sampai syarat-syarat tertentu terpenuhi, sebelum ia kembali masuk kedalam skema penjadwalan thread scheduler (masuk state runnable lagi). Suatu thread menjadi terblok karena hal-hal berikut :

a. Thread itu tidur untuk jangka waktu tertentu, seperti berikut :

```
public void run() {
    ……
    try {
        thread.slepp(3000);
        //thread yg sedang beraksi akan tidur selama 3000 milisecond=3menit
        }
    catch (InterruptedException e) {
        …….
        }
```

b. Thread itu di- suspend(). Thread yang ter-suspend() itu bisa masuk kembali ke state runnable bila ia resume(). seperti hal berikut:

```
……
//timerThread akan segera memasuki state blocked
timerThread.suspend();
……..
timerThread.resume();
//timerThread kembali masuk state runnable
……
```

c. Bila thread tersebut memanggil method wait() dari suatu object yang sedang ia kunci. Thread tersebut bisa kembali memasuki state runnable bila ada thread lain yang memanggil method notify() atau notifyAll() dari object tersebut.

d. Bila thread ini menunggu selesainya aktifitas yang berhubungan dengan I/O. Misalnya, jika suatu thread menunggu datangnya bytes dari jaringan komputer maka secara otomatis thread tersebut masuk ke state blocked.

e. Bila suatu thread mencoba mengakses critical section dari suatu object yang sedang dikunci oleh thread lain. Critical section adalah method/blok kode yang ditandai dengan kata *synchronized*.

Dead

Suatu thread secara otomatis disebut mati bila method run() – nya sudah dituntaskan (return dari method run()). Contoh dibawah ini adalah thread yang akan mengecap state running hanya sekali saat thread scheduler memberinya kesempatan untuk running, ia akan mencetak " I'm doing something….something stupid….but I'm proud of It"… kemudian mati.

```
public class MyThread extends Thread {
        …..
        public void run() {
                System.out.print("I'm doing something…");
                System.out.print("something stupid…");
                System.out.println("but I'm proud of It…");
                // MyThread akan mati begitu baris diatas selesai dieksekusi
                }
        ……
        }
```

PACKAGE DAN INTERFACE

7

Obyektif :

1. Memahami persamaan dan perbedaan antara AWT dan Swing
2. Mendesain aplikasi GUI menggunakan AWT
3. Mendesain aplikasi GUI menggunakan Swing
4. Membuat tampilan yang kompleks dalam mendesain aplikasi GUI

Packages

Packages dalam JAVA berarti pengelompokan beberapa *class* dan *interface* dalam satu unit. Fitur ini menyediakan mekanisme untuk mengatur *class* dan *interface* dalam jumlah banyak dan menghindari konflik pada penamaan.

Mengimport Packages

Supaya dapat meggunakan *class* yang berada diluar *package* yang sedang dikerjakan, Anda harus mengimport *package* dimana *class* tersebut berada. Pada dasarnya, seluruh program JAVA mengimport *package java.lang.*,* sehingga Anda dapat menggunakan *class* seperti String dan Integer dalam program meskipun belum mengimport *package* sama sekali.

Penulisan import *package* dapat dilakukan seperti dibawah ini :

```
import <namaPaket>;
```

Sebagai contoh, bila Anda ingin menggunakan *class* Color dalam *package* awt, Anda harus menuliskan import *package* sebagai berikut :

```
import java.awt.Color;
import java.awt.*;
```

Baris pertama menyatakan untuk mengimport *class* Color secara spesifik pada *package*, sedangkan baris kedua menyatakan mengimport seluruh *class* yang terkandung dalam *package java.awt*.

Cara lain dalam mengimport *package* adalah dengan menuliskan referensi *package* secara eksplisit. Hal ini dilakukan dengan menggunakan nama *package* untuk mendeklarasikan *object* sebuah *class* :

```
java.awt.Color color;
```

Membuat Package

Untuk membuat *package*, dapat dilakukan dengan menuliskan :

```
package <packageName>;
```

Anggaplah kita ingin membuat *package* dimana *class* StudentRecord akan ditempatkan bersama dengan *class – class* yang lain dengan nama *package* schoolClasses. Langkah pertama yang harus dilakukan adalah membuat folder dengan nama schoolClasses. Salin seluruh *class* yang ingin diletakkan pada *package* dalam folder ini. Kemudian tambahkan kode deklarasi *package* pada awal file. Sebagai contoh :

```
package schoolClasses;
public class StudentRecord
{
private String name;
private String address;
private int age;
}
```

Package juga dapat dibuat secara bersarang. Dalam hal ini Java Interpreter menghendaki struktur direktori yang mengandung *class* eksekusi untuk disesuaikan dengan struktur *package*.

Pengaturan CLASSPATH

Diasumsikan *package* schoolClasses terdapat pada direktori C:\. Langkah selanjutnya adalah mengatur classpath untuk menunjuk direktori tersebut sehingga pada saat akan dijalankan, JVM dapat mengetahui dimana *class* tersebut tersimpan. Sebelum membahas cara mengatur classpath, perhatikan contoh dibawah yang menandakan kejadian bila kita tidak mengatur classpath.

Asumsikan kita mengkompilasi dan menjalankan *class* StudentRecord :

```
C:\schoolClasses>javac StudentRecord.java
C:\schoolClasses>java StudentRecord
Exception in thread "main" java.lang.NoClassDefFoundError:
StudentRecord
(wrong name: schoolClasses/StudentRecord)
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(Unknown Source)
at java.security.SecureClassLoader.defineClass(Unknown Source)
at java.net.URLClassLoader.defineClass(Unknown Source)
at java.net.URLClassLoader.access$100(Unknown Source)
at java.net.URLClassLoader$1.run(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at java.lang.ClassLoader.loadClassInternal(Unknown Source)
```

Kita akan mendapatkan pesan kesalahan berupa **NoClassDefFoundError** yang berarti JAVA tidak mengetahui dimana posisi *class*. Hal tersebut disebabkan oleh karena *class*StudentRecord berada pada *package* dengan nama studentClasses. Jika kita ingin menjalankan class tersebut, kita harus memberi informasi pada JAVA

bahwa nama lengkap dari *class* tersebut adalah **schoolClasses.StudentRecord**. Kita juga harusmenginformasikan kepada JVM dimana posisi pencarian *package*, yang dalam hal ini berada pada direktori C:\. Untuk melakukan langkah – langkah tersebut, kita harus mengaturclasspath. Pengaturan classpath pada Windows dilakukan pada *command prompt* :

C:\schoolClasses> **set classpath=C:**

dimana C:\ adalah direktori dimana kita menempatkan *package*. Setelah mengatur classpath, kita dapat menjalankan program di mana saja dengan mengetikkan :

C:\schoolClasses> java **schoolClasses.StudentRecord**

Pada UNIX, asumsikan bahwa kita memiliki *class - class* yang terdapat dalam direktori /usr/local/myClasses, ketikkan :

export classpath=/usr/local/myClasses

Perhatikan bahwa Anda dapat mengatur classpath dimana saja. Anda juga dapat mengatur lebih dari satu classpath, kita hanya perlu memisahkannya dengan menggunakan **;** (Windows), dan **:** (UNIX). Sebagai contoh :

```
set classpath=C:\myClasses;D:\;E:\MyPrograms\Java
```

dan untuk sistem UNIX :

```
export classpath=/usr/local/java:/usr/myClasses
```

Access Modifiers

Pada saat membuat, mengatur *properties* dan *class methods*, kita ingin untuk mengimplementasikan beberapa macam larangan untuk mengakses data. Sebagai contoh, jika Anda ingin beberapa atribut hanya dapat diubah hanya dengan *method* tertentu, tentu Anda ingin menyembunyikannya dari *object* lain pada *class*. Di JAVA, implementasi tersebut disebut dengan ***access modifiers.***

Terdapat 4 macam *access modifiers* di JAVA, yaitu : *public, private, protected* dan *default*. 3 tipe akses pertama tertulis secara ekplisit pada

kode untuk mengindikasikan tipe akses, sedangkan yang keempat yang merupakan tipe *default*, tidak diperlukan penulisan *keyword*atas tipe.

Akses Default (Package Accessibility)

Tipe ini mensyaratkan bahwa hanya *class* dalam *package* yang sama yang memiliki hak akses terhadap variabel dan *methods* dalam *class*. Tidak terdapat *keyword* pada tipe ini. Sebagai contoh :

```
public class StudentRecord
{
//akses dasar terhadap variabel
int name;
//akses dasar terhadap metode
String getName(){
return name;}}
```

Pada contoh diatas, variabel nama dan *method* getName() dapat diakses dari *object* lain selama *object* tersebut berada pada *package* yang sama dengan letak dari file StudentRecord.

Akses Public

Tipe ini mengijinkan seluruh *class member* untuk diakses baik dari dalam dan luar *class*. *Object* apapun yang memiliki interaksi pada *class* memiliki akses penuh terhadap *member* dari tipe ini. Sebagai contoh :

```
public class StudentRecord
{
//akses dasar terhadap variabel
public int name;
//akses dasar terhadap metode
public String getName(){
return name;
}
}
```

Dalam contoh ini, variabel name dan *method* getName() dapat diakses dari *object* lain.

Akses Protected

Tipe ini hanya mengijinkan *class member* untuk diakses oleh *method* dalam *class* tersebut dan elemen – elemen *subclass*. Sebagai contoh :

```
public class StudentRecord
{
//akses pada variabel
protected int name;
//akses pada metode
protected String getName(){
return name;
}
}
```

Pada contoh diatas, variabel name dan *method* getName() hanya dapat diakses oleh *method* internal *class* dan *subclass* dari *class* StudentRecord. Definisi *subclass* akan dibahas pada bab selanjutnya.

Akses Private

Tipe ini mengijinkan pengaksesan *class* hanya dapat diakses oleh *class* dimana tipe ini dibuat. Sebagai contoh :

```
public class StudentRecord
{
//akses dasar terhadap variabel
private int name;
//akses dasar terhadap metode
private String getName(){
return name;
}
}
```

Pada contoh diatas, variabel name dan *method* getName() hanya dapat diakses oleh *method* internal *class* tersebut. ***Petunjuk Penulisan Program :***

Interface

Interface adalah jenis khusus dari blok yang hanya berisi method signature(atau constant). Interface mendefinisikan sebuah(signature) dari sebuah kumpulan method tanpa tubuh.Interface mendefinisikan sebuah cara standar dan umum dalam menetapkan sifat-sifat dari class-class.
Mereka menyediakan class-class, tanpa memperhatikan lokasinya dalam hirarki class, untuk mengimplementasikan sifat-sifat yang umum. Dengan catatan bahwainterface-interface juga menunjukkan polimorfisme, dikarenakan program dapat memanggil method interface dan versi yang tepat dari method yang akan dieksekusi tergantung daritipe object yang melewati pemanggil method interface.

Kenapa Kita Memakai Interface?

Kita akan menggunakan interface jika *kita ingin class yang tidak berhubungan mengimplementasikan method yang sama*. Melalui interface-interface, kita dapatmenangkap kemiripan diantara class yang tidak berhubungan tanpa membuatnya seolaholah class yang berhubungan. Mari kita ambil contoh class **Line** dimana berisi method yang menghitung panjang dari garis dan membandingkan object **Line** ke object dari class yang sama. Sekarang, misalkan kita punya class yang lain yaitu **MyInteger** dimana berisi method yang membandingkan object **MyInteger** ke object dari class yang sama. Seperti yang kita lihat disini, kedua class-class mempunyai method yang mirip dimana membandingkan mereka dari object lain dalam tipe yang sama, tetapi mereka tidak berhubungan sama sekali. Supaya dapat menjalankan cara untuk memastikan bahwa dua class-class ini mengimplementasikan beberapa method dengan tanda yang sama, kita dapat menggunakan sebuah

interface untuk hal ini. Kita dapat membuat sebuah class interface, katakanlah interface **Relation** dimana mempunyai deklarasi method pembanding. Relasi interface dapat dideklarasikan sebagai,

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

Alasan lain dalam menggunakan interface pemrograman object adalah *untuk menyatakan sebuah interface pemrograman object tanpa menyatakan classnya.* Seperti yang dapat kita lihat nanti dalam bagian *Interface vs class*, kita dapat benar-benar menggunakan interface sebagai tipe data. Pada akhirnya, kita perlu menggunakan interface untuk pewarisan model jamak dimana menyediakan class untuk mempunyai lebih dari satu superclass. Pewarisan jamak tidak ditunjukkan di Java, tetapi ditunjukkan di bahasa berorientasi object lain seperti C++.

Interface vs. Class Abstract

Berikut ini adalah perbedaan utama antara sebuah interface dan sebuah class abstract: method interface tidak punya tubuh, sebuah interface hanya dapat mendefinisikan konstanta dan interface tidak langsung mewariskan hubungan dengan class istimewa lainnya, mereka didefinisikan secara independent.

Interface vs. Class

Satu ciri umum dari sebuah interface dan class adalah pada tipe mereka berdua. Ini artinya bahwa sebuah interface dapat digunakan dalam tempat-tempat dimana sebuah class dapat digunakan. Sebagai contoh,

diberikan class Person dan interface PersonInterface, berikut deklarasi yang benar :

```
PersonInterface pi = new Person();
Person pc = new Person();
```

Bagaimanapun, Anda tidak dapat membuat instance dari sebuah interface.

Contohnya :

```
PersonInterface pi = new PersonInterface(); //COMPILE
//ERROR!!!
```

Ciri umum lain adalah baik interface maupun class dapat mendefinisikan method. Bagaimanapun, sebuah interface tidak punya sebuah kode implementasi sedangkan class memiliki salah satunya.

Membuat Interface

Untuk membuat interface, kita tulis,

```
public interface [InterfaceName]
{
//beberapa method tanpa isi
}
```

Sebagai contoh, mari kita membuat sebuah interface yang mendefinisikan hubungan antara dua object menurut urutan asli dari object.

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

Sekarang, penggunaan interface, kita gunakan kata kunci **implements**. Contohnya,

```
/**
* Class ini mendefinisikan segmen garis
*/
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);
}
}
```

Ketika class Anda mencoba mengimplementasikan sebuah interface, selalu pastikan bahwa Anda mengimplementasikan semua method dari interface, jika tidak, Anda akan menemukan kesalahan ini,

```
Line.java:4: Line is not abstract and does not override abstract
method isGreater(java.lang.Object,java.lang.Object) in
Relation
public class Line implements Relation
^
1 error
```

Hubungan dari Interface ke Class

Seperti yang telah kita lihat dalam bagian sebelumnya, class dapat mengimplementasikansebuah interface selama kode implementasi untuk semua method yang didefinisikan dalam interface tersedia.

Hal lain yang perlu dicatat tentang hubungan antara interface ke class-class yaitu, class hanya dapat meng-EXTEND SATU superclass, tetapi dapat meng-IMPLEMENTASI-kan BANYAK interface. Sebuah contoh dari sebuah class yang mengimplementasikan interface adalah,

```
public class Person implements PersonInterface,
LivingThing,
WhateverInterface {
//beberapa kode di sini
}
```

Contoh lain dari class yang meng-extend satu superclass dan mengimplementasikan sebuah interface adalah,

```
public class ComputerScienceStudent extends Student
implements PersonInterface,
LivingThing {
//beberapa kode di sini
}
```

Catatan bahwa sebuah interface bukan bagian dari hirarki pewarisan class. Class yang tidak berhubungan dapat mengimplementasikan interface yang sama.

Pewarisan Antar Interface

Interface bukan bagian dari hirarki class. Bagaimanapun, interface dapat mempunyai hubungan pewarisan antara mereka sendiri. Contohnya, misal kita punya dua interface **StudentInterface** dan **PersonInterface**. Jika StudentInterface meng-extend PersonInterface, maka ia akan mewariskan semua deklarasi method dalam PersonInterface.

```
public interface PersonInterface {
. . .
}
public interface StudentInterface extends PersonInterface {
. . .
}
```

Abstract Windowing Toolkit (AWT) vs. Swing

The Java Foundation Class (JFC), merupakan bagian penting dari Java SDK, yang termasuk dalam koleksi dari API dimana dapat mempermudah pengembangan aplikasi JAVA GUI. JFC termasuk diantara 5 bagian utama dari API yaitu AWT dan Swing. Tiga bagian yang lainnya dari API adalah Java2D, *Accessibility,* dan Drag dan Drop. Semua itu membantu

pengembang dalam mendesain dan mengimplementasikan aplikasi visual yang lebih baik.

AWT dan Swing menyediakan komponen GUI yang dapat digunakan dalam membuat aplikasi Java dan Applet. Anda akan mempelajari applet pada bab berikutnya. Tidak seperti beberapa komponen AWT yang menggunakan *native code,* keseluruhan Swing ditulis menggunakan bahasa pemrograman Java. Swing menyediakan implementasi platform-independent dimana aplikasi yang dikembangkan dengan platform yang berbeda dapat memiliki tampilan yang sama. Begitu juga dengan AWT menjamin tampilan *look and feel* pada aplikasi yang dijalankan pada dua mesin yang berbeda menjadi terlihat sama. Swing API dibangun dari beberapa API yang mengimplementasikan beberapa jenis bagian dari AWT. Kesimpulannya, komponen AWT dapat digunakan bersama komponen Swing.

Komponen GUI pada AWT

Window Classes Fundamental

Dalam mengembangkan aplikasi GUI, komponen GUI seperti tombol atau textfield diletakkan di dalam kontainer. Berikut ini adalah daftar dari beberapa class penting pada kontainer yang telah disediakan oleh AWT.

Class AWT	Deskripsi
Komponen	Abstract Class untuk object yang dapat ditampilkan pada console dan berinteraksi dengang user. Bagian utama dari semua class AWT.
Kontainer	Abstract Subclass dari Component Class. Sebuah komponen yang dapat menampung komponen yang lainnya.
Panel	Turunan dari Container Class. Sebuah frame atau window tanpa titlebar, menubar tidak termasuk border. Superclass dari applet class.

Window	Turunan dari Container class. Top level window, dimana berarti tidak bias dimasukkan dalam object yang lainnya.Tidak memiliki border dan menubar.
Frame	Turunan dari window class. Window dengan judul, menubar, border dan pengatur ukuran di pojok. Memiliki empat constructor , dua diantaranya memiliki penulisan seperti di bawah ini : Frame() dan Frame(String title)

Tabel 1.2.1: Class kontainer AWT

Untuk mengatur ukuran window, menggunakan method setSize.

void setSize(int width, int height)

mengubah ukuran komponen ini dengan *width* dan *height* sebagai parameter.

void setSize(Dimension d)

mengubah ukuran dengan *d.width* dan *d.height* berdasar pada spesifikasi *Dimension d.*

Default dari window adalah *not visible* atau tak tampak hingga Anda mengatur *visibility* menjadi *true.* Inilah *syntax* untuk method *setVisible.*

void setVisible(boolean b)

Dalam mendesain aplikasi GUI, Object *Frame* selalu digunakan. Dibawah ini adalah contoh bagaimana membuat sebuah aplikasi.

```
import java.awt.*;
public class SampleFrame extends Frame {
public static void main(String args[]) {
SampleFrame sf = new SampleFrame();
sf.setSize(100, 100); //Coba hilangkan baris ini
sf.setVisible(true); //Coba hilangkan baris ini
}
}
```

perhatikan bahwa tombol tutup pada frame tidak akan bekerja karena tidak ada mekanisme *event handling* yang ditambahkan di dalam aplikasi. Anda akan belajar tentang *event handling* pada modul selanjutnya.

Grafik

Beberapa method grafik ditemukan dalam class *Graphic*. Dibawah ini adalah daftar dari beberapa method.

drawLine()	drawPolyline()	setColor()
fillRect()	drawPolygon()	getFont()
drawRect()	fillPolygon()	setFont()
clearRect()	getColor()	drawString()

Tabel 1.2.2a: Beberapa metode dari kelas Graphics

Hubungan dari class ini adalah class *Color*, dimana memiliki tiga constructor.

Format Constructor	**Deskripsi**
Color(int r, int g, int b)	Nilai integer 0 - 255.
Color(float r, float g, float b)	Nilai float 0.0 - 1.0.
Color(int rgbValue)	Panjang nilai : 0 ke 224-1 (hitam ke putih). Red: bits 16-23 Green: bits 8-15 Blue: bits 0-7

Dibawah ini adalah contoh program yang menggunakan beberapa method di dalam class G*raphic*.

```
import java.awt.*;
public class GraphicPanel extends Panel {
GraphicPanel() {
setBackground(Color.black); //Konstanta dalam class Color
}
public void paint(Graphics g) {
```

```
g.setColor(new Color(0,255,0)); //hijau
g.setFont(new Font("Helvetica",Font.PLAIN,16));
g.drawString("Hello GUI World!", 30, 100);
g.setColor(new Color(1.0f,0,0)); //red
g.fillRect(30, 100, 150, 10);
}
public static void main(String args[]) {
Frame f = new Frame("Testing Graphics Panel");
GraphicPanel gp = new GraphicPanel();
f.add(gp);
f.setSize(600, 300);
f.setVisible(true);
}
}
```

Agar panel dapat terlihat atau *visible,* dia harus diletakkan didalam window yang dapat terlihat seperti sebuah frame.

Beberapa komponen AWT

Berikut ini adalah daftar dari kontrol AWT. Kontrol adalah komponen seperti tombol atau textfield yang mengijinkan user untuk berinteraksi dengan aplikasi GUI. Berikut ini semua subclass dari class *Components*.

Label	Button	Choice
TextField	Checkbox	List
TextArea	CheckboxGroup	Scrollbar

Tabel 1.2.3: Komponen AWT

Berikut adalah aplikasi membuat sebuah frame dengan kontrol yang telah dimasukkan di dalamnya.

```
import java.awt.*;
class FrameWControls extends Frame {
public static void main(String args[]) {
FrameWControls fwc = new FrameWControls();
```

```
fwc.setLayout(new FlowLayout()); //akan dibahas lebih detail
pada pembahasan berikutnya
fwc.setSize(600, 600);
fwc.add(new Button("Test Me!"));
fwc.add(new Label("Labe"));
fwc.add(new TextField());
CheckboxGroup cbg = new CheckboxGroup();
fwc.add(new Checkbox("chk1", cbg, true));
fwc.add(new Checkbox("chk2", cbg, false));
fwc.add(new Checkbox("chk3", cbg, false));
List list = new List(3, false);
list.add("MTV");
list.add("V");
fwc.add(list);
Choice chooser = new Choice();
chooser.add("Avril");
chooser.add("Monica");
chooser.add("Britney");
fwc.add(chooser);
fwc.add(new Scrollbar());
fwc.setVisible(true);
}
}
```

Layout Manager

Posisi dan ukuran suatu komponen ditentukan oleh layout manager. Layout manager mengatur tampilan dari komponen di dalam kontainer. Berikut ini beberapa layout manager yang terdapat di dalam Java.

1. FlowLayout
2. BorderLayout
3. GridLayout

4. GridBagLayout
5. CardLayout

Layout manager dapat diatur menggunakan method *setLayout* dari class *Container.* Method ini dapat ditulis sebagai berikut.

void setLayout(LayoutManager mgr)

Jika Anda memilih untuk tidak menggunakan layout manager, Anda dapat mengisi null sebagai argumen untuk method ini. Tetapi selanjutnya, Anda akan mengatur posisi elemen secara manual dengan menggunakan method *setBounds* dari class Components.

public void setBounds(int x, int y, int width, int height)

Method ini mengatur posisi berdasarkan pada argumen x dan y, dan ukuran berdasarkan argumen *width* dan *height.* Hal ini akan cukup menyulitkan dan membosankan untuk aplikasi jika Anda memiliki beberapa objek komponen didalam object container. Anda akan memanggil method ini untuk setiap komponen.

FlowLayout Manager

FlowLayout Manager adalah default manager untuk class *Panel* dan subclassnya, termasuk class applet. Cara meletakkan komponen dari FlowLayout Manager dimulai dari kiri ke kanan dan dari atas ke bawah, dimulai dari pojok kiri atas. Seperti pada saat Anda mengetik menggunakan editor kata pada umumnya. Berikut adalah bagaimana FlowLayout Manager bekerja, dimana memiliki tiga constructor seperti daftar di bawah ini.

FlowLayout Constructors
FlowLayout()
Membuat object baru FlowLayout dengan posisi di tengah dan lima unit horizontal dan vertical gap dimasukkan pada komponen sebagai default.

FlowLayout(int align)
Membuat object baru FlowLayout dengan posisi spesifik dan lima unit horizontal dan vertical gap dimasukkan pada komponen sebagai default.
FlowLayout(int align, int hgap, int vgap)
Membuat object baru FlowLayout dengan argumen pertama sebagai posisi pada komponen dan *hgap* untuk horizontal dan *vgap* untuk vertikal pada komponen

Tabel 1.3.1: Constructor FlowLayout

Gap dapat dikatakan sebagai jarak antara komponen dan biasanya diukur dengan satuan pixel. Posisi argumen mengikuti penulisan sebagai berikut:

1. FlowLayout.LEFT
2. FlowLayout.CENTER
3. FlowLayout.RIGHT

Bagaimanakah output dari program berikut :

```
import java.awt.*;
class FlowLayoutDemo extends Frame {
public static void main(String args[]) {
FlowLayoutDemo fld = new FlowLayoutDemo();
fld.setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 10));
fld.add(new Button("ONE"));
fld.add(new Button("TWO"));
fld.add(new Button("THREE"));
fld.setSize(100, 100);
fld.setVisible(true);
}
}
```

BorderLayout Manager

BorderLayout membagi kontainer menjadi lima bagian diantaranya utara, selatan, timur, barat, dan tengah. Setiap komponen dimasukkan ke dalam region yang spesifik. Region utara dan selatan membentuk jalur horizontal sedangkan region timur dan barat membentuk jalur vertikal. Dan region tengah berada pada perpotongan jalur horizontal dan vertikal. Tampilan ini adalah bersifat default untuk object *Window*, termasuk object dari subclass *Window* yaitu tipe Frame dan Dialog.

Constructor BorderLayout
BorderLayout()
Membuat object BorderLayout baru tanpa spasi yang diaplikasikan diantara komponen yang berbeda.
BorderLayout(int hgap, int vgap)
Membuat object BorderLayout baru dengan spasi unit *hgap* horizontal dan unit *vgap* vertical yang diaplikasikan diantara komponen yang berbeda.

Tabel 1.3.2: Constructor BorderLayout

Seperti pada FlowLayout Manager, parameter *hgap* dan *vgap* disini juga menjelaskan jarak antara komponen dengan kontainer.

Untuk menambahkan komponen ke dalam region yang spesifik, gunakan method menambahkan dan melewatkan dua argumen yaitu : komponen yang ingin dimasukkan ke dalam region dan region mana yang ingin dipakai untuk meletakkan komponen. Perlu diperhatikan bahwa hanya satu komponen yang dapat dimasukkan dalam satu region. Menambahkan lebih dari satu komponen pada kontainer yang bersangkutan, maka komponen yang terakhir ditambahkan yang akan ditampilkan. Berikut ini adalah daftar dari kelima region.

1. BorderLayout.NORTH
2. BorderLayout.SOUTH
3. BorderLayout.EAST
4. BorderLayout.WEST

5. BorderLayout.CENTER

Berikut ini adalah contoh program yang menunjukkan bagaimana *BorderLayout* bekerja.

```
import java.awt.*;
class BorderLayoutDemo extends Frame {
public static void main(String args[]) {
BorderLayoutDemo bld = new BorderLayoutDemo();
bld.setLayout(new BorderLayout(10, 10)); //may remove
bld.add(new Button("NORTH"), BorderLayout.NORTH);
bld.add(new Button("SOUTH"), BorderLayout.SOUTH);
bld.add(new Button("EAST"), BorderLayout.EAST);
bld.add(new Button("WEST"), BorderLayout.WEST);
bld.add(new Button("CENTER"), BorderLayout.CENTER);
bld.setSize(200, 200);
bld.setVisible(true);
}
}
```

GridLayout Manager

Dengan *GridLayout manager*, komponen juga diposisikan dari kiri ke kanan dan dari atas ke bawah seperti pada *FlowLayout manager. GridLayout manager* membagi kontainer menjadi baris dan kolom. Semua region memiliki ukuran yang sama. Hal tersebut tidak mempedulikan ukuran sebenarnya dari komponen.

Berikut ini adalah daftar dari constructor untuk class *GridLayout*.

Constructor GridLayout
GridLayout()
Membuat object GridLayout baru dengan satu baris dan satu kolom sebagai default

GridLayout(int rows, int cols)
Membuat object GridLayout baru dengan jumlah baris dan kolom sesuai dengan keinginan
GridLayout(int rows, int cols, int hgap, int vgap)
Membuat object GridLayout baru dengan jumlah baris dan kolom yang ditentukan. Unit spasi hgap horizontal dan vgap vertikal diaplikasikan ke dalam komponen.

Tabel 1.3.3: Constructor GridLayout

Cobalah program ini.

```
import java.awt.*;
class GridLayoutDemo extends Frame {
public static void main(String args[]) {
GridLayoutDemo gld = new GridLayoutDemo();
gld.setLayout(new GridLayout(2, 3, 4, 4));
gld.add(new Button("ONE"));
gld.add(new Button("TWO"));
gld.add(new Button("THREE"));
gld.add(new Button("FOUR"));
gld.add(new Button("FIVE"));
gld.setSize(200, 200);
gld.setVisible(true);
}
}
```

Panel dan Tampilan kompleks

Untuk membuat tampilan yang lebih lengkap, Anda dapat menggabungkan layout manager yang berbeda dengan menggunakan panel. Ingatlah bahwa panel adalah kontainer dan komponen pada saat yang sama. Anda dapat memasukkan komponen ke dalam panel dan

kemudian menambahkan panel ke dalam region yang Anda inginkan di dalam kontainer.

Perhatikan teknik yang digunakan pada contoh berikut.

```
import java.awt.*;
class ComplexLayout extends Frame {
public static void main(String args[]) {
ComplexLayout cl = new ComplexLayout();
Panel panelNorth = new Panel();
Panel panelCenter = new Panel();
Panel panelSouth = new Panel();
/* Panel utara */
//Panel menggunakan FlowLayout sebagai default
panelNorth.add(new Button("ONE"));
panelNorth.add(new Button("TWO"));
panelNorth.add(new Button("THREE"));
/* Panel tengah */
panelCenter.setLayout(new GridLayout(4,4));
panelCenter.add(new TextField("1st"));
panelCenter.add(new TextField("2nd"));
panelCenter.add(new TextField("3rd"));
panelCenter.add(new TextField("4th"));
/* Panel selatan */
panelSouth.setLayout(new BorderLayout());
panelSouth.add(new Checkbox("Choose me!"),
BorderLayout.CENTER);
panelSouth.add(new Checkbox("I'm here!"),
BorderLayout.EAST);
panelSouth.add(new Checkbox("Pick me!"),
BorderLayout.WEST);
/* Menambahkan panel pada container Frame*/
//Frame menggunakan BorderLayout sebagai default
```

```
cl.add(panelNorth, BorderLayout.NORTH);
cl.add(panelCenter, BorderLayout.CENTER);
cl.add(panelSouth, BorderLayout.SOUTH);
cl.setSize(300,300);
cl.setVisible(true);
}
}
```

Komponen Swing

Seperti pada package AWT, package dari Swing menyediakan banyak class untuk membuat aplikasi GUI. Package tersebut dapat ditemukan di *javax.swing*. Perbedaan utama antara keduanya adalah komponen Swing ditulis menyeluruh menggunakan Java. Kesimpulannya, program GUI ditulis menggunakan banyak class dari package Swing yang mempunyai tampilan look and feel yang sama meski dijalankan pada paltform yang berbeda. Lebih dari itu, Swing menyediakan komponen yang lebih menarik seperti *color chooser* dan *option pane.*

Nama dari komponen GUI milik Swing hampir sama persis dengan komponen GUI milik AWT.

Perbedaan jelas terdapat pada penamaan komponen. Pada dasarnya, nama komponen Swing sama dengan nama komponen AWT tetapi dengan tambahan huruf J pada prefixnya. Sebagai contoh, satu komponen dalam AWT adalah button class. Sedangkan pada Swing, nama komponen tersebut menjadi Jbutton class. Berikut adalah daftar dari komponen Swing.

Komponen Swing	**Penjelasan**
JComponent	class induk untuk semua komponen Swing, tidak termasuk top-level container
JButton	Tombol “push”. Berhubungan dengan class button dalam package AWT

JCheckBox	Item yang dapat dipilih atau tidak oleh pengguna. Berhubungan dengan class checkbox dalam package AWT
JFileChooser	Mengijinkan pengguna untuk memilih sebuah file. Berhubungan dengan class filechooser dalam package AWT
JTextField	Mengijinkan untuk mengedit text satu baris. Berhubungan dengan class textfield dalam package AWT.
JFrame	Turunan dan Berhubungan dengan class frame dalam package AWT tetapi keduanya sedikit tidak cocok dalam kaitannya dengan menambahkan komponen pada kontainer. Perlu mendapatkan content pane yang terbaru sebelum menambah sebuah komponen
JPanel	Turunan Jcomponent. Class Container sederhana tetapi bukan top-level. Berhubungan dengan class panel dalam package AWT.
JApplet	Turunan dan Berhubungan dengan class Applet dalam package AWT. Juga sedikit tidak cocok dengan class applet dalam kaitannya dengan menambahkan komponen pada container
JOptionPane	Turunan Jcomponent. Disediakan untuk mempermudah menampilkan popup kotak dialog.
JDialog	Turunan dan Berhubungan dengan class dialog dalam package AWT. Biasanya digunakan untuk menginformasikan sesuatu kepada pengguna atau prompt pengguna untuk input.
JColorChooser	Turunan Jcomponent. Memungkinkan pengguna untuk memilih warna yang diinginkan.

Tabel 1.4: Beberapa komponen Swing

Untuk daftar yang lengkap dari komponen Swing, Anda dapat melihatnya di dokumentasi

Setting Up Top-Level Containers

Seperti disebutkan diatas, top-level containers seperti *Jframe* dan *Japplet* dalam Swing sangat tidak cocok dengan AWT. Ini adalah syarat menambahkan komponen ke dalam kontainer. Jika Anda ingin menambahkan langsung sebuah komponen kedalam kontainer sebagai container AWT, pertama-tama Anda telah mendapatkan content pane dari kontainer. Untuk melakukan hal tersebut, Anda akan menggunakan method *getContentPane* dari container.

Contoh Jframe

```
import javax.swing.*;
import java.awt.*;
class SwingDemo {
JFrame frame;
JPanel panel;
JTextField textField;
JButton button;
Container contentPane;
void launchFrame() {
/* inisialisasi */
frame = new JFrame("My First Swing Application");
panel = new JPanel();
textField = new JTextField("Default text");
button = new JButton("Click me!");
contentPane = frame.getContentPane();
/* menambahkan komponen-komponen ke panel–
menggunakan
FlowLayout sebagai default */
```

```
panel.add(textField);
panel.add(button);
/* menambahkan komponen-komponen contentPane– menggunakan
BorderLayout */
contentPane.add(panel, BorderLayout.CENTER);
frame.pack();
//menyebabkan ukuran frame menjadi dasar pengaturan komponen
frame.setVisible(true);
}
public static void main(String args[]) {
SwingDemo sd = new SwingDemo();
sd.launchFrame();
}
}
```

Perlu diperhatikan pada package *java.awt* masih saja diimpor karena layout manager yang digunakan terdapat pada package tersebut. Juga, memberi judul pada frame dan mengepack komponen di dalam frame dapat juga dilakukan untuk frame AWT.

Petunjuk penulisan program:

Perhatikan penulisan kode yang digunakan pada contoh ini tampak berlawanan dengan contoh untuk AWT. Komponen dideklarasikan sebagai fields, method launchFrame ditentukan, dinisialisasikan dan penambahan semua komponen dilaksanakan di dalam method launchFrame. Kita tidak lagi meng-extend Frame class. Keuntungan penggunaan model ini akan lebih berguna ketika sampai pada event handling.

Contoh JOptionPane

```
import javax.swing.*;
class JOptionPaneDemo {
JOptionPane optionPane;
void launchFrame() {
optionPane = new JOptionPane();
String name = optionPane.showInputDialog("Hi, what's your
name?");
optionPane.showMessageDialog(null,
"Nice to meet you, " + name + ".", "Greeting...",
optionPane.PLAIN_MESSAGE);
System.exit(0);
}
public static void main(String args[]) {
new JOptionPaneDemo().launchFrame();
}
}
```

Lihat, begitu mudahnya memasukkan input dari user.

DAFTAR PUSTAKA

1. Anonim, *Pemrograman JAVA+ (Web Programming Servlet & JSP)*. Nurul Fikri, Depok 2002.

2. Hariyanto,Bambang, Ir., MT, *Esensi-esensi Bahasa Pemrograman JAVA*, Informatika,Bandung,2005

3. Herbert Schildt, *Java2 : A Beginner's Guide, Second Edition*, McGrawHill / Osborne.

4. Indrajani, Martin, *Pemrograman Berorientasi Objek dengan Java*, Elex Media Komputindo, Jakarta, 2004.

5. Isak, Rickyanto, ST, *Dasar Pemrograman Berorientasi Objek Dengan JAVA 2(JDK 1.4) ,* Andi Offset, Yogyakarta, 2003.

6. Patric Naughyon, *Java Handbook : Konsep dasar Pemrograman Java*, McGrawHill / Osborne