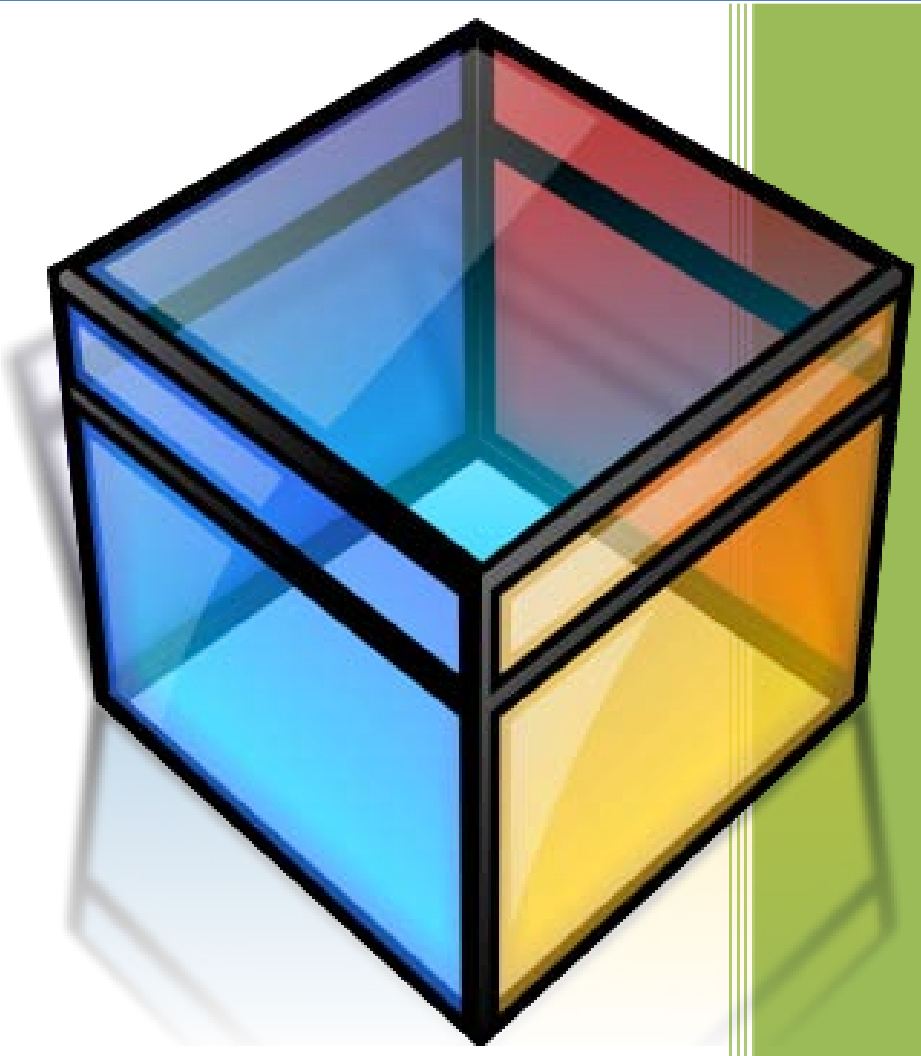
2008

Pengenalan Pemrograman Java



Hendro Steven Tampake

X3-Solution

BAB I
PENDAHULUAN

1.1 Sejarah Perkembangan Java

Java secara resmi diperkenalkan oleh SUN pada dunia pada tanggal 23 Mei 1995. Sedangkan sejarah pembangunan Java sendiri sudah dimulai sejak tahun 1991. Saat itu tim "Stealth Project" mengadakan pertemuan (*brainstorming*) untuk menciptakan suatu sistem *software* yang mampu berjalan pada alat-alat elektronik (*small devices*).

James Gosling berkonsentrasi pada ide pembuatan bahasa pemrograman. Pada Juni 1991, muncullah bahasa interpreter "Oak" yang menjadi cikal bakal dari Java. Kemudian secara resmi pada tahun 1995 Java diperkenalkan bersama dengan browser HotJava , Java pun merambah ke dunia web.

Kenyataan ini mungkin agak sedikit berbeda dengan ide pembuatan Java pada awalnya. Internet ternyata membantu menjadikan Java terkenal seperti sekarang ini. Memang harus diakui karena semakin berkembangnya Internet, maka fokus pemograman saat ini mengarah ke pemograman Internet itu sendiri.

Saat ini Java dibagi menjadi 3 macam *framework* atau teknologi yaitu J2SE untuk pemogramman aplikasi berbasis *console* dan *destop,* kemudian J2EE untuk pemogramman aplikasi berskala *enterprise* seperti aplikasi *web-base* (JSP dan Servlet), komponen (EJB), *Web Services* dan lain – lain. Kemudian framework yang terakhir adalah J2ME untuk pemogramman *small divice* seperti hanphone dan pda.

1.2 Kelebihan dan Karakteristik Java

Sintaks bahasa pemogramman Java adalah pengembangan dari bahasa pemogramman C/C++. Sehingga bagi mereka yang sudah terbiasa dengan C/C++, tidak akan mengalami kesulitan mempelajari bahasa pemogramman Java.

Java adalah bahasa pemogramman yang sederhana dan tangguh. Berikut ini adalah beberapa karakteristik dari Java sesuai dengan *white paper* dari Sun.

1. Berorientasi Object, java telah menerapkan konsep pemograman berorientasi object yang modern dalam implementasinya.
2. Robust, java mendorong pemograman yang bebas dari kesalahan dengan bersifat *strongly typed* dan memiliki *run-time checking*.
3. Protable, program java dapat berjalan pada sistem operasi apapun yang miliki Java Virtual Machine.
4. Multithreading, Java mendukung pemograman multithreading dan telah terintegrasi secara langsung dalam bahasa Java.

5. Dinamis, program Java dapat melakukan sesuatu tindakan yang ditentukan pada saat eksekusi program dan bukan pada saat kompilasi.
6. Sederhana, Java menggunakan bahasa yang sederhana dan mudah dipelajari.
7. Terdistribusi, Java didesain untuk berjalan pada lingkungan yang terdistribusi seperti halnya internet.
8. Aman, aplikasi yang dibuat dengan bahasa java lebih dapat dijamin keamanannya terutama untuk aplikasi internet.
9. Netral secara arsitektur, Java tidak terikat pada suatu mesin atau sistem operasi tertentu.
10. Interpreted, aplikasi Java bisa dieksekusi pada platform yang berbeda-beda karena melakukan interpretasi pada bytecode.
11. Berkinerja Tinggi, bytecode Java telah teroptimasi dengan baik sehingga eksekusi program dapat dilakukan secara cepat.

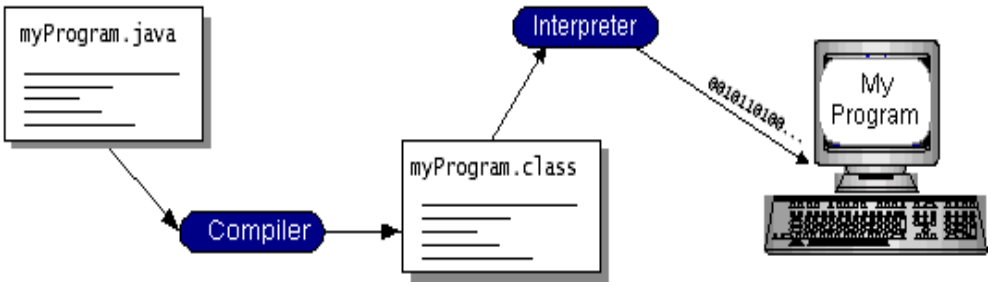
1.3 Menginstal Java

Untuk mulai membuat program dengan Java kita harus memiliki J2SE JDK (*Java Development Kit*) dan dokumentasi API (*Application Programming Interface*). Keduanya dapat di*download* dari website Sun (http://java.sun.com) secara gratis.

Sun juga menyediakan Java Runt-time Environment (JRE) yang hanya digunakan untuk menjalankan aplikasi – aplikasi yang ditulis dengan Java. Yang perlu diperhatikan adalah apabila kita hanya ingin menjalankan aplikasi – aplikasi Java maka kita hanya perlu menginstal JRE. Sedangkan apabila kita akan membuat program menggunakan Java maka kita perlu menginstal JDK. Di dalam JDK sudah terdapat JRE. Setelah selesai menginstal JDK/JRE pastikan <dir instalasi JDK/bin> terdaftar pada PATH sistem windows.

1.4 Java Bytecode

Lingkungan pemrograman pada java menggunakan compiler sekaligus Interpreter agar dapat berjalan pada platform yang berbeda. Java compiler melakukan kompilasi pada *source code* menjadi java *bytecodes.* Java *bytecodes* merupakan instruksi mesin yang tidak spesifik terhadap *processor* pada sistem computer. *Bytecode* dapat dijalankan menggunakan *Java Virtual Machine* (JVM) yang disebut juga *bytecodes interpreter* atau *Java runtime interpreter*. (lihat pada Gambar 1.1)



Gambar 1.1 Java Virtual Machice

Source code pada java diakhiri dengan ekstensi **.java** (Contoh HelloWorld.java). pada saat dikompilasi file java akan menjadi berakhiran **.class** (Contoh HelloWorld.class). Untuk mengkompilasi program java dipergunakan **javac.exe** dari JDK dan untuk menjalankan aplikasi *java stand-alone* dipergunakan **java.exe**

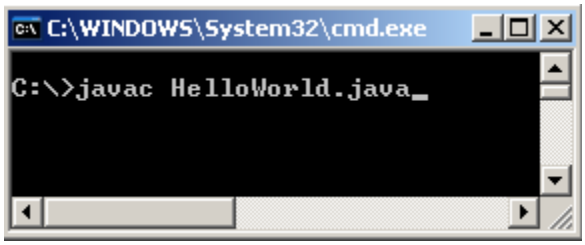
BAB II
SINTAKS BAHASA DAN TIPE DATA

2.1 Struktur Program Java

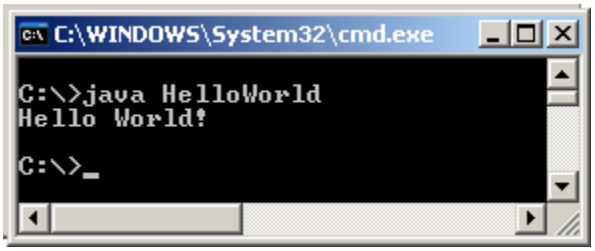
```
public class HelloWorld{
  public static void main(String args[]){
      System.out.printn(“Hello World!”);
  }
}
```

Skrip 2.1 HelloWorld.java

Contoh program di atas adalah contoh program sederhana yang ditulis dengan Java. Apabila kita compile dan jalankan maka program di atas akan menampilkan kalimant “Hello World!” pada *console*.



Gambar 2.1 Meng*compile* Program Java



Gambar 2.2 Menjalankan Program Java

2.1.1 Comments / Komentar

Comments digunakan untuk memberikan keterangan /penjelasan pada program. Comments tidak akan dikompilasi oleh compiler.

Lambang untuk Comments:

double slash (//) untuk komentar satu baris.

/* dan */ untuk komentar satu baris atau lebih.

Contoh:

```
public class HelloWorld{
    //baris ini tidak akan dieksekusi
    public static void main(String args[]){
         System.out.println("Selamat Belajar Java");
    }
    /* Baris ini adalah komentar
        yang lebih dari satu baris
        juga tidak akan dieksekusi
    */
}
```

Skrip 2.2 Contoh Komentar

2.1.2 Keywords / Reserved Words

Keywords / Reserved Words adalah kata-kata yang memiliki arti yang spesifik bagi kompiler dan tak bisa dipakai untuk kegunaan lainnya pada program. Perlu diingat bahwa java adalah ***case-sensitive***.

Java Keywords and Reserved Words

abstract	class	false	import	package	super	try
assert	const	final	instanceof	private	switch	void
boolean	continue	finally	int	protected	synchronized	volatile
break	default	float	interface	public	this	while
byte	do	for	long	return	throw	
case	double	goto	native	short	throws	
catch	else	if	new	static	transient	
char	extends	implements	null	strictfp	true	

Gambar 2.3 Java Keywords and Reserved Words

2.1.3 Modifier

Modifiers digunakan untuk menentukan penggunaan dari suatu data,methods dan *class.* Contoh *modifiers* adalah public, static, private, final, abstract dan protected.

2.1.4 Statements

Statements merupakan baris perintah atau kumpulan perintah. Setiap *statements* pada java selalu diakhiri dengan titik koma (;).

2.1.5 Blocks

Blocks digunakan untuk membentuk suatu grup *statements*. *Blocks* diawali dengan kurung kurawal buka ({ } dan kurung kurawal tutup (}). Blocks dapat digunakan secara *nested* (*blocks* didalam *blocks*).

2.1.6 Classes

Classes merupakan inti dari program Java. Suatu *class* merupakan "blueprint" untuk menciptakan suatu object.

2.1.7 Methods

Methods merupakan kumpulan *statements* yang berfungsi melakukan tugas tertentu di dalam program. Untuk aplikasi Java kecuali Applet harus mempunyai method **main().** Method main pada java selalu berbentuk:

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

2.2 Identifiers

Identifier merupakan penamaan pada pemograman untuk variabel, konstanta, method, class dan package. Java merupakan bahasa pemograman yang *case-sensitive* (membedakan huruf besar dan kecil). Aturan penamaan *identifiers* dalam java adalah:

1. Dapat dimulai dengan huruf, *underscore* (_) atau tanda $
2. Tidak boleh menggunakan simbol operator seperti : + - * / dan lain – lain
3. Tidak boleh menggunakan *reserved words*
4. Panjang *Identifier* beleh berapa saja.

2.3 Naming Convention

Selain mengikuti aturan penulisan indentifiers di atas, penulisan program pada Java juga memiliki bentuk – bentuk tertentu yang sudah dibakukan dan diakui secara internasional. Bentuk penulisan program java ini biasa disebut *Java Naming Convention*.

1. Penamaan Class

   Huruf pertama setiap kata harus huruf besar. Contoh:

   HelloWorld

   Employee

   BankAccount

2. Penamaan Method

   Huruf pertama setiap kata harus huruf besar, kecuali kata pertama. Contoh:

   getEmployeeName()

   setSpeedLimit()

   accelerate()

3. Penamaan Field atau variabel

   Huruf pertama setiap kata harus huruf besar, kecuali kata pertama. Contoh:

   employeeName

   employeeAccountNumber

   address

4. Penamaan Konstatanta / Constant

   Semua huruf harus huruf besar. Apabila lebih dari satu kata, gunakan *underscore* (_) sebagai pemisah. Contoh:

   PI

   MIN_RATE

   MAX_HEIGHT

2.4 Variabel, Konstanta dan Tipe Data

2.4.1 Variabel

Variabel merupakan lokasi penyimpanan yang ada di memori. Setiap variabel memiliki kemampuan menyimpan suatu informasi sesuai dengan tipe data yang didekalrasikan untuk variabel tersebut saja. Sintaks pendekalrasian variabel secara umum adalah sebagai berikut:

```
tipe-data nama-variabel
```

Tipe-data meliputi semua tipe data yang dikenal oleh Java, sedangkan nama-variabel adalah *identifier* yang digunakan untuk merujuk ke variabel tersebut di dalam program. Contoh kode:

```
int counter;
```

Kode di atas mendeklarasikan suatu variabel yang bernama counter dengan tipe data **int**.

2.4.2 Scope Variabel

Scope dari variabel dapat dibedakan menjadi dua bagian yaitu : variabel yang dideklarasikan di dalam blok class dan variabel yang dideklarasikan di dalam blok kode.

Variabel yang dideklarasikan pada blok class, akan dikenali oleh seluruh bagian class. Variabel ini juga dapat diakses dari luar class tetapi tergantung dari *access specifier*nya. Mengenai hal *access spesifier* akan dibahas pada bab selanjutnya.

Variabel yang dideklararsikan di dalam blok kode tertentu hanya akan dikenali di dalam blok kode tersebut. Untuk lebih jelasnya perhatikan contoh berikut.

```
public class Scope{
   static int x=10;
   public static void main(String[] args){
      int a = 5;
      // variabel x dapat diakses
      System.out.println("Nilai x="+x);
      System.out.println("Nilai a="+a);
      //nested blok
      {
         int b = 8;
         // variabel x masih dikenali
         System.out.println("Nilai x="+x);
         // variabel a juga dikenali
         System.out.println("Nilai a="+a);
         System.out.println("Nilai b="+b);

      }
      //variabel tidak dikenali
      System.out.println("Nilai b="+b);
   }
}
```

Skrip 2.4 Scope Variabel

2.4.3 Konstanta

Konstanta merupakan data yang tidak berubah nilainya selama program berjalan. Pendekalrasian konstanta menggunakan:

final tipe-data NAMA_CONSTANTA = value;

2.4.4 Tipe Data

Tipe data diperlukan agar compiler tahu operasi apa yang valid dan seberapa banyak memori yang diperlukan oleh sebuah nilai yang akan disimpan atau dioperasikan. Di dalam Java terdapat tiga tipe data yaitu tipe data primitif, tipe data referensi dan array.

2.4.4.1 Tipe Data Primitif

Tipe data primitif merupakan tipe data dasar yang dikenal oleh Java.

Tipe Data	Besar (bits)	Jangkauan
long	64	-2^{63} s/d $2^{63} - 1$
int	32	-2^{31} s/d $2^{31} - 1$
short	16	-2^{15} s/d $2^{15} - 1$
byte	8	-2^{7} s/d $2^{7} - 1$

double	64	Negatif: -1.7976931348623157E+308 s/d -4.94065645841246544E-324 Positif: 4.94065645841246544E-324 s/d 1.7976931348623157E+308
float	32	Negatif: -3.4028234663852886E+38 s/d -1.40129846432481707E-45 Positif: 1.40129846432481707E-45 s/d 3.4028234663852886E+38
char	16	'\u0000' s/d '\uFFFF' (0 s/d 65535)
boolean	8	true atau false

Tabel 2.1 Tipe Data Primitif dalam Java

Ke delapan tipe data di atas dapat dikelompokan menjadi empat kolompok:

1. Integer, merupakan tipe data bilangan bulat yang terdiri atas **byte**,**short**, **int**, dan **long**.
2. Floating-Point, merupakan tipe data bilangan pecahan yang terdiri ata **float** dan **double**.
3. Karakter, mewakili simbol dari sebuah karakter yang terdiri ata **char**.
4. Boolean, merupakan tipe data yang menunjukan nilai true atau false, yang terdiri atas **boolean**

2.4.4.2 Tipe Data Referensi

Tipe data referensi digunakan untuk memegang referensi dari suatu object (instance dari class). Pendeklarasian tipe data ini sama dengna tipe data primitif, namun penggunaanya agak sedikit berbeda. Perhatikan contoh dibawah ini:

```
public class Segitiga{
   int alas;
   int tinggi;
   public static void main(String args[]){
      /*
      Pendeklarasian variabel dengan tipe data
      class Segitiga
      */
      Segitiga s3;
```

```
/*
      Instantiate class Segitiga menjadi object
      */
      s3 = new Segitiga();
/*
      Setelah proses instantiate ini, anda dapat
      mengakses object Segitiga melalui variabel
      s3
      */
      s3.alas=10;
      s3.tinggi=2;
      System.out.println("Alas = "+s3.alas);
      System.out.println("Tinggi = "+s3.tinggi);
   }
}
```

Skrip 2.5 Tipe Data Referensi

2.4.4.3 Nilai Literal

Literal adalah suatu nilai yang terlihat secara eksplisit. Perhatikan contoh berikut ini:

```
int index = 10;
```

Pada kode di atas, angka 10 merupakan nilai literal.

1. Literal Integer

   Semua angka yang berupa bilangan bulat yang anda tulis di dalam program akan dikenal sebagai tipe data **int** oleh Java. Contohnya 1,2,3,10,40,150 dan seterusnya. Pada contoh ini nilai literal **int** ditulis berupa bilangan berbasis 10 (desimal). Anda dapat juga menulis dalam basis 8 (oktal) atau 16 (heksadesimal). Untuk menulis nilai literal dalam bentuk oktal, anda dapat menulisnya dengan angka nol (0) di depannya, contoh: 01, 034 ,0425. Untuk bilangan heksadesimal, anda dapat menulisnya dengan didahului oleh tanda 0x (nol dan x). contoh 0x1, 0x234, 0xA dan seterusnya.

   Untuk menuliskan angka yang memiliki ukuran yang sangat besar, dimana hanya tipe data **long** (64 bit) yang dapat menampungnya, Anda perlu menambahkan karakter L atau l di akhir nilai literal yang anda ketik, untuk memberitahu Java bahwa nilai literal tersebut bertipe data **long** dan bukan bertipe **int.** Contoh:

   0xfffffffffffff**L** dan 9234857892347523457**L**.

2. Literal Floating-Point

   Bilangan *floating-Point* merupakan bilangan desimal yang berupa pecahan. Bilangan ini dapat ditulis menggunakan notasi standar biasa atau menggunakan notasi ilmiah (*scientific notation*). Notasi standar

menggunakan titik untuk menandakan pecahan. Contohnya : 10.2, 11.3 dan seterusnya. Sementara untuk notasi ilmiah dapat menggunakan lambang E**n** (eksponensial). Contohnya:

$7.02345E3 = 7.02345 \times 10^3 = 7023.45$

3. Literal Boolean

   Untuk tipe data **boolean**, Java hanya mengenal dua niali literal yaitu **true** dan **false**. Contoh penggunaan :

   ```
   Boolean isEmpty = true;
   ```

4. Literal Karakter

   Java didesain untuk menjadi bahasa yang portable dan universal. Karena itu, Java mendukung penggunaan *Unicode Characters,* yang mencakup hampir semua karakter yang dikenal oleh manusia dalam berbagai bahasa seperti *Arabic*, *Latin*, *Greek*, dan lain – lain. Untuk itu lebar data tipe data **char** adalah 16 bit agar dapat merepresentasikan semua karakter yang ada. Contoh:

   ‘a’ untuk karakter a

   ‘\ddd’ untuk karakter a (oktal)

   ‘\x0061’ untuk karakter a (heksadesimal)

   Berikut ini daftar *escape sequence* yang dikenal oleh Java

Escape Sequence	**Keterangan**
\ddd	Karakter Oktal (ddd)
\uxxxx	Karakter Heksadesimal (xxxx)
\’	Tanda petik satu
\”	Tanda petik dua
\\	Backslash
\r	Carriage return
\n	Pindah baris
\f	Form feed
\t	Tab
\b	backspace

Tabel 2.2 Escape Sequence

5. Literal String

   Literal untuk String dalam Java ditulis di antara tanda petik ganda, dan sama halnya seperti literal untuk karakter, escape sequence juga dapat digunakan di sini. Contoh literal String :

   “Literal String”

"Baris 1\nBaris 2"

"\"Tanda Petik Ganda\" dan tanda backslash \\"

2.4.5 Array

Array Satu Dimensi

Array adalah sebuah set variabel yang diberi nama tertentu yang memiliki tipe data yang sama. Tiap variabel di dalam array disebut elemen, dimana tiap elemen memiliki indeks dengan tipe integer.

Berikut ini contoh deklarasi array:

```
int[] nilaiSiswa; //cara ini lebih dianjurkan
int nilaiSiswa[];
float[] jumlahPanen;
```

Array yang sudah dideklarasikan perlu didefinisikan, seperti contoh berikut:

```
nilaiSiswa = new int[50];
jumlahPanen = new float[100];
```

Untuk memberikan nilai kepada sebuah elemen array caranya dengan menyebutkan nama array yang diikuti indeks dan nilai yang diberikan, seperti contoh berikut:

```
nilaiSiswa[0] = 8;
jumlahPanen[2] = 223.66;
```

Agar lebih jelas, silakan perhatikan contoh kelas berikut:

```
public class ArrayAverage{
   public static void main(String[] args){
      int[] values = {1, 3, 5, 7, 9};
      int aveValue;
      aveValue = getAverage(values);
      System.out.println("Jumlah total: " + aveValue);
   }
   public static int getAverage(int[] intArray){
      int sum = 0;        // variabel lokal
      for(int x = 0; x < intArray.length; x++)
      sum = sum + intArray[x];
      return sum / intArray.length;
   }
}
```

Skrip 2.6 Array Satu Dimensi

Array Multidimensi

Cara pendeklarasian array multidimensi ini pada dasarnya sama dengan array satu dimensi di mana anda cukup menambahkan **[]** sesuai dengan dimensi yang anda inginkan. Contoh;

```
int[][] arr2;       //array 2 dimensi
int[][][] arr3;     //array 3 dimensi
int[][][][] arr4;   //array 4 dimensi
```

Untuk pengalokasian memori array multidimensi, memiliki sintaks yang sama dengan array satu dimensi. Contoh:

```
int[][] arr2 = new int[3][4];
```

Kode di atas akan mengalokasikan memori untuk menampung nilai tipe data **int** sebanyak 3*4.

2.4.6 Konversi Tipe Data dan Casting

Java akan melakukan konversi tipe data secara otomatis jika kedua tipe data tersebut kompatibel. Misalnya dari tipe data **int** ke tipe data **long**. Contoh:

```
int data1 = 10;
long data2 = data1;
```

Variabel data1 yang bertipe int ditampung ke dalam variabel data2 yang bertipe long, dengan demikian telah terjadi konversi tipe data. Dalam hal ini konversi dilakukan secara langsung oleh Java.

Namun tidak semua tipe data kompatibel satu dengan yang lainnya, misalnya tipe float dengan tipe data int. Float merupakan tipe data pecahan sedangkan int adalah tipe data bilangan bulat. Hal yang sama juga terjadi apabila kita mengkonversi tipe data yang lebih besar ke tipe data yang lebih kecil ukurannya. Misalnya dari tipe data int ke tipe data short.

Konversi tipe data yang tidak kompatibel dapat kita lakukan secara eksplisit yaitu dengan *casiting*. Sintaks kode;

(target tipe-data) nilai

Contoh kode;

```
Float data = 10.2F;
Int data2 = (int)data1; //casting dari float ke int
Int data3 = 257;
Byte data4 = (byte)data3. //casting dari int ke byte
```

Yang perlu diperhatikan di sini adalah jika Anda mengubah tipe data yang berbeda jenis, seperti dari tipe data pecahan ke tipe data bilangan bulat maka akan terjadi pemotongan. Pada contoh di atas maka data2 akan bernilai 10. Sedangkan untuk tipe data yang lebih kecil jika digunakan untuk menampung tipe data yang lebih besar dari daya tampungnya maka yang akan tertampung adalah nilai modulusnya(sisa bagi). Pada contoh di atas, tipe data variabel data4 adalah byte (jumlah maksimum yang dapat ditampung oleh byte adalah 256), sedangkan nilai

yang hendak ditampung adalah 257. Dari perhitungan 257/256 dipeoleh modulus=1 maka data4 akan bernilai 1.

Konveri tipe data terkadang dapat menimbulkan kebingungan dan kesalahan logic yang sulit dicari penyebabnya. Perhatikan contoh berikut ini:

```
int nilai = 26;
double hasil = nilai/4;
```

Jika kode di atas di eksekusi maka sisi dari variabel hasil adalah 6.0 dan bukan 6.5. Ini disebabkan karena variabel nilai bertipe int sehingga hasil operasi nilai/4 adalah juga bertipe data int yang hanya dapat menampung bilangan bulat saja. Sehingga hasil yang didapat dari nilai/4 adalah 6 dan bukan 6.5.

Untuk menghindari kejadian seperti ini, sebaiknya kita menggunakan tipe data pecahan (double dan float) untuk operasi yang bisa mengahasilkan bilangan pecahan. Atau anda dapat melakukan casting ke tipe data pecahan pada salah satu operan yang terlibat pada operasi tersebut. Contoh kode:

```
int nilai = 26;
double hasil = (double)nilai/4;
```

2.5 Operator

2.5.1 Operator Assignment

Operator	Deskripsi	Contoh
=	Pemberian nilai	nilai=total;
++	Increment , kemudian beri nilai	jumlah++;
--	Decrement , kemudian beri nilai	Jumlah--;
+=	Tambah kemudian beri nilai	Jumlah+=total;
-=	Kurangi kemudian beri nilai	Jumlah-=total;
*=	Kalikan kemudian beri nilai	Jumlah*=total;
/=	Bagi kemudian beri nilai	Jumlah/=total;
%=	Ambil hasil pembagian kemudian beri nilai	Jumlah%=total;

2.5.2 Operator Aritmatika

Operator	Deskripsi	Contoh
+	Penjumlahan	nilai=ujian+ulangan;
-	Pengurangan	Harga=total-diskon;
*	Perkalian	Total=jumlah*harga;
/	Pembagian	Persen=nilai/100;
%	Hasil bagi	Ganjil=nilai%2;

2.5.3 Operator Relasi

Operator	Deskripsi	Contoh
>	Lebih besar dari	Nilai>rerata
<	Lebih kecil dari	Nilai<rerata
>=	Lebih besar atau sama dengan	Nilai>=rerata

<=	Lebih kecil atau sama dengan	Nilai<=rerata
!=	Tidak sama dengan	Nilai!=rerata
==	Tepat sama dengan	Status==’A’

2.5.4 Operator Logika

Operator	Deskripsi	Contoh
!	NOT	!jikaAngka
&&	AND	JikaAngka && jikaTgl
||	OR	JikaAngka || jikaTgl
==	Sama dengan	x == 3

2.5.5 Operator Bitwise (Manipulasi Bit)

Operator	Deskripsi	Contoh
~	complement	~a
&	AND	a&b
|	OR	a|b
^	Ekslusif OR	a^b
>>	Geser kanan	a>>b
<<	Geser kiri	a<<b
>>>	Geser kanan tanpa mempertahankan sign	a>>>b

2.5.6 Operator ?: (if-then-else)

Operator ini dapat digunakan untuk menggantikan beberapa instruksi yang menggunakan **if-then-else** (akan dibahas pada bab selanjutnya). Secara umum penggunaannya mengikuti bentuk berikut:

```
ekspresi1?ekspresi2:ekspresi3
```

Di mana ekspresi1 harus menghasilkan nilai true atau false. Jika ekspresi1 bernilai true maka ekspersi2 akan dieksekusi oleh Java, dan sebaliknya jika ekspresi1 bernilai false maka ekspresi3 yang akan dieksekusi oleh Java. Baik ekspresi2 dan ekspresi3 harus mengembalikan tipe data yang sama dan tida boleh mengembalikan void. Contoh penggunaan:

```
hasil = akhir?x/10:x*20;
```

2.6 Kontrol Eksekusi Program

2.6.1 Percabangan

2.6.1.1 if

Pernyataan if digunakan untuk menguji suatu kondisi kemudian mengerjakan pernyataan yang lain sesuai hasil pengujian. Bentuk pernyataan if adalah:

```
if(ekspresi)
      pernyataan;
```

Contoh:

```
if(angka % 2 != 0) ++angka;
if(nilai<10)
   System.out.println (“kurang dari 10”);

if(nilai<51){
   System.out.println(“Tidak lulus”);
   status=false;
}
```

Skrip 2.7 Contoh if

Untuk menambahkan pernyataan yang akan dijalankan jika kondisi uji salah, maka dapat ditambahkan klausa **else**, seperti contoh berikut:

```
if(nilai<51){
   System.out.println(“Tidak lulus”);
   status=false;
}else{
   System.out.println(“Lulus”);
   status=true;
}
```

Skrip 2.8 Contoh if-else

Untuk melakukan pengujian lebih dari satu kali, dapat ditambahkan klause else if, seperti contoh berikut:

```
if(nilai<51){
   System.out.println(“Tidak lulus”);
   status=false;
}else if((nilai>51) && (nilai<71)){
   System.out.println(“Lulus, status=cukup”);
   status=true;
}else {
   System.out.println(“Lulus, status=memuaskan”);
   status=true;
}
```

Skrip 2.9 Contoh if-elseif

2.6.1.2 switch

Pernyataan switch digunakan untuk menguji beberapa pilihan berdasarkan beberapa nilai tertentu. Ekspresi yang digunakan harus menghasilkan data dengan tipe char, byte, short dan int. Bentuk dari pernyataan switch adalah sebagai berikut:

```
switch(ekspresi){
   case a:
      pernyataan;
      break;
   .....
   default:
      pernyataan;
      break;
}
```

Skrip 2.10 Bentuk Switch

Contoh :

```
switch(status){
   case 1:
      gaji = 500000;
      break;
   case 2:
      gaji = 750000;
      break;
   default:]
      gaji = 300000;
      break;
}
```

Skrip 2.11 Contoh Skrip

2.6.2 Perulangan

2.6.2.1 for

Pernyataan for digunakan untuk melakukan perulangan dengan menentukan kondisi perulangan dan pernyataan increment. Bentuk pernyataan for adalah sebagai berikut:

```
for(inisialisasi ; kondisi; increment){
   pernyataan1;
   pernyataan2;
   ....
}
```

Skrip 2.12 Bentuk for

Contoh:

```
byte nilai=1;
for (int i=0; i<8; i++){
   nilai *= 2;
}
```

Skrip 2.13 Contoh for

2.6.2.2 while

Pernyataan while digunakan untuk melakukan perulangan dengan menentukan kondisi yang menyebabkan perulangan dihentikan. Bentuk pernyataan while adalah sebagai berikut:

```
while(ekspresi){
   pernyataan1;
   pernyataan2;
   ....
}
```

Skrip 2.14 Bentuk while

Contoh:

```
while (int i < 10){
   nilai += 10;
   i++;
}
```

Skrip 2.15 Contoh while

2.6.2.3 do – while

Penggunaan **do-while** ini mirip dengan bentuk while di atas. Perbedaan utamanya hanyalah dua, yaitu:

1. Penyataan yang ingin dieksekusi harus diletakan di dalam blok kode, sekalipun hanya ingin mengeksekusi satu buah penyataan saja.
2. Pengecekan kondisi (true atau false) dilakukan pada bagian akhir sehingga pernyataan yang ada di dalam blok perulangan akan dieksekusi minimal satu kali, sekalipun eksekusi **do-while** pertama kali menemukan kondisi bernilai false.

Penggunaan bentuk do-while mengikuti bentuk berikut ini:

```
do{
   Pernyataan1;
   Pernyataan2;
   ....
}while(kondisi);
```

Skrip 2.16 Bentuk do-while

Jump

1. break

Penggunaan **break** yang utama adalah untuk menghentikan proses perulangan di dalam **while**, **do-while** atau di dalam **for**. Perhatikan contoh program berikut ini:

```
class TestBreak{
   public static void main(String[] args){
      System.out.println(“Sebelum for”);
      for(int x=0;x<10;x++){
         if(x==4)
      break;
         System.out.pritnln(“Nilai x : ”+x);
      }
      System.out.println(“Setelah For”);
  }
}
```

Skrip 2.17 Contoh break

2. continue

Terkadang dalam mengeksekusi suatu kode dalam perulangan anda ingin terus melakukan perulangan, tetapi karena kondisi tertentu anda ingin kode setelah posisi tertentu tersebut tidak dieksekusi. Di sinilah kita dapat menggunakan **continue**. Perhatikan contoh berikut ini:

```
class TestContinue{
   public static void main(String[] args){
      int x=10;
      System.out.println(“Sebelum while”);
      while(x<=50){
      x++;
      if(x%2==0)
         continue;
      System.out.println(“Nilai x : ”+x);
      }
      System.out.println(“Sesudah while”);
   }
}
```

Skrip 2.18 Contoh continue

3. return

Perintah dalam Java terakhir yang dapat dikategorikan sebagai jump adalah perintah **return** yang digunakan di dalam method. Method dipanggil oleh bagian lain dari program. Dengan menggunakan perintah **return**, alur eksekusi dikembalikan ke bagian program yang memanggil method tersebut.

```
public class Orang{
   public String cetakNama(){
      return Hello Nama Saya Hendro!”;
   }
   public statc void main(String[] args){
      Orang org = new Orang();
      System.out.println(“Sebelum panggil method”);
      System.out.println(org.cetakNama());
      System.out.println(“Sesudah panggil method”);
   }
}
```

Skrip 2.19 Contoh return

BAB III
C L A S S

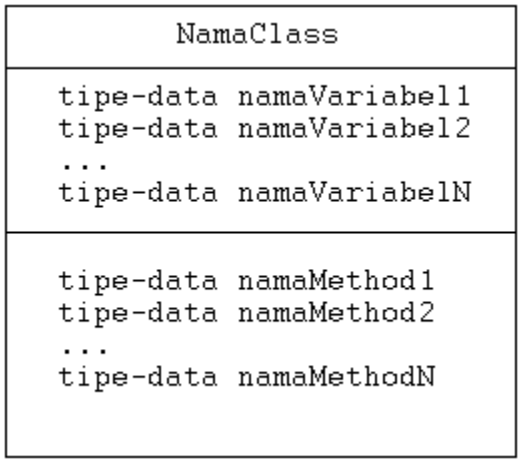
3.1 Pengenalan Dasar Class

Class adalah *blue print* dari objek. Dengan mendeklarasikan suatu class, maka kita telah mendeklarasikan suatu tipe data baru (tipe data referensi). Dengan menggunakan tipe data class ini, kita dapat menciptakan *instance*-nya yang merupakan objek sesungguhnya. Bentuk dasar class adalah sebagai berikut:

```
class NamaClass{
   tipe-data namaVariabel1;
   ….
   ….
   tipe-data namaVariabelN;

   tipe-data namaMethod1(parameter){
      //implementasi method1
   }
   ….
   ….
   tipe-data namaMethodN(parameter){
      //implementasi methodN
   }
}
```

Skrip 3.1 Bentuk Dasar Class

Notasi class dalam UML adalah:



Gambar 3.1 Notasi UML untuk Class

Dari bentuk umum class di atas terlihat isi dari class terbagi menjadi dua bagian besar yaitu deklarasi variabel dan deklarasi method. Variabel yang didefinisikan di dalam class disebut juga *property.*

Berikut ini adalah contoh class yang nyata yang menggambarkan object manusia.

```
class Manusia{
      String nama;
      String jenkel;
      byte usia;
      String alamat;
}
class DemoManusia{
      public static void main(String args[]){
            Manusia m1,m2;
            M1 = new Manusia();
            M2 = new Manusia();

            m1.nama="Hendro";
            m1.jenkel="Laki-laki";
            m1.usia=23;
            m1.alamat="Jl.Kemiri Sari no.7 Salatiga";

            m2.nama="Steven";
            m2.jenkel="Laki-laki";
            m2.usia=23;
            m2.alamat="Jl.Kemiri Sari no.7 Salatiga";

            System.out.println("Nama : "+m1.nama);
            System.out.println("Nama : "+m2.nama);
      }
}
```

Skrip 3.2 Contoh Class

Pada contoh di atas terdapat dua class. Class yang pertama adalah class Manusia dan yang kedua adalah class DemoManusia.

Class Manusia adalah sebuah class sederhana yang merupakan *blue print* dari manusia. Setiap manusia pasti memiliki nama, jenis kelamin, usia dan alamat. Semua ciri – ciri tersebut dijelaskan oleh class Manusia.

Class DemoManusia adalah class yang akan menggunakan class tester atau aplikasi yang akan menggunakan class Manusia. Di dalam class DemoManusia kita mendeklarasikan variabel m1 dan m2 dengan tipe Manusia. Variabel m1 dan m2 ini adalah *instance* dari class Manusia atau apa yang disebut objek (objek sebenarnya).

Yang harus diperhatikan untuk saat ini adalah bagaimana cara menciptakan sebuah objek seperti m1 dan m2 di atas. Pertama yaitu mendeklarasikan variabel yang akan digunakan untuk memegang referensi ke objek, dan yang kedua adalah membuat object/*instance*-nya.

3.2 Variabel atau Property

Property pada suatu class dapat berupa instace variabel, class variabel, constant variabel (konstanta). Untuk lebih jelasnya dapat dilihat pada sub bab sebelumnya (2.4 Variabel, Konstanta dan Tipe Data).

3.3 Method

Pada bahasa – bahasa pemograman yang lain method disebut juga *fuction* atau *procedure.* Dalam pemograman berorientasi object method adalah suatu operasi atau kegiatan yang dapat dilakukan suatu object. Misalnya Manusia memiliki kegiatan seperti makan, tidur, minum dan lain – lain. Kegiatan – kegiatan inilah yang disebut method.

Perhatikan contoh berikut ini:

```
class Manusia{
   String nama;
   boolean isKenyang;
   int jumPiring;

   void setNama(String _nama){
      nama = _nama;
   }
   String getNama(){
      return nama;
   }
   void makan(){
      cekJumPiring();
      if(isKenyang==false){
         System.out.println(nama+" lapar makan ahhh...");
      }else{
         System.out.println(nama+" sudah makan tadi...");
      }
   }

   void cekJumPiring(){
      jumPiring = jumPiring + 1;
      if(this.jumPiring>3)
         isKenyang = true;
   }
}
class DemoManusia{
      public static void main(String args[]){
            Manusia m1 = new Manusia();
            m1.setNama(“Hendro”);
            m1.makan();
            m1.makan();
            m1.makan();
            m1.makan();
      }
}
```

Skrip 3.3 Contoh Method

Dari contoh di atas dapat kita lihat bentuk umum pendeklarasian method adalah sebagai berikut:

```
tipe-data namaMethod(daftar-parameter){
      //implementasi
}
```

3.4 Method Overloading

Dalam Java kita boleh memiliki lebih dari satu method yang memiliki nama sama. Inilah yang disebut *method overloading*. Walaupung Java mengizinkan memiliki nama method sama lebih dari satu, tetapi daftar parameter yang digunakan haruslah berbeda untuk masing – masing method. Karena Java akan menggunakan parameter – parameter ini untuk menentukan method mana yang akan dieksekusi. Perhatikan contoh program di bawah ini.

```
class Manusia{
  String nama;
  String jenkel;

  void setNilai(String param1){
        nama = param1;
  }

  void setNilai(String param1,String param2){
        nama = param1;
        jenkel = param2;
  }

  void cetak(){
        System.out.println(nama+" adalah "+jenkel);
  }
}
class DemoManusia{
      public static void main(String args[]){
            Manusia m1,m2;
            m1 = new Manusia();
            m2 = new Manusia();

            m1.setNilai("Hendro");
            m2.setNilai("Hendro","Laki-laki");

            m1.cetak();
            m2.cetak();

      }
}
```

Skrip 3.4 Method Overloading

3.5 Konstruktor

Konstruktor merupakan method khusus yang digunakan untuk menginisialisasi objek dan tiap class boleh memiliki lebih dari satu konstruktor.

Perbedaan method biasa dengan konstruktor adalah bahwa konstruktor harus memiliki nama yang sama dengan nama classnya dan tidak memiliki nilai kembalian (tipe-data).

Konstruktor dijalankan pada saat sebuah object diinisialisasi (menggunakan kata **new**). Pada konstruktor juga berlaku overloading, artinya boleh mendeklarasikan lebih dari satu konstruktor, asalkan memiliki parameter yang berbeda – beda.

Perhatikan contoh di bawah ini:

```
class Manusia{
  String nama;
  String jenkel;

  Manusia(){  //konstruktor
        nama = "unknown";
        jenkel = "unknown";
  }

  Manusia(String param1,String param2){ //konstruktor
        nama = param1;
        jenkel = param2;
  }
void cetak(){
        System.out.println("Nama : "+nama);
        System.out.println("Jenis Kelamin : "+jenkel +"\n");
  }
}

class DemoManusia{
      public static void main(String args[]){
            Manusia m1,m2;
            m1 = new Manusia();
            m2 = new Manusia("Hendro","Laki-laki");

            m1.cetak();
            m2.cetak();
      }
}
```

Skrip 3.5 Konstruktor

Yang perlu diperhatikan adalah apabila anda tidak mendeklarasikan satu pun konstruktor, maka Java secara otomatis menambahkan konstruktor default ke dalam class yang kita buat walaupun tidak kelihatan pada kode program. Apabila kita mendeklarasikan satu atau lebih konstruktor maka java tidak akan menambahkan kostruktor default.

Saat kita menginisialisasi suatu objek menggukan perintah **new**, maka pasti pasti salah satu konstruktor yang kita buat akan dijalankan. Konstruktor mana yang dijalankan tergantung dari parameter yang kita lewatkan (sama dengan konsep *method overloading*).

3.6 Penggunakan Keyword ‘this’

Terkadang dalam suatu method kita ingin menunjuk ke objek di mana method ini berada. Untuk itu kita menggunakan keyword **this**. Perhatikan Contoh berikut.

```
class Manusia{
   String nama;
   void setNama(String nama){
      this.nama = nama;
   }
}
```

Skrip 3.6 Keyword 'this'

3.7 Penggunaan Keyword 'static'

Apabila kata kunci static kita tempatkan pada pendeklarasian member (variabel dan method) dari suatu class, maka member class tersebut dapat diakses tanpa harus menciptakan objek class tersebut. Untuk lebih jelasnya perhatikan contoh berikut ini.

```
class Manusia{
  static String nama;
  static String jenkel;

  static void cetak(){
        System.out.println("Nama : "+nama);
        System.out.println("Jenis Kelamin : "+jenkel +"\n");
  }
}

class DemoManusia{
      public static void main(String args[]){
            Manusia.nama = "Hendro";
            Manusia.jenkel = "Laki - laki";
            Manusia.cetak();
      }
}
```

Skrip 3.7 Penggunaan keyword 'static'

3.8 Modifier

Penggunaan *modifier* berfungsi untuk melakukan enkapsulasi (membungkus data) pada objeck. Dengan menggunakan *modifier* kita dapat menentukan siapa saja yang boleh menggunakan atau mengakses member dari suatu objek.

Ada empat macam modifier yang dikenal oleh Java, yaitu **private**, **protected**, **public** dan **tanpa *modifier***.

3.8.1 Class Modifier

Bentuk penggunaan modifier pada class:

```
modifier class NamaClass{
   …
   …
}
```

Modifier	Keterangan
(default)	Class visible atau dapat digunakan hanya pada package yang sama
public	Class visible terhadap semua package yang berbeda – beda
final	Class tidak dapat diturunkan lagi / extends

Tabel 3.1 Modifier pada Class

Contoh penggunaan:

```
public class Manusia{
 ....
 ....
}
```

Apabila sebuah class menggunakan modifier public maka class tersebut harus disimpan dengan nama file yang sama dengan nama classnya. Seperti pada contoh di atas, maka class Manusia harus disimpan dengan nama Manusia.java.

3.8.2 Method Modifier

Bentuk penggunaan method modifier:

```
modifier tipe-data namaMethod(parameter){
   …
   …
}
```

Berikut ini adalah daftar modifier yang dapat digunakan pada method.

Modifier	Keterangan
(default)	Method visible atau dapat digunakan hanya pada package yang sama
public	Method visible pada semua package
private	Method visible hanya di dalam class itu sendiri
protected	Method visible didalam package dan sub classnya
static	Lihat sub bab sebelumnya
final	Method tidak dapat diubah / dioverride pada subclass (dibahas pada bab selanjutnya)
abstract	Method harus dioverride / didefinisikan pada subclassnya (dibahas pada bab selanjutnya)

Contoh program :

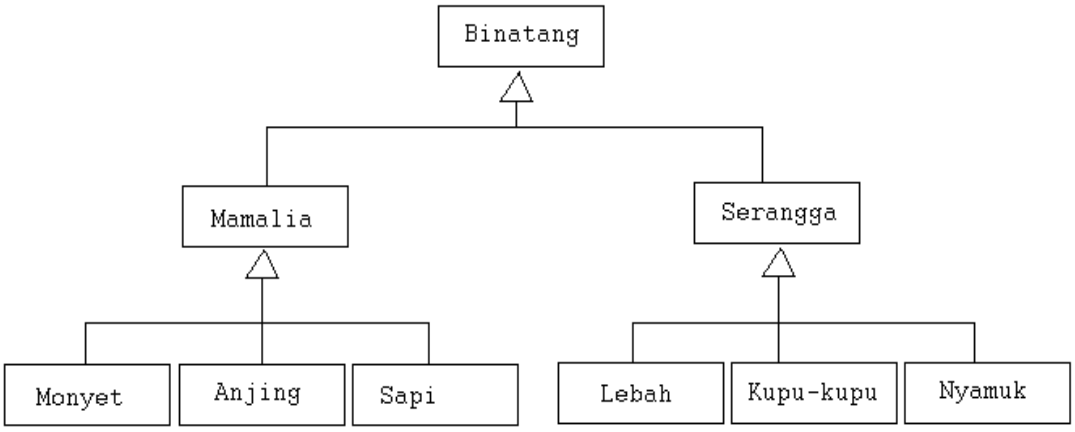
```
public class Manusia{
  private String nama;
  private String jenkel;

  public void setNama(String nama){
        this.nama=nama;
  }
  public void setJenkel(String jenkel){
        this.jenkel=jenkel;
  }
  public void cetak(){
        System.out.println("Nama : "+nama);
        System.out.println("Jenis Kelamin : "+jenkel);
  }
}


public class DemoManusia{
      public static void main(String args[]){
            Manusia m = new Manusia();
            m.setNama("Hendro");
            m.setJenkel("Laki-laki");
            m.cetak();
      }
}
```

Skrip 3.8 Method Modifier

BAB IV
OBJECT ORIENTED PROGRAMMING DENGAN JAVA

Inheritance / Pewarisan

Sekeliling kita semuanya adalah objek. kita sendiri adalah objek, meja, kursi, mobil, pohon dan lain – lain adalah objek. Kalo kita perhatikan objek – objek ini memiliki hierarki berdasarkan jenisnya masing – masing. Untuk jelasnya kita lihat hierarki class dari objek binatang di bawah ini.



Gambar 4.1 Contoh hierarki class

Kalo kita perhatikan hierarki class di atas, semakin ke bawah semakin spesific dan sebaliknya semakin ke atas semakin umum. Apa yang dimikili class Binatang, juga akan dimiliki oleh Mamalia dan Serangga dan semua subclass di bawahnya. Dengan konsep seperti ini, karakteristik yang dimiliki oleh class Binatang cukup didefinisikan pada class Binatang saja. Class Mamalia dan Serangga berserta semua subclass di bawahnya tidak perlu mendefinisikan ulang apa yang telah dimiliki oleh class Binatang, tetapi hanya mendefinisikan karakteristik yang spesifik yang dimilikinya.

Pewarisan pada Java

Untuk menerapkan konsep pewarisan, Java menyediakan keyword **extends** yang dapat dipakai pada waktu mendeklarasikan suatu class. Contoh berikut merupakan implementasi dari dari hierarki class Binatang di atas.

```
public class Binatang {
   protected String kind;
   protected String makanan;
   public void makan(){
      System.out.println(this.kind+" makan "+this.makanan);
   }
}
public class Mamalia extends Binatang {
      public void menyusui(){
            System.out.println(this.kind + "menyusui");
      }
}
public class Serangga extends Binatang{
      public void terbang(){
            System.out.println(this.kind+" terbang");
      }
}
public class Monyet extends Mamalia{
      public void manjatPohon(){
            System.out.println(this.kind+" manjat pohon");
      }
}
public class Anjing {
      public void menggonggong(){
            System.out.println("Ooow...Oooowww...oooww!!!");
      }
}
public class Sapi {
      public void melenguh(){
            System.out.println("OOOOOO.....!!!!!");
      }
}
public class Kupukupu extends Serangga {
      public void metamorfosis(){
            System.out.println(this.kind+" bermetamorfosis");
      }
}
public class Nyamuk extends Serangga{
      public void bertelur(){
            System.out.println(this.kind+" bertelur jentik");
      }
}
public class Lebah extends Serangga{
      public void menyengat(){
            System.out.println(this.kind+" Meyengat");
      }
}
```

Skrip 4.1 Contoh Pewarisan

Di dalam Java semua class yang kita buat sebenarnya adalah turunan atau subclass dari class **Object**. Class Object merupakan class tertinggi dari semua hierarki class dalam Java.

Method Overriding

Jika pada subclass kita menulis ulang method yang ada pada super classnya, maka method yang ada di subclass tersebut disebut meng-override method super classnya. Jadi ketika kita memanggil method tersebut dari objek subclassnya maka yang akan dijalankan adalah method yang berada di subclass tersebut. Untuk lebih jelasnya perhatikan contoh berikut.

```
public class A {
	public void cetak(){
		System.out.println("Dicetak oleh class A");
	}
}
public class B extends A {
	public void cetak(){
		System.out.println("Dicetak oleh class B");
	}
}

public class TestOverride {
	public static void main(String[] args){
		B objB = new B();
		objB.cetak();
	}
}
```

Skrip 4.2 Contoh Method Override

Keyword 'final' dalam Pewarisan

Pada sub bab sebelumnya penggunaan kata **final** adalah untuk mendeklarasikan suatu konstanta. Apabila kata **final** digunakan pada pendeklarasian method, maka method tersebut tidak dapat dioverride. Ubahlah method pada class A di atas menjadi:

```
public class A {
	public final void cetak(){
		System.out.println("Dicetak oleh class A");
	}
}
```

Skrip 4.3 Keywor final pada method

Apabila kita jalankan program sebelumnya, maka akan terjadi error karena method yang final tidak dapat dioverride lagi.

Selanjutnya apabila kita menggunakan kata final pada deklarasi class, akan menyebabkan class tersebut tidak dapat diturunkan atau membuat subclassnya menggunakan keyword **extends**.

Polimorfisme pada pewarisan

Polimorfisme bisa diartikan satu bentuk banyak aksi. Dalam pewarisan polimorfisme dapat kita lakukan. Kita data memerintah sebuah objek untuk melakukan tindakan yang secara prinsip sama tapi secara proses berbeda. Untuk lebih jelasnya perhatikan contoh berikut ini.

```
public class Mobil {
      public void injakPedalGas(){
            System.out.println("Mobil berjalan...");
      }
}
public class Kijang extends Mobil{
      public void injakPedalGas(){
            System.out.println("Mobil Melaju dengan kecepatan
80 Km/jam...");
      }
}
public class Escudo extends Mobil {
   public void injakPedalGas(){
    System.out.println("Mobil Melaju dengan kecepatan 70
Km/jam...");
      }
}
public class BMW extends Mobil{
      public void injakPedalGas(){
            System.out.println("Mobil Melaju dengan kecepatan
100 Km/jam...");
      }
}

public class TestMobil {
      public static void main(String[] args){
            Mobil mobil = new Mobil();
            Kijang kijang = new Kijang();
            Escudo escudo = new Escudo();
            BMW bmw = new BMW();

            mobil.injakPedalGas();

            mobil = kijang;
            mobil.injakPedalGas();

            mobil = escudo;
            mobil.injakPedalGas();


            mobil = bmw;
            mobil.injakPedalGas();
      }
}
```

Skrip 4.4 Polimorfisme dalam Pewarisan

Class Abstrak

Class Abstrak tidak berbeda dengan class – class lainnya yaitu memiliki class members (method dan variabel). Sebuah class adalah abstrak jika salah satu methodnya dideklarasikan abstrak. Method abstrak adalah method yang tidak memiliki implementasi. Contoh deklarasi method abstrak:

```
abstract public void cetak();
```

Beberap hal yang perlu diperhatikan adalah sebagai berikut:

1. Class abstrak tidak dapat dibuatkan instan atau objeknya menggunakan keyword **new**.
2. Sebuah class dapat dideklarasikan sebagai class abstrak walaupun tidak memiliki method abstrak.
3. Variabel dengan tipe class abstrak tetap bisa diciptakan, tetapi harus *refer* ke subclass dari class abtrak tersebut yang tentunya tidak abstrak.

Perhatikan contoh class abstrak di bawah ini.

```
abstrak public class Mobil {
      public void injakPedalGas();
      public void injakRem(){
         System.out.println(“Mobil berhenti!”);
      }
}
public class Kijang extends Mobil{
      public void injakPedalGas(){
            System.out.println("Mobil Melaju dengan kecepatan
80 Km/jam...");
      }
}
public class BMW extends Mobil {
   public void injakPedalGas(){
    System.out.println("Mobil Melaju dengan kecepatan 100
Km/jam...");
      }
}
```

Skrip 4.5 Class Abstrak

Objek class abtrak tidak dapat diciptakan menggunakan keyword **new** secara langsung. Apabila kita terpaksa ingin menciptakan object class abtrak tanpa membuat subclass kongkritnya, maka kita harus mengimplementasikan method – method abstraknya secara langsung saat deklarasi. Perhatikan contoh di bawah ini.

```
public class TestMobil {
      public static void main(String[] args){
            Mobil mobil = new Mobil(){
                  public void injakPedalGas(){
                        System.out.println("Mobil
berjalan...");
                  }
            };

            Kijang kijang = new Kijang();
            Escudo escudo = new Escudo();
            BMW bmw = new BMW();

            mobil.injakPedalGas();

            mobil = kijang;
            mobil.injakPedalGas();

            mobil = escudo;
            mobil.injakPedalGas();

            mobil = bmw;
            mobil.injakPedalGas();
      }
}
```

Skrip 4.6 Deklarasi objek class abstrak

Interface

Interface adalah class yang hanya mengandung deklarasi method tanpa memiliki implementasi dan semua property yang dimilikinya bersifat final. Interface mirip dengan class abstrak, tetapi interface tidak terikat dengan class hierarki.

Berikut ini adalah aturan yang harus kita ingat tentang pendeklarasian interface:

1. Modifier yang digunakan hanya **public** atau tidak sama sekali. Jika tidak menggunakan modifier maka interface tersebut hanya dapat diakses dalam package yang sama.
2. Semua variabel yang dideklarasikan dalam interface secara otomatis adalah static final. Karena itu waktu pendeklarasian harus diberikan nilai.
3. Semua method adalah abstrak. Bedanya dengan class abstrak adalah kita tidak perlu menuliskan *keyword* **abstract** pada saat mendeklarasikan method dalam interface.
4. Kita dapat mengimplementasikan lebih dari satu interface (multiple inheritance) dengan memisahkan nama dari setiap interface dengan tanda koma.
5. Dapat terjadi saat kita mengimplementasikan lebih dari satu interface ternyata interface – interface tersebut memiliki method yang sama. Dalam hal ini method yang akan diimplementasi adalah method yang berada pada posisi pertama.
6. Semua method yang diimplemetasikan harus **public**.

7. Jika kita tidak mengimplementasikan semua method yang ada pada interface, maka class tersebut harus dideklarasikan sebagai **abstract** class.

Untuk lebih jelasnya kita akan mengambil contoh pada dunia nyata. Misalnya pembuat remote control dan pembuat TV. Seorang pembuat remote control tentunya tidak perlu tahu merk TV dan siapa yang membuat TV tersebut untuk dikontrol menggunakan remote control tersebut. Yang perlu diketahui adalah Interface apa yang digunakan oleh TV tersebut agar dapat dikontrol. Demikian pulu si pembuat TV tidak perlu tahu bagaimana remote control tersebut dibuat. Pembuat TV hanya perlu tahu interface apa yang akan digunakan agar Tvnya dapat dikontrol oleh remote control tersebut. Perhatikan implementasinya dalam kode berikut ini.

```
public interface Control {
      public void pindahChannel(int channel);
      public void perbesarVolume(int intensitas);
      public void perkecilVolume(int intensitas);
}
```

```
public class TVPolitron implements Control{
      String[] channel =
{"RCTI","SCTV","INDOSIAR","ANTV","TV7"};
      public void pindahChannel(int channel) {
            System.out.println("Pindah channel pada TV Politron
ke "+ this.channel[channel]);
      }

      public void perbesarVolume(int intensitas) {

            System.out.println("Perbesar volume pada TV
Politron sebanyak "+ intensitas);
      }

      public void perkecilVolume(int intensitas) {
                        System.out.println("Perkecil volume
pada TV Politron sebanyak "+ intensitas);
      }

}
```

```
public class TVSamsung implements Control{
      String[] channel =
{"RCTI","SCTV","INDOSIAR","ANTV","TV7"};

      public void pindahChannel(int channel) {
            System.out.println("Pindah channel pada TV Samsung
ke "+ this.channel[channel]);
      }
public void perbesarVolume(int intensitas) {
            System.out.println("Perbesar volume pada TV Samsung
sebanyak "+ intensitas);
      }
```

```
public void perkecilVolume(int intensitas) {
            System.out.println("Perkecil volume pada TV Samsung
sebanyak "+ intensitas);
      }
}
```

```
public class RemoteControl {
      public void kirimPerintahKeTv(int aksi,Control tv,int
tombol){
            switch(aksi){
                  case 1:
                        tv.pindahChannel(tombol);
                        break;
                  case 2:
                        tv.perbesarVolume(tombol);
                        break;
                  case 3:
                        tv.perkecilVolume(tombol);
                        break;
            }
      }
}
```

```
public class TestRemoteControl {
      public static void main(String[] args){
            TVPolitron tvp = new TVPolitron();
            TVSamsung tvs = new TVSamsung();
            RemoteControl rc = new RemoteControl();

            rc.kirimPerintahKeTv(1,tvp,1);
      }
}
```

Skrip 4.7 Contoh Interface

Package

Package adalah cara untuk mengelompokan class dan interface yang ada ke dalam kelompoknya (*name space*) masing – masing sehingga lebih mudah diatur dan memungkinkan penggunaan nama yang sama.

Untuk mendefinisikan suatu package digunakan *keyword* **package**. Pendefinisian nama package harus terletak di bagian paling atas dari source program kita. Sintaks pendefinisian nama package adalah sebagai berikut:

```
package namaPackage;
```

Contoh:

```
package siswa;
```

Java menggunakan package seperti struktur direktori. Oleh karena itu semua class atau interface yang memiliki definisi package seperti contoh di atas, harus disimpan pada direktori bernama siswa.

Kita juga dapat membuat package secara hierarki layaknya struktur direktori.

Contoh:

```
package hen.com.contoh;
```

Pada contoh di atas menunjukan bahwa semua class atau interface yang menggunkan deklarasi package ini harus disimpan pada direktori hen -> com -> contoh.

Apabila program kita akan menggunakan sebua class yang terletak pada package yang berbeda, maka kita harus meng***import***nya agar dapat digunakan.

Contoh:

```
packace control
public interface Control {
      public void pindahChannel(int channel);
      public void perbesarVolume(int intensitas);
      public void perkecilVolume(int intensitas);
}
```

```
package tv;
import control.Control;
public class TVPolitron implements Control{
      String[] channel =
{"RCTI","SCTV","INDOSIAR","ANTV","TV7"};
      public void pindahChannel(int channel) {
            System.out.println("Pindah channel pada TV Politron
ke "+ this.channel[channel]);
      }

      public void perbesarVolume(int intensitas) {

            System.out.println("Perbesar volume pada TV
Politron sebanyak "+ intensitas);
      }

      public void perkecilVolume(int intensitas) {
                        System.out.println("Perkecil volume
pada TV Politron sebanyak "+ intensitas);
      }

}
```

```
package tv;
import control.Control;
public class TVSamsung implements Control{
      String[] channel =
{"RCTI","SCTV","INDOSIAR","ANTV","TV7"};
      public void pindahChannel(int channel) {
            System.out.println("Pindah channel pada TV Samsung
ke "+ this.channel[channel]);
      }
public void perbesarVolume(int intensitas) {
            System.out.println("Perbesar volume pada TV Samsung
sebanyak "+ intensitas);
      }
```

```
public void perkecilVolume(int intensitas) {
            System.out.println("Perkecil volume pada TV Samsung
sebanyak "+ intensitas);
      }
}
```

```
Package remote;
import control.Control;
import tv.*;
public class RemoteControl {
      public void kirimPerintahKeTv(int aksi,Control tv,int
tombol){
            switch(aksi){
                  case 1:
                        tv.pindahChannel(tombol);
                        break;
                  case 2:
                        tv.perbesarVolume(tombol);
                        break;
                  case 3:
                        tv.perkecilVolume(tombol);
                        break;
            }
      }
}
```

```
import remote.RemoteControl;
import tv.*;
import control.Control;
public class TestRemoteControl {
      public static void main(String[] args){
            TVPolitron tvp = new TVPolitron();
            TVSamsung tvs = new TVSamsung();
            RemoteControl rc = new RemoteControl();

            rc.kirimPerintahKeTv(1,tvp,1);
      }
}
```

Skrip 4.8 Contoh Penggunaan Package

Exception

Exception dalam Java didefinisikan sebagai sebuah obyek yang muncul ketika terjadi kondisi tidak normal dalam sebuah program. Untuk menangani exception dalam sebuah program, dapat digunakan pernyataan try, catch, dan finally. Penjelasan ketiga pernyataan tersebut adalah sebagai berikut:

1. try digunakan untuk mendefinisikan pernyataan yang memungkinkan timbulnya exception.
2. catch digunakan untuk menangani exception ketika muncul.
3. finally digunakan untuk menutup proses sebelumnya, dimana pernyataan ini boleh untuk tidak digunakan.

Perhatikan contoh berikut:

```
public class TestTryCatch{
   public static void main(String[] args){
      int i = 1;
      int j = 0;
      try{
          System.out.println("Try block entered " + "i = "+ i +
" j = "+j);
          System.out.println(i/j);
          System.out.println("blok try berakhir");
      }catch(ArithmeticException e) {
           System.out.println("terjadi exception");
      }
      System.out.println("setelah blok try");
      return;
  }
}
```

Skrip 4.9 Contoh Exception

Perhatikan juga contoh berikut:

```
import java.io.IOException;
public class TryBlockTest{
   public static void main(String[] args) throws IOException{
      int[] x = {10, 5, 0};
      try{
           System.out.println("Blok try pertama");
           System.out.println("hasil = " + divide(x,0));
           x[1] = 0;
           System.out.println("hasil = " + divide(x,0));
           x[1] = 1;
           System.out.println("hasil = " + divide(x,1));
      } catch(ArithmeticException e){
           System.out.println("Arithmetic exception”);
      }
```

```
       catch(ArrayIndexOutOfBoundsException e) {
          System.out.println("Index-out-of-bounds exception");
      }
      System.out.println("\nTekan Enter untuk keluar");
      try {
         System.out.println("blok try kedua");
         System.in.read();
         return;
      }catch(IOException e){
         System.out.println("I/O exception ");
      }finally{
         System.out.println("blok finally");
       }
   }
   public static int divide(int[] array, int index){
      try{
         System.out.println("\nblok try pertama");
         array[index + 2] = array[index]/array[index + 1];
         return array[index + 2];
        }catch(ArithmeticException e) {
          System.out.println("Arithmetic exception");
      }catch(ArrayIndexOutOfBoundsException e) {
       System.out.println("Index-out-of-bounds exception");
      }finally{
        System.out.println("blok finally");
       }
      return array[index + 2];
   }
}
```

Skrip 4.10 Contoh Exception

BAB V
JAVA I/O

Pada contoh – contoh sebelumnya sebenarnya kita telah menggunakan salah satau fungsi output pada java, yaitu method **println()** yang ada pada **System.out**. Di dalam Java semua operasi yang berhubungan dengan input dan output sudah disediakan. Librari yang perlu dipelajari yaitu paket **java.io**.

Dalam paket java.io terdapat banyak sekali class – class yang berhubungan dengan input dan output. Oleh karena itu kita tidak mungkin akan membahas semua class tersebut satu persatu. Tetapi anda tidak perlu kawatir karena documentasi Java cukup mudah dipelajari dan hampir semua class pada paket **java.io** cara penggunaannya sederhana.

5.1 Membaca Input dari Keyboard

Untuk mendapatkan inputan dari keyboard kita dapat menggunakan beberapa cara. Yang pertama kita bisa memanfaatkan parameter **String[] args** yang ada pada method main. Perhatikan contoh berikut ini:

```
public class InputArgs {
      public static void main(String[] args){
            System.out.print("Nama Anda : ");
            for(int x=0;x<args.length;x++){
                  System.out.print(args[x]+ " ");
            }
      }
}
```

Skrip 5.1 Contoh Input Data

Kompile program di atas dan jalankan lewat console seperti di bawah ini.

```
C:\java InputNama Hendro Steven
```

Kelemahan pada contoh di atas adalah kita hanya dapat melakukan input pada saat akan menjalankan program, dan pada saat run-time kita tidak dapat melakukan input lagi. Untuk dapat melakukan inputan pada saat run-time, kita dapat menggunakan cara seperti contoh program di bawah ini.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class InputConsole {
public static void main(String[] args) throws IOException{
      BufferedReader input = new BufferedReader(new
      InputStreamReader(System.in));
```

```
            System.out.print("Nama Anda : ");
            String nama = input.readLine();
            System.out.print("Pekerjaan Anda : " +nama);
            String job = input.readLine();
            System.out.println(nama+" adalah "+job);
      }
}
```

Skrip 5.2 Mengambil input saat Runtime

Dengan contoh di atas kita dapat memberikan inputan saat run-time. Jangan lupa untuk mengimport class – class yang kita gunakan.

5.2 Membaca Input Dari File

Salah satu fungsi yang cukup penting dalam pemograman adalah bagaimana membaca isi dari sebuah file. Untuk membaca isi sebuah file tentunya kita masih akan menggunakan pake java.io dan contoh di berikut ini akan menunjukan betapa mudahnya membaca file dengan Java.

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

public class InputConsole {
   public static void main(String[] args) {
      try{
            FileInputStream file = new
            FileInputStream(args[0]);
            BufferedReader input = new BufferedReader(new
InputStreamReader(file));
            System.out.println(input.readLine());
            input.close();
            file.close();
      }catch(IOException e){
            System.out.println(e.getMessage());
      }
   }
}
```

Skrip 5.3 Baca Isi File

Kompile program di atas dan jalankan lewat console seperti berikut ini:

```
C:\java BacaFile data.txt
```

5.3 Menulis Data ke File

Untuk menulis data ke sebuah file tidak jauh berbeda dengan contoh – contoh sebelumnya. Untuk lebih jelasnya perhatikan contoh berikut ini:

```
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
public class TulisFile {
   public static void main(String[] args) {
      try {
         FileOutputStream file = new
FileOutputStream("C:/Data.txt");
         BufferedWriter output = new BufferedWriter(new
OutputStreamWriter(file));
         String data = "Nama Saya Hendro Steven Tampake";
         output.write(data);
         output.close();
         file.close();
         System.out.println("OK");
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      } catch(IOException e){
        e.printStackTrace();
      }
   }
}
```

Skrip 5.4 Menulis File

Di dalam paket java.io masih terdapat banyak class – class yang berhubungan dengan input dan output yang belum kita bahas. Diharapkan anda melakukan eksplorasi secara individu dengan panduan API yang ada.

BAB VI
JAVA SWING

Java Swing adalah librari java yang digunkan untuk menciptakan *Grafik User Interface* (GUI). Dengan Java Swing kita dapat membuat user interface yang *cross platform* atau *OS independent*. Artinya user interface yang kita buat dapat dijalankan pada system operasi apa saja (OS yang suport Java) dengan tampilan yang relative sama. Bahkan kita dapat membuat user interface yang menyerupai Windows XP, Mac OS atau Linux tanpa tergantung dari OS yang kita gunakan.

6.1 Komponen Dasar Swing

Secara umum ada lima bagian dari Swing yang akan sering kita gunakan. Komponen atau bagian – bagian itu adalah:

1. *Top-level Container*, merupakan container dasar di mana komponen lainnya diletakan. Contoh Top-level container ini adalah Frame, Dialog dan Applet yang diimplementasi dalam class JFrame, Jdialog, dan JApplet.
2. *Intermediate Container*, merupakan komponen perantara di mana komponen lainnya akan diletakan. Salah satu contoh container ini adalah class JPanel.
3. *Atomic Component*, merupakan komponen yang memiliki fungsi spesifik dan biasanya user berinteraksi langsung dengan komponen jenis ini. Contohnya adalah JButton, JLabel, JTextField, dan JTextArea.
4. *Layout Manager*, berfungsi untuk mengatur bagaimana posisi dari komponen – komponen yang diletakan pada container. Secara default terdapat 5 macam layout yaitu berupa class BorderLayout, BoxLayout, FlowLayout, GridBagLayout, dan GridLayout.
5. *Event Handling*, untuk menangani event yang dilakukan oleh user misalnya menekan tombol, mengkilik mouse dan lain – lain.

6.2 Membuat Window dengan JFrame

Untuk membuat Window dengan JFrame tentunya kita akan menggunakan class JFrame. Perhatikan contoh berikut ini:

```
import javax.swing.*;
public class TestFrame{
   public static void main(String[] args){
      JFrame frame = new JFrame(“Contoh JFrame”);
      frame.setSize(400,150);
      frame.show();
   }
}
```

Saat kita jalankan program di atas maka pada layar akan ditampilkan window sederhana. Tetapi saat kita menutup window tersebut ternyata program kita tidak benar – benar berhenti. Oleh karena itu kita perlu menambahakan baris perintah lagi sehingga menjadi seperti contoh berikut:

```
import javax.swing.*;
public class TestFrame{
   public static void main(String[] args){
      JFrame frame = new JFrame(“Contoh JFrame”);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(400,150);
      frame.show();
   }
}
```

Skrip 6.1 Window Sederhana

6.3 Menambahkan Teks Field dan Tombol pada Frame

Untuk menambahkan teks field dan tombol pada frame yang telah kita buat tadi cukup sederhana. Untuk jelasnya perhatikan contoh berikut ini:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpleForm {
   public static void main(String[] args){
      JFrame myFrame = new JFrame("Form Sederhana");
      FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
      layout.setVgap(10);
      layout.setHgap(10);

      JTextField txtPesan = new JTextField(20);
      JButton cmdTampil = new JButton("Tampil");
      JButton cmdClose = new JButton("Keluar");

      myFrame.getContentPane().setLayout(layout);
      myFrame.getContentPane().add(txtPesan);
      myFrame.getContentPane().add(cmdOK);
      myFrame.getContentPane().add(cmdClose);

      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.pack();
      myFrame.show();
   }
}
```

Skrip 6.2 Menambahkan Komponen lain pada Window

6.4 Menambahkan event pada Tombol

Pada contoh di atas tombol – tombol yang ada w`alaupun kita click, tidak akan memberikan reaksi apapun. Untuk itu kita butuh membuat event handlernya seperti pada contoh berikut ini.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpleForm {
   public static void main(String[] args){
      JFrame myFrame = new JFrame("Form Sederhana");
      FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
      layout.setVgap(10);
      layout.setHgap(10);

      JTextField txtPesan = new JTextField(20);
      JButton cmdTampil = new JButton("Tampil");
      JButton cmdClose = new JButton("Keluar");
      cmdClose.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                        System.out.println("Keluar");
                        System.exit(0);
                  }
            });
      myFrame.getContentPane().setLayout(layout);
      myFrame.getContentPane().add(txtPesan);
      myFrame.getContentPane().add(cmdOK);
      myFrame.getContentPane().add(cmdClose);

      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            myFrame.pack();
            myFrame.show();
      }
}
```

Skrip 6.3 Event Handling pada Tombol

6.5 Kotak Dialog

Kotak dialog adalah komponen dalam Swing yang digunakan untuk menampilkan pesan. Misalnya pesan kesalahan, input box atau konfirmasi biasa. Untuk contoh berikut ini kita akan menampilkan pesan saat kita menekan tombol Tampil. Pesan yang di tampilkan sesuai dengan inputan yang kita berikan pada txtPesan.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpleForm {
   public static void main(String[] args){
      JFrame myFrame = new JFrame("Form Sederhana");
      FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
      layout.setVgap(10);
      layout.setHgap(10);

      final JTextField txtPesan = new JTextField(20);
      JButton cmdTampil = new JButton("Tampil");
      cmdTampil.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent evt){
                  JOptionPane.showMessageDialog(null,txtPesan.g
                  etText());
         }
      });
      JButton cmdClose = new JButton("Keluar");
      cmdClose.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent evt) {
                  System.out.println("Keluar");
                  System.exit(0);
         }
      });

      myFrame.getContentPane().setLayout(layout);
      myFrame.getContentPane().add(txtPesan);
      myFrame.getContentPane().add(cmdTampil);
      myFrame.getContentPane().add(cmdClose);

      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.pack();
      myFrame.show();
   }
}
```

Skrip 6.4 Menampilkan kotak Dialog

Masih banyak komponen – komponen Swing yang ada dan tidak dapat kita bahas di sini. Diharapkan anda melakukan eksplorasi sendiri untuk komponen – komponen tersebut. Untuk saat ini sudah banyak terdapat tools dan Java IDE yang menyediakan alat bantu untuk membuat GUI berbasis Swing. Anda bisa mencari dari Internet dan salah satunya adalah Jigloo plugin untuk eclipse yang gratis dan cukup baik.

Tentang Penulis

Hendro Steven Tampake adalah pengajar di Fakultas Teknologi Informasi Universitas Kristen Satya Wacana Salatiga, dan juga sebagai trainer pada beberapa training center di jakarta untuk pemrograman Java, PHP, Linux dan teknologi open source lainnya.

Selain itu aktif sebagai pengembang berbagai jenis aplikasi khususnya aplikasi yang dibangun dengan teknologi open source seperti Java dan PHP.

Penulis dapat dihubungi melalui email di hendro.steven@gmail.com