Pertemuan II

DASAR PEMROGRAMAN

2.1. Java *Identifier*

Java *identifier* adalah suatu tanda yang mewakili nama-nama variabel, metode, kelas, paket, dan *interface*. Contoh dari *identifier* adalah : Hello, main, System, out.

Pendeklarasian Java adalah *case-sensitive*, yaitu membedakan huruf besar (kapital) dan huruf kecil. Hal ini berarti bahwa *identifier* : **H**ello tidak sama dengan **h**ello. *Identifier* harus dimulai dengan huruf, *underscore* "_", atau tanda dollar "$". Karakter selanjutnya dapat menggunakan huruf atau nomor 0 sampai 9.

Huruf pada identifier dapat berupa huruf besar (kapital), huruf kecil maupun karakter unicode yang menunjukkan huruf dari suatu bahasa, seperti huruf jerman "ä" (*umlaut*) atau yunani "Π".

Meskipun penamaan *identifier* dapat diawali dengan *underscore* "_" atau tanda dollar "$", tetapi sebaiknya dihindari, karena mungkin digunakan untuk pengolahan *internal* dan file *import*.

Penamaan *identifier* dapat menggunakan karakter yang panjangnya tidak terbatas, tetapi lebih baik menggunakan karakter secukupnya dan merepresentasikan fungsi *identifier*.

Identifier tidak boleh menggunakan symbol-simbol seperti "+", spasi, "@" dan lain sebagainya. *Identifier* juga tidak boleh menggunakan *reserved words* atau *keywords* java. Identifier tidak boleh mengandung spasi atau *white space*.

Identifier tidak dapat menggunakan kata kunci (*keywords*) dalam Java seperti class, public, void, dsb, yang akan dibahas selanjutnya.

2.2. Java *Keywords*

Keywords (kata kunci) adalah *identifier* yang telah dipesan untuk didefinisikan sebelumnya oleh Java untuk tujuan tertentu. Anda tidak dapat menggunakan *keyword* sebagai nama variabel, kelas, metode, dan sebagainya. Berikut ini adalah daftar dari kata kunci dalam Java (*Java Keywords*).

Tabel 2.1. Daftar *Java Keywords*

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

Keterangan table 2.1 :

* Tidak digunakan
** Ditambahkan pada versi 1.2
*** Ditambahkan pada versi 1.4
**** Ditambahkan pada versi 5.0

Catatan : true, false, dan null bukan termasuk kata kunci, tetapi termasuk kata-kata khusus, jadi kita tidak dapat menggunakannya sebagai nama *identifier*.

2.3. Java *Literals*

Penulisan besaran untuk variable adalah penting, literal di java terdiri dari :

a. Angka
b. Karakter
c. String

Angka terdiri dari bilangan bulat (*integer*), bilangan mengambang (*Floating-point*) dan *boolean*. *Boolean* dianggap angka karena nilai *true* dan *false* direpresentasikan sebagai 1 dan 0. Karakter selalu mengacu ke *Unicode*, sedangkan string berisi rangkaian karakter.

2.3.1. *Literal Integer*

Bilangan integer dapat mempunyai beberapa format berikut :

a. Decimal (basis 10), ditulis bilangan biasa tanpa notasi khusus.
b. Hexadecimal (basis 16), penulisan dimulai 0x atau 0X serupa dengan cara C/C++
c. Octal (basis 8), penulisandimulaidengan 0.

Contoh :

- Penulisan decimal : 9, 15, 28
- Penulisan hexadecimal : 0xA, 0X1F, 0XD4
- Penulisan octal : 06, 014, 027

2.3.2. *Literal Floating-Point*

Floating point literals mewakili bentuk decimal dengan bagian yang terpisah. Sebagai contoh adalah 3.1415. *Floating point literals* dapat dinyatakan dalam notasi standard atau scientific. Sebagai contoh, 583.45 dinyatakan dalam notasi standard, sementara 5.8345e2 dinyatakan dalam notasi scientific.

Default *Floating point literals* mempunyai tipe data *double* yang dinyatakan dalam 64-bit. Untuk menggunakan ketelitian yang lebih kecil (32-bit) *float*, hanya dengan menambahkan karakter “f” atau “F”.

2.3.3. *Literal Boolean*

Boolean literals hanya memiliki dua nilai, yaitu *true* dan *false*.

2.3.4. *Literal* Karakter

Character Literals diwakili oleh karakter *single Unicode*. Karakter *Unicode* adalah 16-bit *character set* yang menggantikan 8-bit ASCII *character set*. *Unicode* memungkinkan pengunaan symbol dan karakter khusus dari bahasa lain.

Untuk menggunakan *character literals*, karakter tersebut di dalam tanda *single pute* (' ') (*single quote delimiters*). Sebagai contoh huruf a, diwakili sebagai **'a'**.

Untuk menggunakan karakter khusus seperti karakter baris baru, *backslash* digunakan *backslash* (\) diikuti dengan karakter kode. Sebagai contoh, '\n' untuk karakter baris baru atau ganti baris, '\r' untuk menyatakan nilai balik (*carriage return*), '\b' untuk *backspace*.

Tabel 2.2. Karakter spesial yang didukung java

Representasi	Deskripsi
\\	*Backslah*
\	*Continuation*
\b	*Backspace*
\r	*Carriage return*
\f	*Form feed*
\t	*Horizontal tab*
\n	*Newline*
\'	*Single quote*
\"	*Double quote*
\ddd	*Octal character* (\000 sampai \377)
\udddd	*Unicode character* (\u0000 sampai \uFFFF)

Contoh penggunaan karakter spesial :

LiteralChar.java

```
import javax.swing.*;

public class LiteralChar {
  public static void main (String args[]) {
     System.out.println("literal java :\'"+'\n'+"\' slash n");
     System.out.println("literal java :\'"+'\r'+"\' slash r");
     System.out.println("literal java :\'"+'\b'+"\' slash b");
     System.out.println("literal java :\'"+'\t'+"\' slash t");
     System.out.println("literal java :\'"+'\f'+"\' slash f");
     System.out.println("literal java :\'"+'\101'+"\' slash 101");
     System.out.println("literal java :\'"+'\u0416'+"\' slash u0416");

     JOptionPane.showMessageDialog(null,"literal java :\'"+'\u0416'+ "\'
slash u0416");
   }
}
```

2.3.5. *Literal String*

String literals mewakili beberapa karakter dan dinyatakan dalam tanda *double pute* (" ") (*double quotes*). Sebagai contoh *string literal* adalah, **"**Program java**".**

2.4. Pernyataan Dalam Java dan Blok

Pernyataan adalah satu atau lebih baris kode yang diakhiri dengan *semicolon* (;), Sebagai contoh untuk pernyataan tunggal adalah :

```
System.out.println("Program Java");
```

Blok adalah satu atau lebih pernyataan yang terbentang antara kurung kurawal "{" buka dan kurung kurawal tutup "}" yaitu sekumpulan pernyataan sebagai satu unit kesatuan. Blok pernyataan dapat dikumpulkan akan tetapi tidak secara pasti mempunyai keterkaitan fungsi. Beberapa jumlah spasi kosong diijinkan terdapat di dalamnya, sebagai contoh dari suatu blok adalah :

```
public static void main( String[] args ){
    System.out.println("Program");
    System.out.println("Java");
}
```

2.5. Komentar

Komentar adalah catatan yang ditulis pada kode dengan tujuan sebagai bahan dokumentasi. Teks tersebut bukan bagian dari program dan tidak mempengaruhi jalannya program. Java mendukung tiga jenis komentar, yaitu C++ *style* komentar satu baris, C *style* beberapa baris, dan komentar java doc khusus.

2.5.1. Penulisan Komentar C++ Style

Komentar C++ *style* diawali dengan *double slash* (//), semua teks setelah // dianggap sebagai komentar, sebagai contoh :

```
// This is a C++ style or single line comments
```

2.5.2. Penulisan Komentar C Style

Komentar C *style* atau juga disebut komentar beberapa baris diawali dengan /* dan diakhiri dengan */. Semua teks yang ada diantara kedua tanda tersebut dianggap sebagai komentar.

Tidak seperti komentar C++ style, komentar ini dapat menjangkau beberapa baris, sebagai contoh :

```
/* this is an example of a
C style or multiline comments */
```

2.5.3. Komentar Khusus Java Doc

Komentar java doc khusus digunakan untuk men-*generate* dokumentasi HTML untuk program java anda. Anda dapat menciptakan komentar java doc dengan memulai baris dengan /** dan mengakhirinya dengan */. Seperti komentar C *style*, penulisan komentar ini dapat menjangkau beberapa baris. Komentar ini juga dapat terdiri atas tag-tag untuk menambahkan lebih banyak informasi pada komentar Anda. Sebagai contoh :

```
/**
This is an example of special java doc comments used for \n
generating an html documentation. It uses tags like:
@author Florence Balagtas
@version 1.2
*/
```

2.6. Tipe Data Dasar (*Primitive*)

Bahasa pemrograman Java mendefinisikan delapan tipe data primitif. Tipe-tipe tersebut adalah *boolean* (untuk bentuk logika), *char* (untuk bentuk tekstual), *byte*, *short*, *int*, *long* (untuk *integral*), *double* and *float* (untuk *floating point*).

2.6.1. Logika (*boolean*)

Tipe data Boolean memiliki dua nilai, yaitu true dan false. Sebagai contoh adalah :

```
boolean result = true;
```

Contoh yang ditunjukkan diatas, mendeklarasikan variabel yang dinamai **result** sebagai tipe data **Boolean** dan memberinya nilai **true**.

2.6.2. Tekstual (*char*)

Tipe data *character* (*char*), diwakili oleh karakter *single unicode* yang terdiri dari 16 bit *unsigned integer*. Tipe data ini harus memiliki ciri berada dalam tanda *single quotes*(' '). Sebagai contoh :

```
'a' //huruf a
'\t' //tab
```

Untuk menampilkan karakter khusus seperti ' (*single quotes*) atau " (*double quotes*), menggunakan karakter *escape* "\". Sebagai contoh :

```
'\'' //untuk single quotes
'\"' //untuk double quotes
```

String bukan merupakan tipe data primitive, tetapi merupakan suatu *Class*. Kita akan memperkenalkan mengenai *string* pada bagian ini. *String* mewakili tipe data yang terdiri atas beberapa karakter. *String* memiliki literal yang terdapat di antara tanda *double quotes* (" "), sebagai contoh :

```
String Pesan = "Silahkan belajar Java !"
```

2.6.3. *Integral (byte, short, int* dan *long)*

Tipe data integral dalam Java menggunakan tiga bentuk, yaitu desimal, octal dan heksadesimal. Contohnya :

```
2 //nilai desimal 2
077 //angka 0 pada awal pernyataan mengindikasikan nilai oktal
0xBACC //karakter 0x mengindikasikan nilai heksadesimal
```

Tipe-tipe integral memiliki default tipe data yaitu int. Anda dapat merubahnya ke bentuk long dengan menambahkan huruf l atau L. Tipe data integral memiliki range sebagai berikut :

Tabel 2.3.Tipe-tipe integral dan jangkauannya

Tipe	Ukuran	Jangkauan
byte	8 bit	-128 sampai 127
short	16 bit	-32,768 sampai 32,767
int	32 bit	-2,147,483,648 sampai 2,147,483,647
long	64 bit	-9,223,372,036,854,775,808 sampai +9,223,372,036,854,775,807

Catatan : Dalam mendefinisikan suatu nilai long tidak dianjurkan menggunakan *lowercase* L, karena sulit dibedakan dengan angka 1.

2.6.4. *Floating Point (float dan double)*

Tipe data *floating point* memiliki nilai standar double. Floating-point literal termasuk salah satunya desimal point atau salah satu dari pilihan berikut ini :

```
E or e //(add exponential value)
F or f //(float)
D or d //(double)
```

Contohnya adalah :

```
3.14 //nilai floating-point sederhana (double)
6.02E23 //nilai floating-point yang besar
2.718F //nilai float size sederhana
123.4E+306D //nilai double yang besar dengan nilai redundant D
```

Pada contoh yang ditunjukkan diatas, 23 setelah E pada contoh kedua bernilai positif. Contoh tersebut sama dengan 6.02E+23. Tipe data *Floating-point* memiliki jangkauan sebagai berikut :

Table 2.4. Tipe *Floating point* dan jangkauannya

Tipe	Ukuran	Range
float	32 bit	+/-1.4E-45 sampai +/-3.4028235E+38, +/-*infinity (continue)*, +/-0, NAN (*Not a Number*)
double	64 bit	+/-4.9E-324 sampai +/-1.7976931348623157E+308, +/-*infinity (continue)*, +/-0, NaN(*Not a Number*)

Table 2.5. Struktur tipe *Floating point*

Tipe	Ukuran	Struktur		
		Sign	Exponent	Mantissa
float	32 bit	1 bit	8 bit	23 bit
double	64 bit	1 bit	11 bit	52 bit

2.7. Variabel

Variabel adalah unit dasar penyimpanan di program java. Variabel memiliki tipe data dan nama. Tipe data menandakan tipe nilai yang dapat dibentuk oleh variabel itu sendiri. Nama variabel harus mengikuti aturan untuk *identifier*.

2.7.1. Deklarasi dan Inisialisasi Variabel

Untuk deklarasi variabel adalah sebagai berikut :

```
<data tipe> <name> [=initial value];
```

Catatan : Nilai yang berada diantara <> adalah nilai yang disyaratkan, sementara nilai dalam tanda [] bersifat optional.

Berikut ini adalah contoh program yang mendeklarasikan dan menginisialisasi beberapa variable :

VariableSamples.java

```
public class VariableSamples {
   public static void main( String[] args ){
      // deklarasi tipe data dengan nama variable result
      // dengan tipe data boolean
      boolean result;

      //deklarasi tipe data dengan nama variable option
      //dengan tipe data char
      char option;
      option = 'C'; //menginisialisasi option dengan nilai 'C'

      //deklarasi tipe data dengan nama variable grade,
      //dengan tipe double dan langsung di inisialisasi
      //dengan nilai 0.0
      double grade = 0.0;
   }
}
```

Petunjuk Penulisan Program:

a. Sebaiknya menginisialisasi variabel yang dibuat saat mendeklarasikannya.
b. Gunakan nama yang dapat menggambarkan variabel yang dibuat, jika ingin membuat variabel untuk menyimpan nilai siswa, sebaiknya diberi nama nilaiSiswa, jangan hanya dengan beberapa huruf random.
c. Deklarasikan satu variabel tiap baris kode. Sebagai contoh, deklarasi variabel adalah sebagai berikut :

```
double exam=0;
double quiz=10;
double grade = 0;
```

Bentuk yang lebih disukai ketika melakukan deklarasi adalah :

```
double exam=0, quiz=10, grade=0;
```

2.7.2. Menampilkan Data Variabel

Untuk mengeluarkan nilai dari variabel yang diinginkan, kita dapat menggunakan perintah berikut ini :

```
System.out.println()
System.out.print()
```

Berikut ini adalah contoh program :

OutputVariable.java

```
public class OutputVariable {
  public static void main( String[] args ){
    int nilai = 10;
    char x;
    x = 'A';

    System.out.println( nilai );
    System.out.println("nilai dari x = " + x );
  }
}
```

Program tersebut akan mengeluarkan teks pada layar sebagai berikut :

```
10
nilai dari x = A
```

2.7.3. *System.out.println()* dan *System.out.print()*

Apa yang membedakan diantara perintah *System.out.println*() dengan *System.out.print*()? *System.out.println*() akan menambahkan baris baru pada akhir data yang dikeluarkan, sementara *System.out.print*() tidak menambah baris baru. Perhatikan pernyataan tersebut :

```
System.out.print("Hello ");
System.out.print("world!");
```

Pernyataan tersebut akan menghasilkan output berikut ini pada layar :

```
Hello world!
```

Sekarang perthatikan pernyataan berikut :

```
System.out.println("Hello ");
System.out.println("world!");
```

Pernyataan ini akan menghasilkan output sebagai berikut pada layar,

```
Hello
world!
```

2.7.4. Variabel *Reference* dan Variabel Primitif

Sekarang kita akan membedakan dua tipe variabel yang dimiliki oleh program Java. Ada variabel *reference* dan variabel primitif.

Variabel primitif adalah variabel dengan tipe data primitif. Mereka menyimpan data dalam lokasi memori yang sebenarnya dimana variabel tersebut berada.

Variabel *reference* adalah variabel yang menyimpan alamat lokasi dalam memori. Yang menunjuk ke lokasi memori dimana data sebenarnya berada. Ketika kita mendeklarasikan variabel pada *class* tertentu, kita sebenarnya mendeklarasikan *reference* variable dalam bentuk objek dalam *class*-nya tersebut.

Sebagai contoh, Apabila kita mempunyai dua variabel dengan tipe data int dan String.

```
int num = 10;
String name = "Hello"
```

Dimisalkan ilustrasi yang ditunjukkan dibawah ini adalah memori yang ada pada komputer Anda, dimana Anda memiliki alamat dari setiap sel memorinya, nama variabel dan datanya terbentuk sebagai berikut.

Memory Address	***Variable Name***	**Data**
1001	num	10
::		::
1563	name	Address(2000)
::		::
::		::
2000		“Hello”
::		::

Seperti yang dapat kita lihat, untuk variable primitif num, datanya berada dalam lokasi dimana variabel berada. Untuk *reference variable name*, variabel hanya menunjuk alamat dimana data tersebut benar-benar ada.

2.8. Konstanta

Untuk menunjukkan konstanta, di java menggunakan kata kunci (*keyword*) final. Konstanta adalah suatu variable yang didefinisikan dan diberi nilai sekali saja dan tidak dapat diubah.

Konstanta.java

```
public class Konstanta {
   public static void main (String args[]) {
      final double PHI = 3.14;
      double radius = 10;

      System.out.println("Luas lingkaran yang berjari-jari "+radius+"
adalah "+(PHI*radius*radius));
    }
 }
```

Keluaran dari program diatas adalah :

```
Luas lingkaran yang berjari-jari 10.0 adalah 314.0
```

2.9. Operator

Dalam Java, ada beberapa tipe operator. Ada operator aritmatika, operator relasi, operator logika, dan operator kondisi. Operator ini mengikuti bermacam-macam prioritas yang pasti sehingga compilernya akan tahu yang mana operator untuk dijalankan lebih dulu ketika beberapa operator dipakai bersama-sama dalam satu pernyataan.

2.9.1. Operator Aritmatika

Berikut ini adalah dasar operator aritmatika yang dapat digunakan untuk membuat suatu program Java,

Tabel 2.6. Operator Aritmatika dan fungsi-fungsinya

Operator	Penggunaan	Keterangan
+	op1 + op2	Menambahkan op1 dengan op2
*	op1 * op2	Mengalikan op1 dengan op2
/	op1 / op2	Membagi op1 dengan op2
%	op1 % op2	Menghitung sisa dari pembagian op1 dengan op2
-	op1 - op2	Mengurangkan op2 dari op1

Berikut ini adalah contoh program untuk menunjukkan penggunaan operator-operator diatas :

AritmatikaDemo.java

```
public class AritmatikaDemo {
  public static void main(String[] args) {
    //deklarasi dan inisialisasi variabel
    int i = 37;
    int j = 42;
    double x = 27.475;
    double y = 7.22;

    System.out.println("Menampilkan nilai awal Variable :");
    System.out.println(" i = " + i);
    System.out.println(" j = " + j);
    System.out.println(" x = " + x);
    System.out.println(" y = " + y);
    //penjumlahan angka
    System.out.println("Penambahan :");
    System.out.println(" i + j = " + (i + j));
    System.out.println(" x + y = " + (x + y));
    //pengurangan angka
    System.out.println("Pengurangan :");
    System.out.println(" i - j = " + (i - j));
    System.out.println(" x - y = " + (x - y));
    //perkalian angka
    System.out.println("Perkalian :");
    System.out.println(" i * j = " + (i * j));
    System.out.println(" x * y = " + (x * y));
    //pembagian angka
    System.out.println("Pembagian :");
    System.out.println(" i / j = " + (i / j));
    System.out.println(" x / y = " + (x / y));
    //menghitung hasil modulus dari pembagian
    System.out.println("Sisa pembagian :");
```

```
      System.out.println(" i % j = " + (i % j));
      System.out.println(" x % y = " + (x % y));
      //tipe penggabungan
      System.out.println("Penggabungan tipe :");
      System.out.println(" j + y = " + (j + y));
      System.out.println(" i * x = " + (i * x));
   }
}
```

Berikut ini adalah output program :

```
Menampilkan nilai awal Variable :
 i = 37
 j = 42
 x = 27.475
 y = 7.22
Penambahan :
 i + j = 79
 x + y = 34.695
Pengurangan :
 i - j = -5
 x - y = 20.255000000000003
Perkalian :
 i * j = 1554
 x * y = 198.36950000000002
Pembagian :
 i / j = 0
 x / y = 3.805401662049862
Sisa pembagian :
 i % j = 37
 x % y = 5.815000000000002
Penggabungan tipe :
 j + y = 49.22
 i * x = 1016.575
```

Catatan :

Ketika integer dan floating-point number digunakan sebagai operand untuk operasi aritmatika tunggal, hasilnya berupa floating point. Integer adalah converter secara implisit ke bentuk angka floating-point sebelum operasi berperan mengambil tempat.

2.9.2. Operator *Increment* dan *Decrement*

Dari sisi operator dasar aritmatika, Java juga terdiri atas operator *unary increment* (++) dan operator *unary decrement* (--). operator increment dan decrement menambah dan mengurangi nilai yang tersimpan dalam bentuk variabel angka terhadap nilai 1. Sebagai contoh, pernyataan :

```
count = count + 1; //increment nilai count dengan nilai 1
```

pernyataan tersebut sama dengan :

```
count++;
```

Tabel 2.7. operator *Increment* dan *Decrement*

Operator	Penggunaan	Keterangan
++	op++	Menambahkan nilai 1 pada op; mengevaluasi nilai op sebelum diincrementasi/ditambahkan
++	++op	Menambahkan nilai 1 pada op; mengevaluasi nilai op setelah diincrementasi/ditambahkan
--	op--	Mengurangkan nilai 1 pada op; mengevaluasi nilai op sebelum didecrementasi/dikurangkan
--	--op	Mengurangkan nilai 1 pada op; mengevaluasi nilai op setelah didecrementasi/dikurangkan
+=	op += x	Menambahkan nilai op sebanyak x; atau sama dengan kita menulis : op = op + x
-=	op -= x	Mengurangi nilai op sebanyak x; atau sama dengan kita menulis : op = op - x

Operator increment dan decrement dapat ditempatkan sebelum atau sesudah operand. Ketika digunakan sebelum operand, akan menyebabkan variabel diincrement atau didecrement dengan nilai 1, dan kemudian nilai baru digunakan dalam pernyataan dimana dia ditambahkan. Sebagai contoh :

```
int i = 10, int j = 3;
int k = 0;
k = ++j + i; //akan menghasilkan k = 4+10 = 14
```

Ketika operator increment dan decrement ditempatkan setelah operand, nilai variabel yang lama akan digunakan/dioperasikan lebih dulu terhadap pernyataan dimana dia ditambahkan. Sebagai contoh :

```
int i = 10, int j = 3;
int k = 0;
k = j++ + i; //akan menghasilkan k = 3+10 = 13
```

2.9.3. Operator Relasi

Operator Relasi adalah membandingkan dua nilai dan menentukan keterhubungan diantara nilai-nilai tersebut. Hasil keluarannya berupa nilai boolean yaitu true atau false.

Table 2.8. Operator Relasi

Operator	Penggunaan	Keterangan
>	op1 > op2	op1 lebih besar dari op2
>=	op1 >= op2	op1 lebih besar dari atau sama dengan op2
<	op1 < op2	op1 kurang dari op2
<=	op1 <= op2	op1 kurang dari atau sama dengan op2
==	op1 == op2	op1 sama dengan op2
!=	op1 != op2	op1 tidak sama dengan op2

Berikut ini adalah contoh program yang menggunakan operator Relasi :

RelasiDemo.java

```
public class RelasiDemo {
   public static void main(String[] args) {
     //beberapa nilai
     int i = 37;
     int j = 42;
     int k = 42;

     System.out.println("Nilai variabel...");
     System.out.println(" i = " + i);
     System.out.println(" j = " + j);
     System.out.println(" k = " + k);
     //lebih besar dari
     System.out.println("Lebih besar dari...");
     System.out.println(" i > j = " + (i > j)); //false
     System.out.println(" j > i = " + (j > i)); //true
     System.out.println(" k > j = " + (k > j)); //false
     //lebih besar atau sama dengan
     System.out.println("Lebih besar dari atau sama dengan...");
     System.out.println(" i >= j = " + (i >= j)); //false
     System.out.println(" j >= i = " + (j >= i)); //true
     System.out.println(" k >= j = " + (k >= j)); //true
     //lebih kecil dari
     System.out.println("Lebih kecil dari...");
     System.out.println(" i < j = " + (i < j)); //true
     System.out.println(" j < i = " + (j < i)); //false
     System.out.println(" k < j = " + (k < j)); //false
     //lebih kecil atau sama dengan
     System.out.println("Lebih kecil dari atau sama dengan...");
     System.out.println(" i <= j = " + (i <= j)); //true
     System.out.println(" j <= i = " + (j <= i)); //false
     System.out.println(" k <= j = " + (k <= j)); //true
     //sama dengan
     System.out.println("Sama dengan...");
     System.out.println(" i == j = " + (i == j)); //false
     System.out.println(" k == j = " + (k == j)); //true
     //tidak sama dengan
     System.out.println("Tidak sama dengan...");
     System.out.println(" i != j = " + (i != j)); //true
     System.out.println(" k != j = " + (k != j)); //false
   }
}
```

Di bawah ini adalah hasil keluaran dari program di atas :

```
Nilai variabel...
 i = 37
 j = 42
 k = 42
Lebih besar dari...
 i > j = false
 j > i = true
 k > j = false
Lebih besar dari atau sama dengan...
 i >= j = false
 j >= i = true
 k >= j = true
```

```
Lebih kecil dari...
 i < j = true
 j < i = false
 k < j = false
Lebih kecil dari atau sama dengan...
 i <= j = true
 j <= i = false
 k <= j = true
Sama dengan...
 i == j = false
 k == j = true
Tidak sama dengan...
 i != j = true
 k != j = false
```

2.9.4. Operator Logika

Operator logika memiliki satu atau lebih operand boolean yang menghasilkan nilai boolean. Terdapat enam operator logika yaitu: && (logika AND), & (boolean logika AND), || (logika OR), | (boolean logika inclusive OR), ^ (boolean logika exclusive OR), dan ! (logika NOT).

Pernyataan dasar untuk operasi logika adalah :

```
x1 op x2
```

Dimana x1, x2 dapat menjadi pernyataan boolean. Variabel atau konstanta, dan op adalah salah satu dari operator &&, &, ||, | atau ^. Tabel kebenaran yang akan ditunjukkan selanjutnya, merupakan kesimpulan dari hasil dari setiap operasi untuk semua kombinasi yang mungkin dari x1 dan x2.

2.9.4.1. Operator && (logika AND), & (boolean logika AND)

Berikut ini adalah tabel kebenaran untuk && dan & :

Tabel 2.9. Tabel kebenaran untuk && dan &

x1	x2	Hasil
True	True	True
True	False	False
False	True	False
False	False	False

Perbedaan dasar antara operator && dan & adalah bahwa && mensupports short-circuit evaluations (atau evaluasi perbagian), sementara operator & tidak. Apa arti dari pernyataan tersebut? Diberikan suatu pernyataan:

```
exp1 && exp2
```

&& akan mengevaluasi pernyataan exp1, dan segera mengembalikan nilai false dan menyatakan bahwa exp1 bernilai false. Jika exp1 bernilai false, operator tidak akan pernah mengevaluasi exp2 karena hasil operasi operator akan menjadi false tanpa memperhatikan nilai dari exp2. Sebaliknya, operator & selalu mengevaluasi kedua nilai dari exp1 dan exp2 sebelum mengembalikan suatu nilai jawaban.

Berikut ini adalah suatu contoh *source code* yang menggunakan logika dan boolean AND :

TestAND.java

```
public class TestAND {
   public static void main( String[] args ){
      int i = 0;
      int j = 10;
      boolean test= false;

      //demonstrasi &&
      test = (i > 10) && (j++ > 9);
      System.out.println(i);
      System.out.println(j);
      System.out.println(test);
      //demonstrasi &
      test = (i > 10) & (j++ > 9);
      System.out.println(i);
      System.out.println(j);
      System.out.println(test);
   }
}
```

Keluaran dari program diatas adalah :

```
0
10
false
0
11
False
```

Catatan :

bahwa j++ pada baris yang mengandung operator && tidak dievaluasi jika pernyataan pertama (i>10) telah bernilai sama dengan false.

2.9.4.2. Operator || (logika OR), | (boolean logika OR)

Berikut ini adalah tabel kebenaran untuk || dan | :

Tabel 2.10. Tabel kebenaran untuk || dan |

x1	x2	Hasil
True	True	True
True	False	True
False	True	True
False	False	False

Perbedaan dasar antara operator || dan | adalah bahwa || mendukung short-circuit evaluations (atau proses evaluasi sebagian), sementara | tidak. Apa maksud dari pernyataan tersebut? Berikut ini adalah suatu pernyataan :

```
exp1 || exp2
```

|| akan mengevaluasi pernyataan exp1, dan segera mengembalikan nilai true dan menyatakan bahwa exp1 bernilai true. Jika exp1 bernilai true, operator tidak akan pernah mengevaluasi exp2 karena hasil dari operasi operator akan bernilai true tanpa memperhatikan nilai dari exp2. Sebaliknya, operator | selalu mengevaluasi kedua nilai dari exp1 and exp2 sebelum mengembalikan suatu jawaban suatu nilai.

Berikut ini sebuah contoh source code yang menggunakan operator logika dan boolean OR :

TestOR.java

```
public class TestOR {
  public static void main( String[] args ){
    int i = 0;
    int j = 10;
    boolean test= false;

    //demonstrasi ||
    test = (i < 10) || (j++ > 9);
    System.out.println(i);
    System.out.println(j);
    System.out.println(test);
    //demonstrasi |
    test = (i < 10) | (j++ > 9);
    System.out.println(i);
    System.out.println(j);
    System.out.println(test);
  }
}
```

Hasil keluaran dari program diatas adalah :

```
0
10
true
0
11
True
```

Catatan : bahwa j++ pada baris yang terdiri atas operator || tidak dievaluasi jika pernyataan pertama (i<10) telah bernilai sama dengan true.

2.9.4.3. Operator ^ (boolean logika ExclusiveOR)

Berikut ini adalah tabel kebenaran untuk ^ :

Tabel 2.11. Tabel kebenaran untuk ^

x1	x2	Hasil
True	True	False
True	False	True
False	True	True
False	False	False

Hasil operasi operator exclusive OR adalah TRUE, jika dan hanya jika satu operand bernilai TRUE dan yang lain bernilai False. Catatan jika kedua operand harus selalu dievaluasi untuk menjumlahkan hasil dari suatu exclusive OR.

Berikut ini adalah contoh source code yang menggunakan operator logika exclusive OR :

TestXOR.java

```
public class TestXOR {
   public static void main( String[] args ){
      boolean val1 = true;
      boolean val2 = true;
      System.out.println(val1 ^ val2);
      val1 = false;
      val2 = true;
      System.out.println(val1 ^ val2);
      val1 = false;
      val2 = false;
      System.out.println(val1 ^ val2);
      val1 = true;
      val2 = false;
      System.out.println(val1 ^ val2);
   }
}
```

Hasil keluaran program tersebut adalah :

```
false
true
false
true
```

2.9.4.4. Operator ! (logika NOT)

Logika NOT digunakan dalam satu argumen, dimana argumen tersebut dapat menjadi suatu pernyataan, variabel atau konstanta. Berikut ini adalah tabel kebenaran untuk operator NOT ! :

Tabel 2.12. Tabel kebenaran NOT !

x1	Hasil
True	False
False	True

Berikut ini adalah contoh *source code* menggunakan operator logika NOT :

TestNOT.java

```
public class TestNOT {
   public static void main( String[] args ){
      boolean val1 = true;
      boolean val2 = false;
      System.out.println(!val1);
      System.out.println(!val2);
   }
}
```

Hasil keluaran program adalah sebagai berikut :

```
false
true
```

2.9.5. Operator Kondisi (?:)

Operator kondisi **?:** adalah operator ternary. Berarti bahwa operator ini membawa tiga argumen yang membentuk suatu ekspresi bersyarat. Struktur pernyataan yang menggunakan operator kondisi adalah :

```
exp1?exp2:exp3
```

Dimana nilai exp1 adalah suatu pernyataan boolean yang memiliki hasil yang salah satunya harus berupa nilai true atau false. Jika exp1 bernilai true, exp2 merupakan hasil operasi. Jika bernilai false, kemudian exp3 merupakan hasil operasinya. Sebagai contoh, diberikan code sebagai berikut :

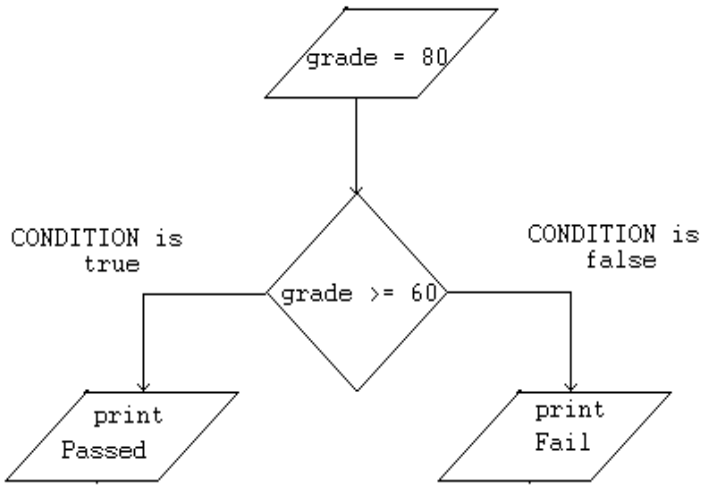
ConditionalOperator.java

```
public class ConditionalOperator {
   public static void main( String[] args ){
      String status = "";
      int grade = 80;
      //mendapatkan status pelajar
      status = (grade >= 60)?"Passed":"Fail";
      //print status
      System.out.println( status );
   }
}
```

Hasil keluaran dari program ini akan menjadi :

```
Passed
```

Berikut ini adalah flowchart yang menggambarkan bagaimana operator ?: bekerja :



Gambar 2.1. Flowchart operator ?:

Berikut ini adalah program lain yang menggunakan operator ?:

ConditionalOperator2.java

```
class ConditionalOperator2 {
  public static void main( String[] args ){
    int score = 0;
    char answer = 'a';
    score = (answer == 'a') ? 10 : 0;
    System.out.println("Score = " + score );
  }
}
```

Hasil keluaran program adalah :

```
Score = 10
```

2.9.6. Operator *Precedence*

Operator *precedence* didefinisikan sebagai perintah yang dilakukan *compiler* ketika melakukan evaluasi terhadap operator, untuk mengajukan perintah dengan hasil yang tidak ambigu/hasil yang jelas. Diberikan pernyataan yang membingungkan :

```
6%2*5+4/2+88-10
```

Kita dapat menuliskan kembali pernyataan diatas dan menambahkan beberapa tanda kurung terhadap operator *precedence* :

```
((6%2)*5)+(4/2)+88-10;
```

Catatan :

Untuk menghindari kebingungan dalam evaluasi operasi matematika, buatlah pernyataan sesederhana mungkin dan gunakan bantuan tanda kurung.

.	[ ]	( )	
++	--	!	~
*	/	%	
+	-		
<<	>>	>>>	<<<
<	>	<=	>=
==	!=		
&	|		
^			
&&			
||			
?:			
=			

Gambar 2.2. Operator Precedence

2.9.7. Operator *String*

String juga dapat dimanipulasi dengan operator, hanya terdapat satu operator string, yaitu operator penyambungan (*concatenation*) dengan simbol "+". Java tidak memiliki tipe data *String* secara *built in*. Java menyelesaikan tipe data *String* dengan kelas *String* di pustaka java. Selain operasi penyambungan, operasi-operasi lain adalah pemanggilan metode-metode di obyek bertipe *String*.

Metode yang digunakan untuk mengambil bagian string adalah *substring*. Sedangkan untuk menguji kesamaan antara dua string dapat digunakan metode equals (*case-sensitive*) atau equalsIgnoreCase (tidak *case-sensitive*).

Contoh program untuk operasi string :

OperatorString.java

```
public class OperatorString {
   public static void main(String[] args) {
      String S1 = "Saya";
      String S2 = "Belajar Java";
      String S3 = "belajar java";

      System.out.println(S1+" "+S2);
      System.out.println(S2.substring(0,7));
      System.out.println(S2.equals(S3));
      System.out.println(S2.equalsIgnoreCase(S3));
   }
}
```

Keluaran dari program diatas adalah :

```
Saya Belajar Java
Belajar
false
true
```

2.10. Konversi Tipe

Ketika membuat program, kita biasa memberikan nilai satu tipe dengan tipe lain. Jika dua tipe tersebut kompatibel, maka java akan melakukan konversi secara otomatis. Misalnya, jika kita memberikan nilai bertipe *int* ke variabel bertipe *long*.

Tidak semua tipe data adalah kompatibel, misalnya kita mengisi variabel bertipe *byte* dengan data bertipe *double*. Tetapi kita masih dapat melakukannya secara manual, tetapi kita harus memperhatikan konsekuensi yang mungkin terjadi, seperti hilangnya sebagian data.

2.10.1. Konversi Tipe Otomatis

Ketika satu variabel diisi dengan data bertipe lain, maka java akan mengkonversi tipe data tersebut secara otomatis bila memenuhi dua syarat berikut ini :

a. Kedua tipe adalah kompatible
b. Tipe data tujuan memiliki ukuran lebih besar dibanding tipe sumber

Ketika kedua syarat diatas terpenuhi, maka konversi pelebaran ukuran akan dilakukan secara otomatis. Tipe data int lebih besar dari pada tipe data *byte*, ketika kita

melakukan konversi dari data bertipe *byte* ke data bertipe *int*, kita tidak perlu melakukan konversi secara ekplisit.

Konversi pelebaran akan dilakukan pada bilangan bulat dan bilangan *floating-point* yang kompatibel antara satu dengan yang lain. Tipe data numerik tidak kompatibel dengan *char* atau *boolean*. Tipe *char* dan *boolean* juga tidak saling kompatibel.

Java juga secara otomatis melakukan konversi tipe ketika menyimpan literal bilangan bulat ke variabel bertipe *byte*, *short*, atau *long*.

2.10.2. *Casting*

Casting atau typecasting adalah proses konversi data dari tipe data tertentu ke tipe data yang lain. Adakalanya kita perlu memaksa konversi dari tipe data tertentu ke tipe data lain yang tidak kompatibel atau memiliki ukuran lebih kecil. Misalnya ketika fungsi mengirim tipe berbeda dari tipe yang diperlukan untuk operasi, maka tindakan *casting* diperlukan.

Tindakan *casting* dilakukan dengan menempatkan tipe data yang diharapkan di dalam tanda kurung disebelah kiri nilai yang dikonversi.

Sintaks casting adalah :

```
(TipeTarget) Nilai
```

TipeTarget adalah tipe data yang diinginkan, sedangkan Nilai adalah nilai yang akan dikonversi. Contoh penggunaannya dapat dilihat pada program dibawah ini :

ContohCasting.java

```
public class ContohCasting {
  public static void main(String args[]){
    int i1=17, i2=4;

    double d= (double) i1/i2;

    System.out.println(d);
    System.out.println((int) d);
  }
}
```

Keluaran dari program diatas adalah :

```
4.25
4
```

Jika proses *casting* pada pengisian variabel d, yaitu "(*double*)" dihilangkan maka keluarannya menjadi seperti :

```
4.0
4
```

Pengetahuan ukuran penyimpanan penting dalam menentukan hasil akhir *casting*. Tidak semua *casting* dapat menyelamatkan semua data sebelumnya secara utuh. Pada proses *casting long* menjadi *int*, berarti tipe *long* (64 bit) menjadi tipe *int* (32 bit). Jika ini

dilakukan maka kompilator akan memotong 32 bit tipe data *long* menjadi 32 bit tipe data *int*. Jika nilai 32 bit bagian atas berisi informasi penting, maka informasi itu akan hilang.

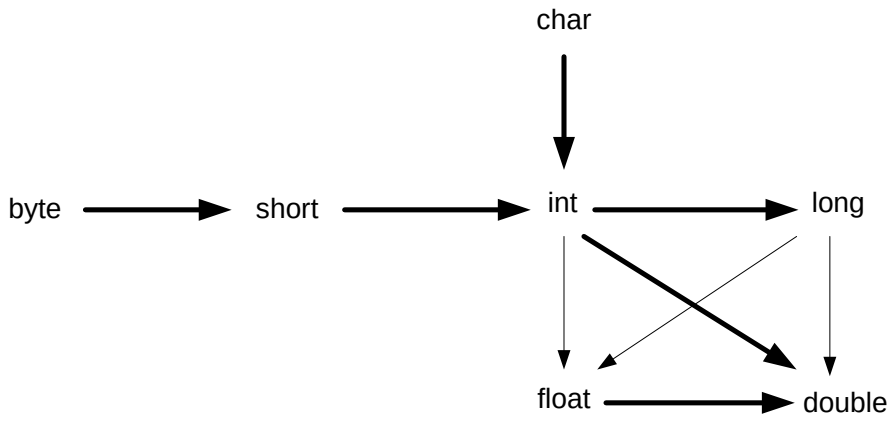
Informasi juga dapat hilang ketika kita melakukan *casting* antara tipe data yang berbeda, walupun memiliki ukuran penyimpanan yang sama. Misalnya kita melakukan *casting* bilangan *double* menjadi *long*, walaupun ukurannya sama 64 bit, hal ini akan menghilangkan nilai pecahannya.

Konversi yang mengakibatkan pemotongan disebut *truncation*. Pada tabel 2.13 berikut ini adalah daftar proses *casting* yang tidak mengakibatkan pemotongan dan hasilnya dijamin tidak ada yang hilang.

Tabel 2.13. *Casting* yang tidak mengalami *truncation*

Tipe Asal	Tipe Tujuan
byte	short, char, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

Alur konversi tipe-tipe data primitif dapat dilihat pada gambar 2.3 dibawah ini :



Gambar 2.3. Alur konversi tipe-tipe data primitif

Enam anak panah tebal menunjukkan konversi yang tidak menghilangkan informasi. Sedangkan tiga anak panah tipis menunjukkan konversi yang dapat menimbulkan kehilangan presisi. Contoh bilangan int 123456789 mempunyai banyak digit yang dapat direpresentasikan float, hasilnya mempunyai magnitudo yang benar namun menghilangkan presisi.

CastingInt2Float.java

```
public class CastingInt2Float {
  public static void main(String args[]){
    int N_int = 123456789;
    float N_float = N_int;
```

```
      System.out.println(N_int);
      System.out.println(N_float);
   }
}
```

Keluaran program diatas adalah :

```
123456789
1.23456792E8
```

Terlihat bahwa digit ke-8 dan ke-9 nilainya berubah dari 89 menjadi 92. Hal ini penting untuk diperhatikan dalam membuat program agar tidak terjadi kesalahan dalam operasi.

2.11. Membaca masukan dari *Arguments* (parameter)

Kita dapat memberikan masukan ke program yang kita buat melalui argument (parameter) pada saat menjalankan program. Contoh programnya adalah sebagai berikut :

PenjumlahanArgs.java

```
public  class  PenjumlahanArgs{
   public static void main(String[] args){
      int bilangan1 = Integer.parseInt(args[0]);
      int bilangan2 = Integer.parseInt(args[1]);

      System.out.print("Hasil penjumlahan : " + (bilangan1 + bilangan2));
   }
}
```

Kemudian compile dengan perintah javac, jika tidak ada kesalahan, jalankan dengan menggunakan perintah :

```
java PenjumlahanArgs 4 5
```

2.12. Membaca masukan dari *keyboard*

Kita telah mempelajari konsep dasar pemrograman pada Java dan menulis beberapa program sederhana. Sekarang kita akan mencoba membuat program yang lebih interaktif dengan menggunakan *input* dari *keyboard*. Pada bab ini, kita akan mempelajari dua cara memberikan *input*, yaitu menggunakan *class BufferedReader* dan melalui GUI (*Graphical User Interface*) dengan menggunakan *class JOptionPane*.

2.12.1. Menggunakan *Scanner*

Untuk membaca masukan dari keyboard, kita dapat menggunakan *class scanner* yang ada pada paket *java.util.Scanner*. Metode-metode yang dapat digunakan untuk membaca masukan dari keyboard antara lain :

a. `nextInt()` : untuk menerima tipe data integer
b. `nextShort()` : untuk menerima tipe data short
c. `nextLong()` : untuk menerima tipe data long
d. `nextDouble()` : untuk menerima tipe data double

e. `nextFloat()` : untuk menerima tipe data float
f. `nextLine()` : untuk menerima tipe data string
g. `nextBoolean()` : untuk menerima tipa data boolean

Contoh penggunaan metode scanner dalam program adalah sebagai berikut :

ScannerString.java

```
import java.util.Scanner;

public class ScannerString {
   public static void main( String[] args ){
      Scanner masukan = new Scanner(System.in);

      System.out.print("Ketik Nama Anda : ");
      String nama = masukan.nextLine();

      System.out.println("Halo " + nama + ", kamu sekarang sudah bisa
java");
    }
}
```

ScannerInt.java

```
import java.util.Scanner;

public  class  ScannerInt{
   public static void main(String[] args){
      Scanner input = new Scanner(System.in);
      System.out.print("Ketik bilangan pertama : ");
      int bilangan1 = input.nextInt();

      System.out.print("Ketik bilangan kedua : ");
      int bilangan2 = input.nextInt();

      System.out.print("Hasil perkalian: " + (bilangan1 * bilangan2));
   }
}
```

2.12.2. Menggunakan *BufferedReader*

Pada bagian ini, kita akan menggunakan *class BufferedReader* yang berada di paket *java.io* untuk mendapatkan *input* dari *keyboard*.

Berikut ini adalah program untuk membaca input dari keyboard menggunakan class BufferedReader :

GetInputKeyboardBufferedReader.java

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

public class GetInputKeyboardBufferedReader{
   public static void main( String[] args ){
     BufferedReader dataIn = new BufferedReader(new
InputStreamReader(System.in));
     String nama = "";
     System.out.print("Ketik nama anda : ");
```

```
      try{
        nama = dataIn.readLine();
      }
      catch( IOException e ){
        System.out.println("Ada kesalahan !");
      }

      System.out.println();
      System.out.println("Hello " + nama +"\nLanjutkan belajarnya pasti
menjadi programmer Java !");
    }
  }
```

Penjelasan dari program diatas adalah sebagai berikut:

Statement :

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

menjelaskan bahwa kita akan menggunakan class *BufferedReader*, *InputStreamReader* dan *IOException* yang berada di paket *java.io*. Java *Application Programming Interface* (API) berisi ratusan *class* yang sudah didefinisikan sebelumnya yang dapat digunakan untuk program kita. *Class-class* tersebut dikumpulkan di dalam paket-paket**.**

Paket berisi class yang mempunyai fungsi yang saling berhubungan. Seperti pada contoh di atas, paket *java.io* mengandung *class-class* yang memungkinkan program untuk melakukan *input* dan *output* data. Pernyataan di atas juga dapat ditulis sebagai berikut :

```
import java.io.*
```

yang akan mengeluarkan semua *class* yang berada dalam paket, dan selanjutnya kita dapat menggunakan *class-class* tersebut dalam program kita.

Dua statement selanjutnya :

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

kita sudah mempelajari pada pelajaran sebelumnya. Pernyataan ini mendeklarasikan *class* bernama *GetInputKeyboardBufferedReader* dan kita mendeklarasikan metode *main* yang akan diproses pertama kali ketika program dijalankan.

Dalam statement :

```
BufferedReader dataIn = new BufferedReader(new
InputStreamReader(System.in));
```

kita mendeklarasikan sebuah variabel bernama *dataIn* dengan tipe *class BufferedReader*. *BufferedReader* adalah *class* yg disediakan oleh java untuk melakukan proses *input*/*output* oleh pengguna dari *keyboard* tanpa menggunakan fasilitas *Swing* atau AWT.

Sekarang, kita akan mendeklarasikan variabel nama bertipe *String* untuk menyimpan input dari pengguna melalui keyboard :

```
String nama = "";
```

Variabel nama diinisialisasi sebagai *String* kosong "", Sebaiknya kita selalu menginisialisasi sebuah variabel setelah kita mendeklarasikannya.

Baris berikutnya adalah untuk menampilkan teks untuk berinteraksi dengan pengguna, yaitu untuk memerintahkan pengguna agar mengetikkan namanya.

```
System.out.print("Ketik nama anda : ");
```

Sekarang, blok di bawah ini merupakan blok *try-catch*,

```
Try {
   nama = dataIn.readLine();
}
catch( IOException e ) {
   System.out.println("Ada kesalahan !");
}
```

Untuk mengantisipasi jika terjadi kesalahan pada pernyataan:

```
nama = dataIn.readLine();
```

maka digunakan blok *try-catch* agar kesalahan yang terjadi tidak menggangu jalannya program. Kita akan membahas *try-catch* pada penanganan exception di bab selanjutnya. Sekarang kita cukup mencatat bahwa kita perlu menambahkan kode ini untuk menggunakan metode *readLine*() dari *BufferedReader* untuk mendapatkan masukan pengguna dari keyboard.

Selanjutnya kembali ke pernyataan :

```
nama = dataIn.readLine();
```

metode diatas memanggil *dataIn.readLine*(), untuk mendapatkan masukan dari pengguna melalui keyboard dan memberikan sebuah nilai *String*. Nilai ini akan disimpan ke dalam variabel nama, yang akan kita gunakan pada statement berikut ini :

```
System.out.println();
System.out.println("Hello " + nama +"\nLanjutkan belajarnya pasti
menjadi programmer Java !");
```

Baris pertama untuk membuat baris kosong, sehingga tulisan masukan dari pengguna memiliki jarak dengan tampilan berikutnya (Hello…..).

2.12.3. Menggunakan *JOptionPane*

Cara lain untuk mendapatkan masukan dari pengguna adalah dengan menggunakan *class JoptionPane* yang didapatkan dari paket *javax.swing*. *JoptionPane* memudahkan memunculkan kotak dialog standard yang memberikan kepada user sebuah nilai atau menginformasikan sesuatu.

Penggunaannya dapat dilihat pada program dibawah ini :

GetInputKeyboardJOptionPane.java

```
import javax.swing.JOptionPane;

public class GetInputKeyboardJOptionPane{
   public static void main( String[] args ){
      String nama = "";

      nama = JOptionPane.showInputDialog("Ketik nama anda : ");
      String msg = "Hello " + nama +"\nLanjutkan belajarnya pasti menjadi
programmer Java !";
      JOptionPane.showMessageDialog(null, msg);
   }
}
```

Tampilan dari program diatas adalah :

Gambar 2.4. Tampilan *input* menggunakan *JoptionPane*

Penjelasan program diatas adalah sebagai berikut :
Statement pertama :

```
import javax.swing.JOptionPane;
```

Menjelaskan bahwa kita mengimpor *class JoptionPane* dari paket *javax.swing*, atau bisa ditulis seperti pernyataan berikut ini :

```
import javax.swing.*;
```

Tiga baris pernyataan berikutnya sudah dijelaskan pada bagian input keyboard menggunakan BufferedReader. Pernyataan berikut ini :

```
nama = JOptionPane.showInputDialog("Ketik nama anda : ");
```

digunakan untuk membuat sebuah kotak dialog *JOptionPane*, yang akan menampilkan dialog dengan sebuah pesan, sebuah *textfield* dan tombol Ok seperti pada gambar 3.2. Hasil dari dialog tersebut adalah *String* dan disimpan ke dalam variabel nama.

Pernyataan selanjutnya adalah mendefinisikan variabel msg bertipe String dan diisi dengan serangkaian teks :

```
String msg = "Hello " + nama +"\nLanjutkan belajarnya pasti menjadi
programmer Java !";
```

Baris selanjutnya adalah menampilkan sebuah dialog yang berisi sebuah pesan dan tombol Ok :

```
JOptionPane.showMessageDialog(null, msg);
```

2.13. Format keluaran

Java memisahkan komponen untuk menampilkan keluaran dengan komponen untuk melakukan format keluaran. Keuntungan pemisahan ini antara lain dapat memberikan format keluaran yang lebih banyak, melebihi yang disediakan oleh C++.

Paket *java.text* menyediakan beragam format tampilan yang biasa ditemui di kehidupan sehari-hari. Terdapat tiga metode utama, yaitu format terhadap angka, mata uang dan persentase.

Untuk memperoleh *formatter* (obyek pelaku format) maka kita dapat memanggil :

- getNumberInstance()
- getCurrencyInstance()
- getPercentInstance()

Metode mengirim obyek bertipe *NumberFormat*. Kita dapat menggunakan obyek itu untuk melakukan format satu angka atau lebih. Kita kemudian menerapkan metode format ke obyek itu untuk melakukan format yang diinginkan, sehingga memperoleh string yang berisi angka yang telah terformat.

Metode di *NumberFormat* untuk mengkonfigurasi format angka dapat dilihat pada tabel 2.14 dibawah ini :

Tabel 2.14. Metode di *NumberFormat*

No	Metode	Deskripsi
1	`static NumberFormat getCurrencyInstance()`	Mengirim obyek NumberFormat yang bertugas mengkonversi ke string yang menunjukkan mata uang menggunakan konvensi lokal
2	`static NumberFormat getNumberInstance()`	Mengirim obyek NumberFormat yang bertugas mengkonversi ke string yang menunjukkan angka menggunakan konvensi lokal
3	`static NumberFormat getPercentInstance()`	Mengirim obyek NumberFormat yang bertugas mengkonversi ke string yang menunjukkan persentase menggunakan konvensi lokal
4	`void setMaximumFractionDigits (int digits)`	Mengeset jumlah angka maksimum setelah koma
5	`void setMaximumIntegerDigits( int digits)`	Mengeset jumlah angka maksimum sebelum koma
6	`void setMinimumFractionDigits (int digits)`	Mengeset jumlah angka minimum setelah koma
7	`void setMinimumFractionDigits (int digits)`	Mengeset jumlah angka minimum sebelum koma

Untuk menggunakan metode format, kita harus meng-*import java.text.** karena *NumberFormat* terdapat di paket *java.text*. Contoh aplikasi metode format dapat dilihat pada program dibawah ini :

FormatAngka.java

```
 import java.text.*;

 public class FormatAngka{
    public static void main(String args[]){
       double Angka=83243463.342245;
       double AngkaPecahan=0.902235643;

       NumberFormat NumberFormatter = NumberFormat.getNumberInstance();
       NumberFormat CurrFormatter = NumberFormat.getCurrencyInstance();
       NumberFormat PercentFormatter = NumberFormat.getPercentInstance();

       String NumberStr = NumberFormatter.format(Angka);
       String CurrStr = CurrFormatter.format(Angka);
       String PercentStr = PercentFormatter.format(AngkaPecahan);

       System.out.println("double Angka = "+ Angka +" berformat number :
"+NumberStr);
       System.out.println("double Angka = "+ Angka +" berformat currency :
"+CurrStr);
       System.out.println("double Angka = "+ AngkaPecahan +" berformat
percent : "+PercentStr);
       System.out.println();
       System.out.println();

       NumberFormatter.setMaximumIntegerDigits(10);
       CurrFormatter.setMaximumIntegerDigits(10);
       PercentFormatter.setMaximumIntegerDigits(10);

       NumberFormatter.setMinimumIntegerDigits(1);
       CurrFormatter.setMinimumIntegerDigits(1);
       PercentFormatter.setMinimumIntegerDigits(1);

       NumberFormatter.setMaximumFractionDigits(5);
       CurrFormatter.setMaximumFractionDigits(2);
       PercentFormatter.setMaximumFractionDigits(4);

       NumberFormatter.setMinimumFractionDigits(2);
       CurrFormatter.setMinimumFractionDigits(2);
       PercentFormatter.setMinimumFractionDigits(6);

       System.out.println("double Angka = "+ Angka +" berformat number :
"+NumberFormatter.format(Angka));
       System.out.println("double Angka = "+ Angka +" berformat currency :
"+CurrFormatter.format(Angka));
       System.out.println("double Angka = "+ AngkaPecahan +" berformat
percent : "+PercentFormatter.format(AngkaPecahan));
    }
 }
```

Keluaran dari program diatas adalah sebagai berikut :

```
 double Angka = 8.3243463342245E7 berformat number : 83.243.463,342
 double Angka = 8.3243463342245E7 berformat currency : Rp83.243.463,34
```

```
double Angka = 0.902235643 berformat percent : 90%


double Angka = 8.3243463342245E7 berformat number : 83.243.463,34224
double Angka = 8.3243463342245E7 berformat currency : Rp83.243.463,34
double Angka = 0.902235643 berformat percent : 90,223564%
```

Kita dapat memperoleh format angka yang cocok untuk lokal-lokal berbeda. Misalnya kita ingin membuat format sesuai lokal jerman, maka kita dapat memanggil metode :

- `getNumberInstance(Locale.GERMANY)`
- `getCurrencyInstance(Locale.GERMANY)`
- `getPercentInstance(Locale.GERMANY)`

Karena metode *Locale* berada di paket *java.util*, maka harus meng-*import java.util.** untuk dapat menggunakannya. Contoh penggunaan di program dapat dilihat pada program dibawah ini :

FormatAngkaLokal.java

```
import java.text.*;
import java.util.*;

public class FormatAngkaLokal{
  public static void main(String args[]){
     double Angka=83243463.342245;
     double AngkaPecahan=0.902235643;

     NumberFormat NumberFormatterGERMANY =
NumberFormat.getNumberInstance(Locale.GERMANY);
     NumberFormat CurrFormatterGERMANY =
NumberFormat.getCurrencyInstance(Locale.GERMANY);
     NumberFormat PercentFormatterGERMANY =
NumberFormat.getPercentInstance(Locale.GERMANY);

     String NumberStrGERMANY = NumberFormatterGERMANY.format(Angka);
     String CurrStrGERMANY = CurrFormatterGERMANY.format(Angka);
     String PercentStrGERMANY =
PercentFormatterGERMANY.format(AngkaPecahan);

     System.out.println("double Angka = "+ Angka +" berformat number :
"+NumberStrGERMANY);
     System.out.println("double Angka = "+ Angka +" berformat currency :
"+CurrStrGERMANY);
     System.out.println("double Angka = "+ AngkaPecahan +" berformat
percent : "+PercentStrGERMANY);
     System.out.println();
     System.out.println();

     NumberFormat NumberFormatterUS =
NumberFormat.getNumberInstance(Locale.US);
     NumberFormat CurrFormatterUS =
NumberFormat.getCurrencyInstance(Locale.US);
     NumberFormat PercentFormatterUS =
NumberFormat.getPercentInstance(Locale.US);

     String NumberStrUS = NumberFormatterUS.format(Angka);
     String CurrStrUS = CurrFormatterUS.format(Angka);
```

```
      String PercentStrUS = PercentFormatterUS.format(AngkaPecahan);

      System.out.println("double Angka = "+ Angka +" berformat number :
"+NumberStrUS);
      System.out.println("double Angka = "+ Angka +" berformat currency :
"+CurrStrUS);
      System.out.println("double Angka = "+ AngkaPecahan +" berformat
percent : "+PercentStrUS);
    }
  }
```

Keluaran dari program diatas adalah sebagai berikut :

```
double Angka = 8.3243463342245E7 berformat number : 83.243.463,342
double Angka = 8.3243463342245E7 berformat currency : 83.243.463,34 Ç
double Angka = 0.902235643 berformat percent : 90%


double Angka = 8.3243463342245E7 berformat number : 83,243,463.342
double Angka = 8.3243463342245E7 berformat currency : $83,243,463.34
double Angka = 0.902235643 berformat percent : 90%
```

Referensi:

1. Hariyanto, Bambang, (2007), *Esensi-esensi Bahasa Pemrograman Java*, Edisi 2, Informatika Bandung, November 2007.
2. Utomo, EkoPriyo, (2009), *Panduan Mudah Mengenal Bahasa Java*, Yrama Widya, Juni 2009.
3. Tim Pengembang JENI, JENI 1-6, Depdiknas, 2007
4. http://people.uncw.edu/tompkinsj/133/numbers/Reals.htm, diakses tanggal 10 Oktober 2010
5. http://www.janeg.ca/scjp/lang/charLiteral.html, diakses tanggal 10 Oktober 2010
6. http://www.javacamp.org/javaI/primitiveTypes.html, diakses tanggal 10 Oktober 2010