SINTAKS BAHASA DAN TIPE DATA

3.1 Contoh Program Java Sederhana

Setelah Anda mendapatkan gambaran yang cukup mengenai dasar-dasar OOP, kini saatnya kita memasuki dunia Java yang sebenarnya. Berikut ini contoh program Java yang sederhana:

```
/*
     Contoh program java sederhana
     -----------------------------
*/

class Hello{

     //awal program selalu dimulai dari main()
     public static void main(String[] args) {
            System.out.println("Hello Java...");
     }
}
```

Catatan Penting!

Java bersifat *case sensitive*. Huruf besar dan kecil adalah berbeda dalam Java. Oleh karena itu, pastikan mengetik code di atas sama persis, tanpa mengubah tipe huruf. Selain itu dalam Java, *white space* seperti karakter spasi, tab, pindah baris, dan karakter lainnya yang berfungsi

untuk memformat tampilan, tidak memiliki arti apa pun selain untuk memudahkan kita membaca code yang ditulis. Oleh karena itu, gunakan karakter white space ini sesuka Anda untuk memformat tampilan code yang ditulis, agar memudahkan Anda atau siapa pun membacanya.

Ketik program di atas menggunakan *text editor* (contoh: *Windows Notepad*) dan simpan dengan nama `Hello.java`.

Catatan Penting!

Untuk selanjutnya, kemungkinan Anda akan banyak bereksperimen dengan membuat class baru sendiri. Ingatlah, selalu simpan setiap class yang dibuat ke dalam filenya masing-masing yang sesuai dengan nama class-nya. Contohnya jika Anda membuat class **`Test`**, simpan ke dalam file **`Test.java`**.

Lakukan kompilasi dengan mengetikkan perintah berikut pada command prompt:

```
javac Hello.java
```

Contoh:

```
C:\java_projects>javac Hello.java
```

Setelah kompilasi, Anda akan mendapatkan *file bytecode* dengan nama `Hello.class`. Ketik perintah berikut untuk mengeksekusi program Java ini:

```
java Hello
```

Contoh:

```
C:\java_projects>java Hello
```

Ingat, pada perintah di atas, `Hello` adalah nama class, bukan nama file, karena itu jangan memasukkan nama ekstensi file `.class`. Sebagai hasil output dari program di atas, Anda akan mendapatkan hasil sebagai berikut:

```
Hello Java...
```

3.1.1 Penjelasan Program Baris per Baris

Program di atas sangat sederhana, namun memiliki beberapa bagian penting. Kita perjelas dengan melihat bagian-bagian tersebut satu per satu. Dimulai dengan bagian:

```
/*
     Contoh program java sederhana
     -----------------------------
*/
```

Bagian ini merupakan bagian yang dikenal sebagai komentar. Komentar bukan merupakan bagian dari program karena ia tidak akan di-compile oleh compiler Java dan tidak akan mempengaruhi alur eksekusi program. Komentar ini lebih ditujukan untuk memperjelas *source code* yang kita tulis sehingga code akan lebih mudah dibaca oleh siapa pun. Komentar di atas ditulis di antara tanda `/*` dan `*/`.

```
class Hello {
```

Baris code ini mendeklarasikan sebuah class dengan nama `Hello`. Perhatikan bahwa keseluruhan deklarasi dari class ini diawali dengan `{` dan ditutup dengan `}`. Semua code yang terletak di antara `{ }` merupakan bagian dari class ini.

Class VS Objek

Jika sampai saat ini Anda masih dibingungkan dengan hubungan antara class dan objek, maka penjelasan berikut ini mungkin dapat sedikit memberikan pencerahan. Class dan objek memang memiliki hubungan yang erat. Class sendiri dapat dikatakan sebagai spesifikasi/desain dari objek, atau mungkin jika Anda lebih suka dengan istilah lain, yaitu *Blue Print*.

Jika Anda seorang arsitek, yang harus dilakukan sebelum membangun sebuah gedung adalah membuat desain gedung yang akan dibangun di atas kertas, baru kemudian

gedung yang sesungguhnya dibangun berdasarkan desain tersebut. Dalam hal ini, desain yang dibuat adalah class itu sendiri, sedangkan gedung yang dibangun berdasarkan desain tersebut adalah objeknya. Dengan demikian, jelas bahwa class adalah suatu spesifikasi/desain dari objek, sedangkan objek sendiri adalah *instance* (perwujudan) dari class. Alasan ini juga yang menyebabkan kata instance sering digunakan sebagai ganti dari objek karena memiliki arti yang sama.

```
//awal program selalu dimulai dari main()
```

Baris code ini juga merupakan komentar. Perbedaannya dengan `/* */` adalah bahwa `//` hanya berlaku untuk satu baris ini saja dan semua kata-kata setelah tanda `//` merupakan komentar.

Beberapa bagian dari code ini akan terasa sulit di sini karena membutuhkan pengenalan akan bahasa Java yang lebih mendalam. Anda tidak perlu khawatir, penjelasan singkat akan tetap diberikan. Teruslah membaca sekalipun masih kurang memahaminya karena Anda pasti akan mendapatkan penjelasan yang lebih mendalam pada bab-bab selanjutnya.

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

Baris code di atas mendeklarasikan suatu *method* dengan nama *main*. Nama main di sini merupakan suatu keharusan dalam Java karena Java akan mencari method yang bernama main ini sebagai titik awal eksekusi program. *Keyword* **public** merupakan *access specifier* yang menentukan *visibility level* dari method ini. *Public* berarti method ini dapat diakses/dipanggil dari luar class di mana ia dideklarasikan. Method `main()` akan dipanggil dari luar oleh *runtime* Java pada saat program akan dieksekusi sehingga access specifier yang dimilikinya haruslah **public**. Keyword **static** memungkinkan method `main()` dipanggil tanpa harus terlebih dahulu membuat instance dari class `Hello`. Ini diperlukan karena method `main()` akan dieksekusi sebelum objek dari class `Hello` dibuat di memori. Keyword **void** berarti bahwa method `main()` tidak mengembalikan nilai apa pun setelah dipanggil/dieksekusi.

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

Baris code ini memiliki beberapa bagian penting, antara lain:

- **`System`** adalah nama dari salah satu class standar yang dimiliki oleh Java.
- **`out`** merupakan anggota dari class `System` dan juga merupakan objek tersendiri, `out` merupakan objek yang mewakili *standard output stream* yang dalam hal ini adalah layar komputer. Seperti halnya dengan method `main()` di atas, objek `out` ini dideklarasikan menggunakan keyword **static** di dalam class-nya sehingga ia dapat langsung dipanggil tanpa perlu terlebih dulu membuat instance dari class `System`.
- **`println`** merupakan method yang terdapat pada objek `out`. Berfungsi untuk mencetak keluaran ke *standard output*. Method ini juga mencetak karakter pindah baris. Untuk mencetak keluaran ke standard output tanpa mencetak karakter pindah baris, Anda dapat menggunakan method `print()`.
- **`"Hello Java…"`** merupakan parameter dari method `println()` yang diterima oleh internal method ini dan dicetak ke standard output yang dalam hal ini adalah layar komputer sehingga pada saat Anda menjalankan program ini, di layar komputer akan tercetak `Hello Java...`
- Tanda **`;`** yang Anda lihat di paling belakang baris diperlukan untuk menandakan akhir suatu *statement*/pernyataan/perintah.

Anda Merasa Tersesat?

Saat ini mungkin Anda telah mulai merasa kehilangan arah dan penjelasan tentang baris program di atas kelihatan tidak masuk akal, tetapi teruslah membaca. Penjelasan yang lebih detail akan Anda dapatkan pada bab-bab berikutnya dari buku ini.

3.2 Identifier

Identifier merupakan nama yang digunakan untuk menamai class, interface, variabel, dan method. Anda dapat menentukan sendiri identifier yang akan digunakan. Yang perlu diperhatikan mengenai identifier ini adalah:

- Tidak ada batasan panjang, jadi kita dapat membuat identifier dengan panjang berapa pun.
- Identifier harus diawali dengan huruf, *underscore*/garis bawah `(_)` atau lambang dolar `($)`. Untuk selebihnya dapat menggunakan karakter apa pun, kecuali karakter yang digunakan sebagai operator oleh Java (seperti `+`, `-`, `*`, atau `/`).
- Bukan merupakan keywords yang dikenal oleh Java.

Pada contoh program sebelumnya, `Hello` merupakan identifier yang digunakan untuk menamai class yang dibuat.

Contoh identifier yang valid:

```
strTemp     $counter    b4Th33      _aValidOne
```

Contoh identifier yang tidak valid:

```
4Ever       from-to     Good/Bad    *by4
```

3.3 Reserved Words/Keywords

Merupakan kata-kata yang dikenal oleh kompiler Java dan memiliki arti khusus dalam program. Berikut ini daftar kata-kata tersebut.

assert*	abstract	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	enum**	false	final	finally
float	for	goto	if	implements
import	instanceof	int	interface	long
native	new	null	package	private

protected	public	return	short	static
strictfp	super	switch	synchronized	this
throw	throws	transient	true	try
void	volatile	while		

** keyword tambahan yang dikenal mulai JDK 1.4*

*** keyword tambahan yang dikenal mulai JDK 1.5*

Tabel 3.1 Daftar Keywords yang dikenal oleh Java

3.4 Variabel

Variabel merupakan lokasi penyimpanan yang ada di memori. Setiap variabel memiliki kemampuan menyimpan suatu informasi sesuai dengan tipe data yang dideklarasikan untuk variabel tersebut saja. Sintaks pendeklarasian 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 akan digunakan untuk merujuk ke variabel tersebut di dalam program.

Contoh code:

```
int counter;
```

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

3.4.1 Scope dari Variabel

Dalam Java, secara garis besar *scope* dari variabel dapat dibedakan menjadi dua bagian, yaitu variabel yang dideklarasikan di dalam blok class (dikenal juga sebagai property) dan variabel yang dideklarasikan di dalam blok code.

Variabel yang dideklarasikan dalam blok class akan dikenal di bagian manapun dalam blok class tersebut. Variabel ini juga bahkan dapat diakses dari luar class menggunakan referensi objek atau instance dari class tersebut. Namun hal ini akan dipengaruhi oleh penggunaan access specifier. Access specifier akan kita bahas pada saat kita memasuki pembahasan tentang class. Untuk sementara ini demi kemudahan, di dalam contoh program di buku ini, setiap variabel yang dideklarasikan di dalam class tidak akan menggunakan access specifier apa pun.

Pada dasarnya variabel dapat dideklarasikan di dalam blok code manapun. Yang dimaksud dengan blok code di sini adalah bagian dari code yang dimulai dengan karakter **{** dan ditutup dengan karakter **}**. Blok ini menentukan scope dari suatu variabel, yaitu apakah suatu variabel akan dikenal di bagian lain dari program atau tidak. Perhatikan juga bahwa Anda dapat membuat *nested* blok, di mana di dalam suatu blok code terdapat blok code lainnya. Aturan sederhana yang perlu diingat hanyalah, pendeklarasian suatu variabel dalam suatu blok code akan dikenal oleh nested blok yang ada di dalam blok code tersebut, namun tidak berlaku sebaliknya. Untuk lebih jelasnya, perhatikan contoh code berikut ini:

```
class Scope {

static int a = 2; //deklarasi variabel dalam blok class
public static void main(String[] args) {
   int x;  //variabel x ini dikenal di seluruh method main()
   x = 10;

   //variabel a juga dikenal di sini
   System.out.println("Nilai a : " + a);

   { //awal dari blok baru
     int y; //variabel ini hanya dikenal di dalam
            //blok code ini saja

     y = 5;

     //variabel x dikenal di sini
     System.out.println("Nilai x : " + x);

     //variabel a juga dikenal di sini
     System.out.println("Nilai a : " + a);


    { //nested blok
        int z;//variabel ini hanya dikenal di
              //dalam nested blok ini saja
```

```
            z = 20;

            /* variabel x,y dan a dikenal di dalam
               nested blok ini */

            System.out.println("Nilai x+y+z+a : "
                               + (x + y + z + a));
        } //akhir dari nested blok

        //z = 11; // -> Error variabel z tidak lagi
                  //dikenal di sini

        /* variabel y masih dikenal di sini karena
           masih dalam blok code tempat ia
           dideklarasikan */

        System.out.println("Nilai y : " + y);

    }  //akhir dari blok baru

    //y = 12; // -> Error: variabel y tidak dikenal di sini

    /* variabel x masih dikenal di sini karena masih
       dalam blok code tempat ia dideklarasikan */

    System.out.println("Nilai x : " + x);

    } //akhir dari method main

    } //akhir dari deklarasi class
```

Deklarasi variabel dapat diletakkan di baris manapun dari program, tetapi perhatikan juga bahwa Anda tidak dapat menggunakan variabel yang belum dideklarasikan, sebagai contoh:

```
index = 0; // -> Erorr, variabel index belum
           //    dideklarasikan
int index; // -> di sini index baru
           //    dideklarasikan sebagai tipe int
```

Anda juga dapat langsung memberikan nilai pada suatu variabel pada saat mendeklarasikannya, sebagai contoh:

```
int index = 10;
```

Harus diingat bahwa variabel yang telah dideklarasikan dalam blok class, dapat dideklarasikan ulang (dengan nama yang sama) dalam blok code, namun variabel yang telah dideklarasikan dalam suatu blok code, tidak boleh dideklarasikan lagi dalam scope blok code yang sama (di dalam scope di mana variabel tersebut masih dikenal). Perhatikan contoh berikut ini:

```
class Scope {
    int x; //➔ deklarasi variabel int dengan nama
           //   x dalam blok class

    public static void main(String[] args) {
            int x; // ➔ deklarasi variabel int
                   // dengan nama x dalam blok code

     // baris code lainnya...

            {   // awal dari blok(scope) baru

            int x;//➔ Error, variabel x telah
                  // dideklarasikan sebelumnya

                // baris code lainnya...
            }

            int x; //➔ Error, variabel x telah
                   // dideklarasikan sebelumnya pada
                     // blok code yang sama

    } //akhir dari method main
}
```

3.5 Tipe Data

Tipe data diperlukan agar kompiler tahu operasi apa yang valid dan seberapa banyak memori yang diperlukan oleh sebuah nilai yang akan disimpan atau dioperasikan. Variabel digunakan untuk menampung suatu nilai, karena itu setiap variabel pasti memiliki tipe data dan harus dideklarasikan terlebih dahulu sebelum dapat digunakan. Perbedaan tipe data juga mengakibatkan setiap operasi penugasan (lihat Bab 4 tentang operator), baik secara eksplisit maupun melalui *passing* parameter pada waktu pemanggilan method (akan dibahas nanti), selalu dicek kompatibilitas tipe datanya. Dalam Java, tipe data dapat dikelompokkan menjadi dua jenis tipe data, yaitu tipe data primitif dan referensi.

3.5.1 Tipe Data Primitif

Tipe data primitif merupakan tipe data dasar yang dikenal oleh Java. Terdapat delapan buah tipe data primitif yang dikenal dalam Java sebagai berikut.

Nama Tipe Data	Ukuran dalam bit	Nilai	Standar
byte	8	-2^7 s.d. $2^7 - 1$	
short	16	-2^{15} s.d. $2^{15} - 1$	
Int	32	-2^{31} s.d. $2^{31} - 1$	
long	64	-2^{63} s.d. $2^{63} - 1$	
float	32	Negatif: -3.4028234663852886E+38 s.d. -1.40129846432481707e-45 Positif: 1.40129846432481707e-45 s.d. 3.4028234663852886E+38	IEEE 754 floating point
double	64	Negatif: -1.7976931348623157E+308 s.d. -4.94065645841246544e-324 Positif: 4.94065645841246544e-324 s.d. 1.7976931348623157E+308	IEEE 754 floating point
char	16	'\u0000' s/d '\uFFFF' (0 s/d 65535)	ISO Unicode Charater Set
boolean	8	true atau false	

Tabel 3.2 Tipe data primitif dalam Java

Kedelapan tipe data primitif ini dapat dikelompokkan ke dalam empat group:

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 atas **float** dan **double**.
3. *Karakter* mewakili simbol dari sebuah karakter yang terdiri atas **char**.
4. *Boolean* merupakan tipe data yang menunjukkan nilai **true** atau **false**, yang terdiri atas **boolean**.

Variabel dengan tipe data primitif ini dapat digunakan secara langsung untuk menyimpan suatu nilai tertentu.

Contoh:

```
int indeks = 2;
int hasil  = indeks * 11;
```

3.5.2 Tipe Data Referensi

Tipe data referensi digunakan untuk memegang referensi dari suatu objek (*instance* dari *class*). Pendeklarasian variabel tipe data ini sendiri sama dengan tipe data primitif. Namun, penggunaannya agak sedikit berbeda.

Perhatikan contoh code di bawah ini untuk lebih jelasnya:

```
class ProgSederhana {
     int x = 0;  /* Pendeklarasian variabel dengan
                    tipe data int */

     public static void main(String[] args) {

          /*Pendeklarasian variabel dengan tipe data
            class ProgSederhana*/
         ProgSederhana var;

          //Instantiate class ProgSederhana menjadi
          //objek
          var = new ProgSederhana();

          /*setelah proses instantiate ini, Anda
            dapat mengakses objek ProgSederhana
            melalui variabel var*/
          var.x = 20;
          System.out.println("Nilai x : " + var.x);
     }
}
```

Code berikut ini:

```
var = new ProgSederhana();
```

memerintahkan *run-time* Java untuk mengalokasikan memori membuat objek `ProgSederhana` di memori dan menyerahkan referensi (alamat) dari objek tersebut untuk dipegang oleh variabel `var`. Inilah yang kita sebut dengan *instantiate class* menjadi sebuah objek. Dengan variabel `var` ini, Anda dapat mengakses internal objek yang telah dibuat tersebut:

```
var.x = 20;
```

Penjelasan yang lebih lengkap untuk tipe data ini akan Anda dapatkan setelah mendalami lebih lanjut tentang class pada Bab 6.

3.5.3 Penjelasan Tentang Nilai Literal

Literal adalah suatu nilai konstan yang terlihat secara eksplisit.

Perhatikan contoh code berikut ini:

```
int index = 100;
index = index * 11;
```

Pada code di atas, angka `100` merupakan nilai literal, begitu juga dengan angka `11` pada baris berikutnya. Notasi `index * 11` mempunyai arti bahwa nilai yang dipegang oleh variabel `index` (`index` bukan berupa nilai literal melainkan variabel) dikalikan dengan nilai literal `11` dan hasilnya ditampung kembali oleh variabel `index`. Ada baiknya Anda mencoba membuat sebuah *class* baru dan mengetik perintah di atas untuk melihat hasilnya. Tambahkan code berikut ini untuk melihat nilai dari variabel `index`:

```
System.out.println(index);
```

Literal Integer

Semua angka berupa bilangan bulat yang ditulis dalam program akan dikenal sebagai tipe data **int** oleh Java. Contohnya `1`, `10`, `19`, `30`, dan seterusnya. Pada contoh ini, nilai literal **int** ditulis berupa bilangan berbasis 10 (desimal). Anda juga dapat menulisnya dalam basis 8 (oktal) atau 16 (heksadesimal). Untuk menulis nilai literal dalam bentuk oktal, Anda dapat menuliskannya dengan angka **0** di depannya, contoh `01`, `05`, `0456`. Digit untuk bilangan oktal memiliki *range* antara `0 - 7`, karenanya jangan menggunakan bilangan yang lebih besar dari angka `7` untuk menuliskan bilangan oktal. Untuk bilangan heksadesimal, Anda dapat menulisnya dengan didahului oleh tanda **0x** (nol dan `x`) atau **0X** (nol dan `X`), contohnya `0x1`, `0X92`, `0xA`, dan seterusnya. Digit bilangan heksadesimal memiliki range antara `0-15`, untuk digit `10-15` digunakan huruf `A-F` atau `a-f` sebagai lambangnya.

Untuk menuliskan angka yang memiliki ukuran yang sangat besar, di mana hanya tipe data **long** (64 bit) yang dapat menampungnya,

Anda perlu menambahkan karakter `L` atau `l` di akhir nilai literal yang Anda ketik, untuk memberi tahu Java bahwa nilai literal yang diketik tersebut bertipe data **long** dan bukan bertipe **int**. Contoh `0x7fffffffffffff`**L** dan `903487261738384756`8**L**.

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`, `11.13`, `99.9`, dan seterusnya. Sementara untuk notasi ilmiah dapat digunakan dengan menambah lambang **En** pada notasi standard floating-point. **E** (eksponensial) berarti bahwa nilai floating-point-nya harus dikalikan dengan nilai `10` sebanyak `n` kali untuk mendapatkan nilai yang sebenarnya, contoh:

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

Secara default semua nilai literal floating-point yang ditulis dalam Java akan dianggap memiliki tipe data **double**. Akan tetapi, secara eksplisit Anda dapat menentukan tipe data dari nilai literal yang ditulis dengan menambahkan lambang **F** atau **f** untuk tipe data **float** dan **D** atau **d** untuk tipe data **double**.

Kesalahan umum yang sering terjadi adalah kebanyakan orang mencoba menuliskan code seperti berikut ini:

```
float num = 10.5;
```

Nilai literal `10.5` di atas secara default akan bertipe data **double** sehingga menimbulkan kesalahan karena **double** memiliki ukuran lebih besar daripada **float**. Untuk memperbaikinya, Anda dapat menambahkan tanda **F** atau **f** di belakang nilai literal tersebut, seperti berikut ini:

```
float num = 10.5f;
```

Literal Boolean

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

```
boolean value1 = true ;
boolean value2 = false;
```

> **Catatan kecil bagi programmer C/C++**
>
> Para programmer yang biasa menggunakan C/C++ harus memperhatikan bahwa nilai literal **true** dan **false** tidak dapat dikonversi ke dalam bentuk representasi numerik seperti halnya di dalam bahasa C/C++. Dalam C/C++ semua selain **0** adalah **true**. Dalam Java hal ini tidak berlaku.

Literal Character

Java didesain untuk menjadi bahasa yang portabel dan universal. Oleh 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 untuk tipe data **char** adalah 16 bit agar dapat merepresentasikan semua karakter yang ada. Untuk kode '**ASCII**' yang telah dikenal luas, kita dapat langsung menulisnya di dalam tanpa petik, contoh **'a'**, **'$'**, **'*'**, dan seterusnya. Namun, untuk beberapa karakter yang tidak dapat kita masukkan dengan mengetik secara langsung melalui keyboard (seperti karakter ganti baris, tab, backspace, dan lain-lain), dapat digunakan apa yang dikenal sebagai *escape sequences*. Cara menulisnya menggunakan karakter **** (***backslash***). Contohnya, untuk karakter ganti baris, kita dapat menulis **'\n'**. Selain itu, juga dapat secara langsung menulis nilai *Unicode* dari karakter yang diinginkan menggunakan bilangan oktal atau heksadesimalnya. Jika menggunakan bilangan oktal, dapat langsung ditulis dengan format **'\ddd'**, di mana nilai `ddd` diganti dengan bilangan oktal. Contohnya **'\141'** untuk karakter **'a'**. Dan jika menggunakan heksadesimal Anda dapat menggunakan format penulisan **'\uxxxx'** di mana nilai `xxxx` diganti dengan bilangan heksadesimal. Contohnya **'\u0061'** untuk karakter **'a'**.

Berikut ini daftar escape sequence yang dikenal oleh Java.

Escape Sequences	Deskripsi
\ddd	**karakter Oktal(ddd)**
\uxxxx	**karakter Heksadesimal(xxxx)**
\'	**tanda petik**
\"	**tanda petik ganda**
****	**tanda \ (*backslash*)**
\r	***carriage return***
\n	**pindah baris**
\f	***form feed***
\t	***tab***
\b	***backspace***

Tabel 3.3 Daftar Escape Sequences yang dikenal oleh Java

Literal String

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

```
"Literal String"
"Baris 1\nBaris 2"
"\"Tanda petik ganda\" dan tanda backslash \\"
```

Untuk melihat efek penggunaan escape sequences ini, Anda dapat menggunakan perintah `System.out.println()` dalam program.

Hasilnya sebagai berikut:

```
Literal String
Baris 1
Baris 2
"Tanda petik ganda" dan tanda backslash \
```

3.5.4 Array

Seperti yang telah kita bahas sebelumnya bahwa setiap kali hendak menggunakan suatu variabel, kita harus terlebih dahulu mendeklarasikannya. Yang menjadi masalah adalah bagaimana jika hendak menggunakan sekumpulan variabel yang sangat banyak dengan tipe data tertentu, misalnya membutuhkan 1000 buah variabel bertipe **int** untuk suatu perhitungan. Sangat tidak efisien jika kita harus mendeklarasikan ke 1000 variabel tersebut satu per satu. Oleh karena itulah Java memiliki tipe data *array*.

Array Satu Dimensi

Pada dasarnya array satu dimensi merupakan kumpulan dari variabel yang memiliki tipe data yang sama.

Pendeklarasian tipe data array memiliki dua bentuk:

```
tipe-data   nama-array[];
tipe-data[] nama-array;
```

Variasi cara pendeklarasian ini hanyalah untuk memudahkan programmer, bentuk mana yang digunakan adalah terserah kepada programmer karena pada dasarnya keduanya sama.

`Tipe-data` merupakan tipe data yang akan ditampung oleh variabel array ini. Semua tipe data yang dikenal dalam Java, baik yang primitif maupun berupa class dapat digunakan di sini. Untuk `nama-array`, merupakan identifier yang menunjukkan nama dari variabel array itu sendiri sama seperti nama variabel biasa. Tanda **[]** memberitahukan Java bahwa variabel ini bertipe array.

Contoh code:

```
int   arr1[];
int[] arr2;
```

Code di atas mendeklarasikan variabel array `arr1` dan `arr2` dengan tipe data **int**. Perlu diperhatikan di sini, Java memperlakukan array sama dengan tipe data referensi di mana variabel ini sebenarnya hanya digunakan untuk menyimpan referensi dari lokasi array yang sebenarnya di memori. Oleh karena itu, Anda harus mengalokasikan terlebih dahulu tempat di memori sebelum dapat mengakses array

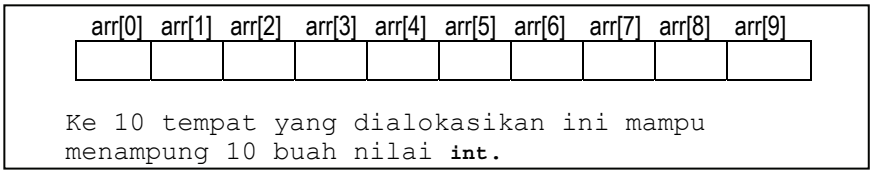
menggunakan variabel ini. Pendeklarasian yang ditunjukkan oleh code di atas hanya mendeklarasikan suatu variabel array dengan tipe data **int** bernama `arr1` dan `arr2`, namun tempat yang akan digunakan untuk menampung array itu sendiri belum dibuat di memori. Berikut ini bentuk umum pengalokasian memori untuk array:

```
nama-array = new tipe-data[ukuran-array];
```

Untuk `ukuran-array` harus berupa bilangan bulat, dan jumlah maksimum yang dapat dimasukkan sebagai `ukuran-array` hanya terbatas pada seberapa banyak memori yang tersedia pada komputer. Berikut ini contoh pengalokasian memori:

```
int[] arr = new int[10];
```

Code di atas mengalokasikan variabel `arr` dengan tipe data **int** (array of **int**) sebanyak 10 buah. Array ini dapat diilustrasikan seperti gambar di bawah ini.



Gambar 3.1 Ilustrasi array satu dimensi berukuran 10

Untuk mengakses nilai yang ditampung dalam sebuah array, kita menggunakan indeks yang menentukan urutan dari array yang akan diakses. Indeks untuk array dalam Java selalu dimulai dari nol. Perhatikan contoh program sederhana berikut ini untuk melihat penggunaan array:

```
class SingleArray {
     public static void main(String[] args) {
       int[] x = new int[3];

          x[0] = 20;
          x[1] = 10;
          x[2] = 0;

          System.out.println("Nilai x[0] : "+ x[0]);
          System.out.println("Nilai x[1] : "+ x[1]);
          System.out.println("Nilai x[2] : "+ x[2]);
     }
}
```

Variabel `x` pada program di atas dideklarasikan bertipe array of **int**, di mana ukuran array yang dialokasikan mampu menampung 3 buah data bertipe **int**. Pengaksesan variabel `x` dilakukan menggunakan indeksnya. Hasil eksekusi program di atas adalah sebagai berikut:

```
Nilai x[0] : 20
Nilai x[1] : 10
Nilai x[2] : 0
```

Perlu diperhatikan di sini bahwa kita tidak dapat/tidak boleh mengakses array di luar indeksnya, contoh:

```
int[] x = new int[3];
x[10] = 100;
```

Baris ini akan mengakibatkan eksepsi berupa **java.lang.ArrayIndexOutOfBoundsException**

Jika mengetik code di atas pada program, maka pada saat Anda mengeksekusinya, Anda akan mendapatkan eksepsi berupa:

```
java.lang.ArrayIndexOutOfBoundsException
```

Penjelasan tentang eksepsi akan dibahas pada Bab 10 nanti. Untuk sekarang Anda dapat mengacuhkannya dahulu.

Pengalokasian kapasitas dari array juga dapat dilakukan pada saat pendeklarasian, contoh:

```
int[] x = new int[3];
```

Bahkan dapat langsung menginisialisasi nilai dari array tersebut pada saat mendeklarasikannya, contoh:

```
int[] x = {0,1,2};
```

Pada contoh code di atas, secara otomatis akan dialokasikan tempat di memori yang dapat menampung 3 buah tipe data **int** untuk variabel `x`, di mana nilai pada masing-masing indeksnya akan langsung diinisialisasi seperti berikut ini `x[0]=0`, `x[1]=1`, `x[2]=2`. Perlu diperhatikan di sini, Anda hanya dapat melakukan inisialisasi seperti ini pada saat Anda mendeklarasikan array tersebut. Oleh karena itu, contoh code berikut ini akan menghasilkan pesan kesalahan pada waktu hendak di-compile:

```
int[] x = new x[3];
x = {0,1,2};        // Error...
```

Array Multi Dimensi

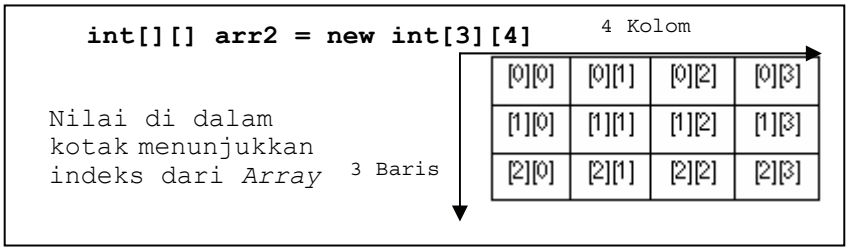
Untuk menyederhanakan pengertian array multidimensi, kita dapat mengartikannya sebagai ***array of array***, yaitu array yang terdapat di dalam array. Cara pendeklarasian array multidimensi ini, pada dasarnya sama dengan array satu dimensi, di mana cukup menambahkan `[]` sesuai dengan dimensi yang kita 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];
```

Code di atas akan mengalokasikan memori untuk menampung nilai tipe data **int** sebanyak `3*4`. Untuk ilustrasi, array ini dapat digambarkan seperti gambar di bawah ini.



*Gambar 3.2 Ilustrasi gambar array berukuran 3*4*

Pada prinsipnya, cara penggunaan array untuk dimensi berapa pun adalah sama. Oleh karena itu, untuk kemudahan kita hanya akan menggunakan array dua dimensi untuk menjelaskan tentang array multidimensi ini. Hal lain yang perlu diketahui adalah pada saat pengalokasian memori, Anda dapat mengalokasikan memori hanya untuk dimensi yang paling kiri, contoh:

```
int[][]  arr2;      //Pendeklarasian Array 2 dimensi

arr2     = new int[3][]; /* Alokasi memori untuk dimensi yang
                            paling kiri saja */
```

```
/* Alokasi memori untuk dimensi berikutnya */

arr2[0] = new int[4];
arr2[1] = new int[4];
arr2[2] = new int[4];
```

Alokasi dengan cara di atas sama dengan mengalokasikan memori dengan cara:

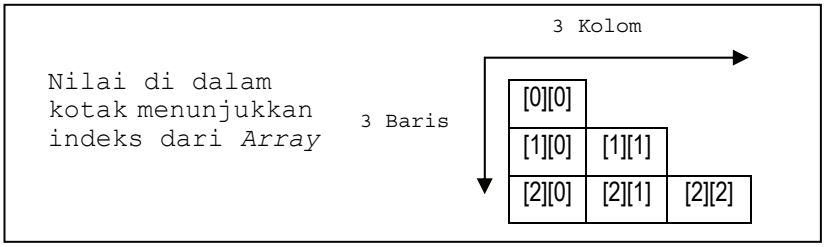
```
arr2 = new int[3][4];
```

Namun, dengan cara pengalokasian memori mulai dari dimensi yang paling kiri, Anda dapat mengalokasikan array multidimensi yang tidak simetris, contoh:

```
int[][]  arr2;

arr2    = new int[3][];
arr2[0] = new int[1];
arr2[1] = new int[2];
arr2[2] = new int[3];
```

Ilustrasi *array* seperti ini dapat digambarkan seperti berikut ini.



Gambar 3.3 Ilustrasi gambar array yang tidak simetris

Walaupun pengalokasian kapasitas array dapat dilakukan secara tidak simetris, sangat tidak disarankan untuk menggunakan array multidimensi yang tidak simetris seperti ini karena orang punya kecenderungan untuk menganggap bahwa suatu array biasanya memiliki ukuran simetris sehingga dapat menimbulkan kebingungan bagi orang lain yang akan membaca program Anda. Namun, terkadang kita dapat menggunakan array yang tidak simetris ini jika memang membutuhkan array multidimensi yang sangat besar. Namun, tidak semua tempat dalam array tersebut akan digunakan. Jika harus membuat array multidimensi yang simetris, akan banyak tempat dalam array yang tidak digunakan, yang berarti penyia-

nyiaan memori. Dalam hal ini, penggunaan array multidimensi yang tidak simetris mungkin merupakan solusi terbaik.

Seperti yang telah disebutkan di atas, sebenarnya array multidimensi adalah array yang terdapat dalam array. Dengan konsep ini, Anda dapat melakukan inisialisasi terhadap array multidimensi sama seperti menginisialisasi array satu dimensi.

Contoh inisialisasi array dua dimensi:

```
int[][] arr2 = {{11,12,13},
                {21,22,23},
                {31,32,33}};
```

Array di atas memiliki ukuran `3*3` di mana nilai dari setiap elemennya:

```
arr2[0][0] = 11     arr2[0][1] = 12        arr2[0][2] = 13
arr2[1][0] = 21     arr2[1][1] = 22        arr2[1][2] = 23
arr2[2][0] = 31     arr2[2][1] = 32        arr2[2][2] = 33
```

Contoh inisialisasi array tiga dimensi:

```
int[][][] arr3 = {{{111,112,113},{121,122,123}},
                  {{211,212,213},{221,222,223}},
                  {{311,312,313},{321,322,323}}};
```

Array di atas memiliki ukuran `3*2*3` di mana nilai dari setiap elemennya:

```
arr3[0][0][0] = 111 arr3[1][0][0] = 211
arr3[0][0][1] = 112 arr3[1][0][1] = 212
arr3[0][0][2] = 113 arr3[1][0][2] = 213

arr3[0][1][0] = 121 arr3[1][1][0] = 221
arr3[0][1][1] = 122 arr3[1][1][1] = 222
arr3[0][1][2] = 123 arr3[1][1][2] = 223

arr3[2][0][0] = 311 arr3[2][1][0] = 321
arr3[2][0][1] = 312 arr3[2][1][1] = 322
arr3[2][0][2] = 313 arr3[2][1][2] = 333
```

Satu hal lagi yang perlu diperhatikan tentang tipe data array, yaitu bahwa Anda dapat mengetahui kapasitas atau panjang dari suatu array dengan mengakses property yang dimiliki oleh objek array, yaitu **length**. Perhatikan contoh penggunaannya berikut ini:

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

     // array 3*2
```

```
        int arr[][] = {{1,2},{3,4},{5,6}};

        System.out.println("Panjang dimensi pertama:"
                           + arr.length);

        System.out.println("Panjang dimensi kedua:"
                           + arr[0].length);
    }
}
```

Hasil eksekusi program:

```
Panjang dimensi pertama:3
Panjang dimensi kedua:2
```

3.5.5 Konversi Tipe Data dan Casting

Adalah hal yang biasa dalam pemrograman untuk menampung suatu nilai dengan tipe data tertentu ke dalam variabel yang mempunyai tipe data yang berbeda. Java akan melakukan konversi tipe data secara otomatis jika kedua tipe data tersebut kompatibel. Misalnya dari tipe data **int** ke tipe data **long**.

Contoh code:

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

Variabel `data1` yang bertipe **int** ditampung ke dalam variabel `data2` yang bertipe **long**. Dengan demikian, terjadi konversi tipe data, dan dalam hal ini konversi dilakukan secara otomatis oleh Java tanpa membutuhkan tambahan code apa pun.

Namun, tidak semua tipe data kompatibel satu dengan yang lainnya. Misalnya tipe data **float** dengan tipe data **int**, **float** merupakan tipe data pecahan (*floating-point*), sedangkan **int** adalah tipe data bilangan bulat (*integer*). Hal yang sama juga terjadi jika Anda hendak mengonversi tipe data yang lebih besar ke tipe data yang lebih kecil, seperti dari **int** ke **short**. Konversi tipe data yang tidak kompatibel ini masih memungkinkan, namun harus menyebutkan secara eksplisit tipe data tujuan (*casting*).

Sintaks code:

```
(target-tipe-data)nilai
```

`target-tipe-data` : tipe data yang menjadi tujuan konversi.

`Nilai` : dapat berupa nilai literal atau berupa variabel.

Contoh code:

```
Float data1 = 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 (floating-point) ke tipe data bilangan bulat (Integer), maka akan terjadi pemotongan (truncation). Dalam contoh di atas, `data2` akan bernilai `10`. Sedangkan untuk tipe data yang lebih kecil jika digunakan untuk menampung data yang lebih besar dari daya tampungnya, maka yang akan tertampung adalah nilai modulusnya (sisa bagi). Dalam 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` diperoleh modulus = `1` maka `data4` akan bernilai `1`.

Anda juga perlu memperhatikan bahwa di dalam Java dikenal apa yang disebut promosi tipe data secara otomatis dalam suatu ekspresi matematik. Untuk lebih jelasnya perhatikan contoh code berikut ini:

```
byte  data1 = 10;
byte  data2 = 50;
byte  data3 = data1 * data2;
```

Variabel `data1` dan `data2` memiliki nilai yang terdapat dalam *range byte*. Namun, hasil perkalian `data1` dan `data2` akan menghasilkan nilai `500` yang tidak dapat lagi ditampung oleh tipe data **byte**. Karena alasan ini, maka pada waktu melakukan komputasi, Java akan melakukan promosi tipe data **byte** ke tipe data **int** supaya hasil perhitungannya benar. Yang menjadi masalah adalah, promosi tipe data secara otomatis tersebut akan tetap dilakukan, sekalipun hasil dari perkalian variabel `data1` dan `data2` tidak melebihi kapasitas tampung tipe data **byte**.

Contoh code:

```
byte  data1 = 20;
byte  data2 = data1 * 5;
```

Error karena hasil komputasi dari `data1*5` adalah berupa **int** sehingga tidak dapat ditampung oleh variabel `data2` yang bertipe **byte**.

Hasil dari `data1*5` adalah `100` yang seharusnya dapat ditampung oleh tipe data **byte**, namun karena adanya promosi tipe data secara otomatis, nilai `100` tersebut bukan lagi bertipe **byte** melainkan bertipe **int**. Jika ingin memaksa, Anda dapat melakukan *type casting* agar code di atas dapat di-compile tanpa menimbulkan pesan kesalahan, seperti berikut ini:

```
byte  data1 = 20;
byte  data2 = (byte)(data1 * 5);
```

Sebagai ringkasan, promosi tipe data secara otomatis dalam suatu ekspresi matematik yang dilakukan oleh Java mengikuti pedoman sebagai berikut:

1. Untuk tipe data **byte** dan **short** akan dipromosikan ke tipe data **int**.
2. Jika salah satu operand bertipe data **long,** maka semua tipe data dari operand yang terlibat dalam ekspresi matematik tersebut akan dipromosikan ke tipe data **long**.
3. Jika salah satu operand bertipe data **float,** maka semua tipe data dari operand yang terlibat dalam ekspresi matematik tersebut akan dipromosikan ke tipe data **float**, kecuali bila ada operand yang bertipe data **double**.
4. Jika salah satu operand bertipe data **double,** maka semua tipe data dari operand yang terlibat dalam ekspresi matematik tersebut akan dipromosikan ke tipe data **double**.

Catatan Penting!

Konversi tipe data terkadang dapat mengakibatkan kebingungan dan kesalahan logik yang sulit dicari penyebabnya. Perhatikan contoh code di bawah ini.

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

Jika code di atas dieksekusi, maka isi variabel `hasil` adalah `6.0` dan bukan `6.5`. Ini karena variabel `nilai` bertipe data **`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`. Hasil ini baru kemudian ditampung oleh variabel `hasil` dan mengakibatkan variabel `hasil` bernilai `6.0`.

Untuk menghindari kesalahan seperti ini usahakan selalu menggunakan tipe data pecahan (floating-point, seperti **`double`** dan **`float`**) untuk melakukan operasi yang dapat menghasilkan bilangan pecahan. Selain itu, Anda juga dapat melakukan *casting* ke tipe data floating-point terhadap salah satu operand yang terlibat dalam operasi tersebut, seperti yang ditunjukkan oleh contoh code berikut ini:

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

3.6 Kuis

1. Apa perbedaan antara tipe data primitif dan tipe data referensi?
2. Untuk melatih pengetahuan kita akan tipe data, buatlah program untuk menghitung luas:
   - Segitiga
   - Lingkaran
   - Bujursangkar
   - Empat persegi panjang

   Hati-hati dengan konversi tipe data secara otomatis yang dilakukan oleh Java, terutama konversi dari tipe floating-point ke integer.