

## BAB III

### THE FUNDAMENTALS OF JAVA PROGRAMMING

#### 3.1 Identifier

Identifier atau pengenal adalah suatu nama yang biasa dipakai dalam pemrograman untuk menyatakan nama *variable* atau konstanta, nama *method* atau *function*, nama *class*, *interface* serta hal-hal lain yang dideklarasikan atau didefinisikan oleh pemrogram (*programmer*) [2].

➤ **Ciri-ciri:**

- ✓ *Case sensitive* (perbedaan besar kecil huruf mempengaruhi)
- ✓ Tidak memiliki batasan panjang kata.

➤ **Syarat penamaan sebuah identifier**

- ✓ Tidak boleh menggunakan keyword java
- ✓ Tidak boleh dimula dengan angka.
- ✓ Tidak boleh mengandung karakter khusus kecuali "\_" dan "\$".
- ✓ Tidak boleh mengandung whitespace.

➤ **Contoh penulisan identifier (benar)**

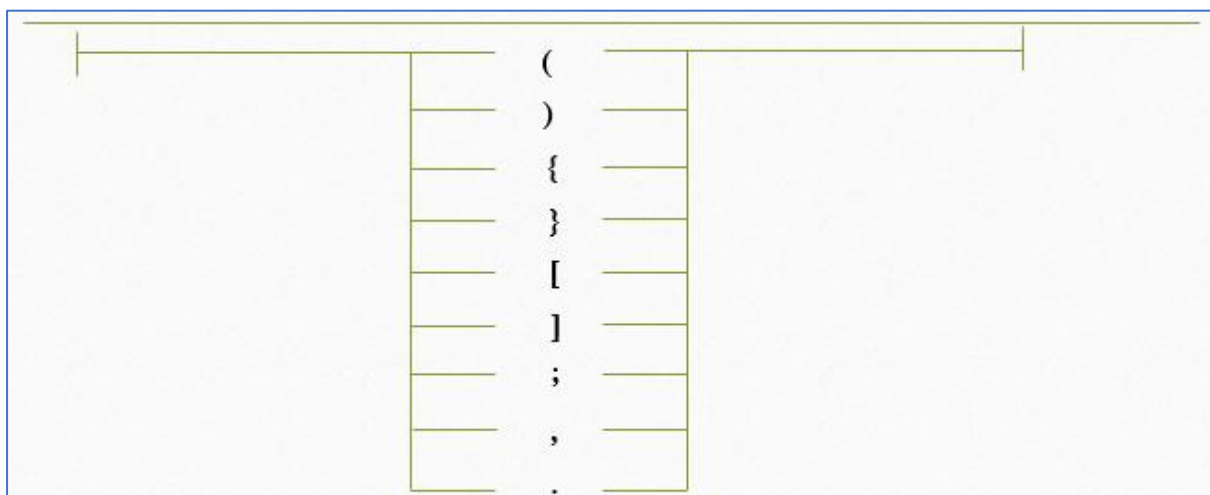
- ✓ **String** namaLengkap = "M. Nishom";
- ✓ **String** nama\_lengkap = "M. Nishom";
- ✓ **String** \$namaLengkap = "M. Nishom";
- ✓ **String** nam4Lengkap = "M. Nishom";

➤ **Contoh penulisan identifier (salah)**

- ✓ **String** nama Lengkap = "M. Nishom"; // penggunaan spasi
- ✓ **String** while = "M. Nishom"; // **while** adalah keyword Java  
// mengandung karakter spesial (selain \_ dan \$)
- ✓ **String** kelasA&KelasB = "";
- ✓ **String** 4alamat = "M. Nishom"; // mengandung angka di awal

#### 3.2 Separator

Separator merupakan salah satu elemen dalam bahasa Java yang digunakan untuk menginformasikan ke kompilator Java mengenai adanya pengelompokan di kode program. Diagram sintaks separator dapat ditunjukkan pada Gambar 3.1:



Gambar 3.1 Diagram Sintaks Separator

Dari semua separator yang ada di dalam diagram sintaks di atas, dapat dikelompokkan berdasarkan fungsinya seperti ditunjukkan pada Tabel 3.1 berikut:

Tabel 3.1 Daftar Separator

No.	Simbol	Nama
1	(...)	Kurung ( <i>parentheses</i> )
2	{...}	Kurung kurawal ( <i>braces</i> )
3	[...]	Kurung siku ( <i>brackets</i> )
4	;	Titik koma ( <i>semicolon</i> )
5	,	Koma ( <i>comma</i> )
6	.	Titik ( <i>period</i> )

### 3.2.1 Simbol Kurung (*parentheses*)

Fungsi tanda kurung dalam pemrograman Java digunakan untuk berbagai kondisi diantaranya sebagai berikut:

- Mendefinisikan daftar parameter yang digunakan dalam **method**.
- Mendefinisikan ekspresi pada operasi tertentu.
- Mendefinisikan ekspresi dalam statement control.
- Melakukan atau mengubah tipe data (*casting*)

- A
- ```
private static double luas_segitiga(double a, double t){
    return 0.5 * a * t;
}
```
- B
- ```
private static double kel_persegi_panjang(double p, double l){
    return 2 * (p + l);
}
```
- C
- ```
private static String ganjil_genap(int angka){
    if(angka%2 == 0){
        return "Genap";
    }else{
        return "Ganjil";
    }
}
```
- D
- ```
private static byte the_byte(int a){
    return (byte) a;
}
```

### 3.2.2 Simbol Kurung Kurawal (*braces*)

Simbol kurung kurawal dalam bahasa pemrograman Java digunakan untuk berbagai hal diantaranya:




- A. Inisialisasi *array* dengan nilai tertentu.
- B. Mendefinikan blok kode untuk kelas, metode, dan lingkup lokal.

- A
- ```
String[] data = {"M. Nishom", "09.017.337", "0619048701"};
```
- B1
- ```
private static byte the_byte(int a){
    return (byte) a;
}
```
- B2
- ```
public class ElemenDasar {
    public static void main(String[] args) {
    }
}
```
- B3
- ```
private static String ganjil_genap(int angka){
    if(angka%2 == 0){
        return "Genap";
    }else{
        return "Ganjil";
    }
}
```

### 3.2.3 Simbol Kurung Siku (brackets)

Simbol kurung siku dalam bahasa pemrograman Java digunakan untuk beberapa kondisi diantaranya adalah sebagai berikut:

- A. Mendeklarasikan tipe *array*
- B. Mendefinisikan nilai *array*
- C. Memanggil nilai *array* berdasarkan indeksNya.

	<pre>String[] brackets = new String[3];</pre>
	<pre>brackets[0] = "M. Nishom"; brackets[1] = "09.017.337"; brackets[2] = "0619048701";</pre>
	<pre>String NIDN = brackets[2]; System.out.println("NIDN: "+NIDN);</pre>

### 3.2.4 Simbol Titik koma (semicolon)

Fungsi simbol "titik koma" digunakan untuk mengakhiri sebuah *statement*, contoh:

```
String[] brackets = new String[3];  
  
int nilai = 45;  
  
String keterangan = "contoh simbol semicolon";  
  
ElemenDasarJava edj = new ElemenDasarJava();  
edj.print();
```

### 3.2.5 Simbol Koma (comma)

Fungsi simbol atau elemen "koma" dalam bahasa pemrograman Java digunakan untuk memisahkan *identifier-identifier*, contoh:

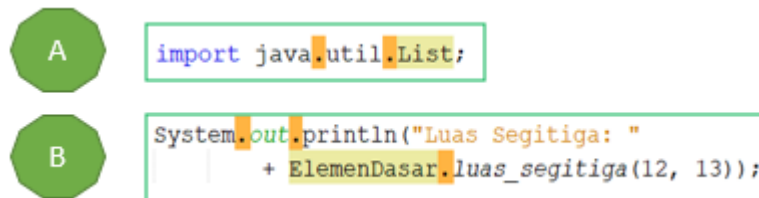
```
private static double luas_segitiga(double a, double t){  
    return 0.5 * a * t;  
}
```

```
int x, y, z;  
int a=0, b=1, c=2;
```

### 3.2.6 Titik (period)

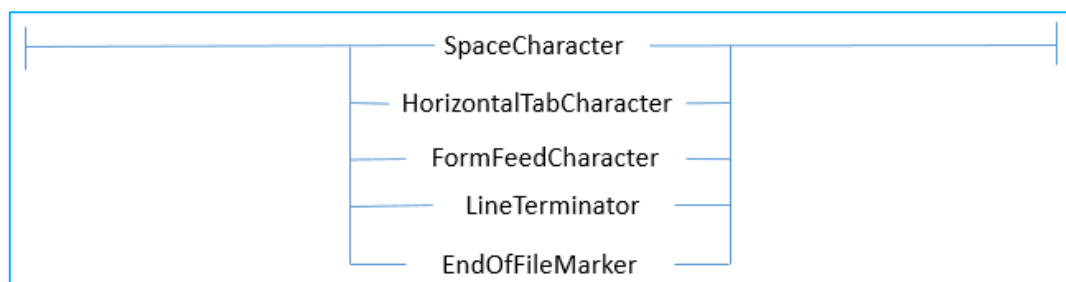
Fungsi elemen “koma” dalam bahasa pemrograman Java digunakan untuk beberapa kondisi, diantaranya:

- A. Memisahkan nama-nama paket dari sub-paket dan kelas.
- B. Memisahkan variabel atau metode dari variabel *reference*.



### 3.3 Whitespace

Whitespace digunakan untuk memisahkan antara token (elemen terkecil di program yang memiliki arti bagi kompilator) dalam program Java. Whitespace terdiri dari spasi, tab dan tanda ubah baris (*linefeed*). Diagram sintaks whitespace dapat digambarkan sebagai berikut:



Gambar 3.2 Diagram Sintaks Whitespace

#### Contoh:

```
String value = "Halo,\r\n\tapa kabar sobat?";
System.out.println(value);
String result = removeAllWhitespace(value);
System.out.println(result);

result = collapseWhitespace(value);
System.out.println(result);
```

#### Output:

```
Output - PemKom2018 (run) x
run:
Halo,
    apa kabar sobat?
Halo,apakabarsobat?
Halo, apa kabar sobat?
BUILD SUCCESSFUL (total time: 0 seconds)
```

### 3.4 Comment

Komentar merupakan elemen dalam Bahasa Java, tetapi tidak menjadi bagian yang akan dieksekusi. Seorang programmer dapat menambahkan komentar sebanyak mungkin dalam sebuah program, tanpa perlu khawatir akan membesarkan ukuran kode program *bytecodes*. Komentar yang digunakan dalam Bahasa Java dapat dilihat pada Tabel 3.2 berikut:

Tabel 3.2 Komentar dalam Bahasa Java

Jenis Komentar	Tipe	Penggunaan
<code>/* Komentar */</code>	Lebih dari satu baris	Semua karakter yang berada di antara simbol <code>/*</code> dan <code>*/</code> akan diabaikan oleh <i>compiler</i> .
<code>// Komentar</code>	Satu baris	Semua karakter setelah simbol <code>//</code> sampai akhir baris akan diabaikan oleh <i>compiler</i> .
<code>/** komentar */</code>	Javadoc/ dokumentasi	Semua karakter di antara <code>/**</code> dan <code>*/</code> adalah karakter yang akan digunakan oleh javadoc untuk menciptakan dokumentasi.

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package elemen;
7
8  /**
9   *
10   * @author nishom
11   */
12  public class Komentar {
13
14      /**
15       *
16       * @param a parameter yang digunakan untuk menampung nilai alas
17       * @param t parameter yang digunakan untuk menampung nilai tinggi
18       * @return luas segitiga
19       */
20      public static double luas_segitiga(double a, double t){
21          return 0.5 * a * t;
22      }
23
24      /**
25       * @param args merupakan argumen baris perintah
26       */
27      public static void main(String[] args) {
28          // ketik kode program anda di sini
29      }
30
31  }
```

Komentar lebih dari satu baris

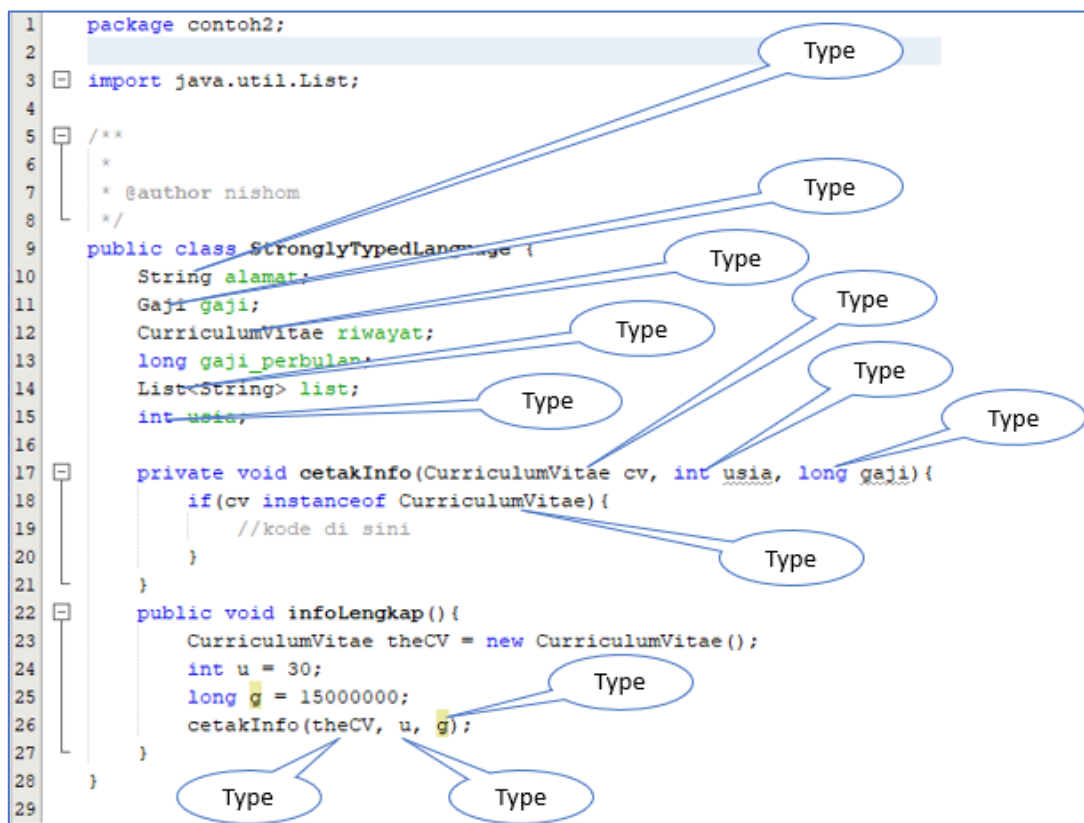
Javadoc

Komentar satu baris

### 3.5 Tipe Data

Tipe Data mendefinisikan cara representasi informasi dan cara informasi direpresentasikan. Tipe data berkaitan erat dengan penyimpanan variabel di memori karena tipe data dari sebuah variabel menentukan cara kompilator menginterpretasikan isian memori. Bahasa java dapat menjadi aman karena termasuk *strongly typed language*, yaitu:

1. Setiap variabel harus mempunyai tipe. Setiap ekspresi harus mempunyai tipe dan setiap tipe harus didefinisikan secara kuat.
2. Semua penugasan, baik eksplisit atau via pelewatan parameter di pemanggilan metode melewati pemeriksaan kompatibilitas tipe.
3. Kompilator java memeriksa semua ekspresi dan parameter untuk menjamin tipe-tipenya kompatibel. Setiap ketidakcocokan tipe adalah kesalahan yang harus dikoreksi pemrogram sampai benar sehingga kode sumber (*source code*) dapat dikompilasi.



Gambar 3.3 Strongly Typed Language

### 3.6 Deklarasi Variabel

Variabel adalah unit dasar penyimpanan di program Java. Variabel didefinisikan menggunakan kombinasi *identifier*, *type*, dan juga sekaligus diinisialisasi. Semua variabel memiliki lingkup yang mendefinikan ketampakan dan waktu hidupnya. Di Java, semua variabel harus dideklarasikan sebelum digunakan untuk menciptakan variabel di memori, kita harus mendeklarasikan dengan menyatakan tipe variabel yang dikehendaki serta *identifier* unik yang mengidentifikasi variabel. Deklarasi variabel memberitahu kompilator untuk menyediakan memori bagi variabel dengan tipe *Type* serta nama variabel *Identifier*. Tanda kurung siku menyatakan kita sekaligus dapat mendeklarasikan banyak variabel dengan tipe yang sama dengan memisahkan variabel-variabel dengan tanda koma. Sintaks deklarasi variabel adalah sebagai berikut:

```
Type Identifier [= value] [, Identifier=[value]...];
```

#### Type:

- ✓ Tipe atomic seperti short, int, long, char, float, double, dan sebagainya
- ✓ Nama kelas
- ✓ Nama interface

#### Contoh penulisan:

```
String fullname = "M. Nishom";  
String author = "M. Nishom", city = "Rembang";  
int height = 161;  
String[] shoes_number = {"39", "40"};
```

### 3.7 Kategori Tipe Data

Tipe data Java dibagi dalam 2 kategori, yaitu:

#### 1. Sederhana

Tipe data sederhana merupakan tipe inti. Tipe sederhana tidak diturunkan dari tipe lain. Tipe ini adalah tipe primitive.

Terdapat 8 tipe primitive di Java, yaitu:

- ✓ 4 tipe adalah untuk bilangan bulat (byte, short, int, long)
- ✓ 2 untuk tipe angka pecahan atau *floating-point* yaitu: float dan double



- ✓ 1 untuk tipe karakter yaitu char untuk karakter dengan pengkodean Unicode, yaitu char. Ada juga yang mengelompokkan char sebagai bilangan bulat.
- ✓ 1 lagi adalah boolean untuk nilai-nilai logika (true/false)

## 2. Komposit

Tipe data komposit disusun dengan tipe sederhana atau tipe komposit lain yang telah ada. Tipe ini antara lain adalah *string*, *array*, *class*, dan *interface*.

### 3.7.1 Tipe Data Integer

Tipe data `integer` merupakan tipe data yang digunakan untuk bilangan bulat. Tipe data ini paling banyak digunakan dalam pemrograman. Untuk bilangan besar, digunakan tipe `long`. Sedangkan tipe `byte` dan `short` biasanya digunakan untuk aplikasi khusus seperti penanganan *file* level rendah atau array besar yang disimpan di tempat kecil.

Contoh penulisan:

```
int nilai = 100;
System.out.println(nilai);
```

Pada Java, *range* tipe bilangan bulat tidak bergantung mesin yang menjalankan kode Java namun telah ditetapkan di spesifikasi Bahasa Java (oleh **sun Microsystems**). Penetapan ini meniadakan upaya yang menyulitkan bagi pemrogram (*programmer*) ketika memindahkan perangkat lunak, dari satu mesin ke mesin yang lain. Spesifikasi tipe data integer dapat dilihat pada Tabel 3.3 di bawah ini.

Tabel 3.3 Spesifikasi Tipe Data Integer

Tipe	Ukuran	Range
byte	8 bit	-128 s/d 127
short	16 bit	-32768 s/d 32768
int	32 bit	-2147483648 s/d 2147483648
long	64 bit	-9223372036854775808 s/d 9223372036854775808
char	16 bit	0 s/d 65535

Tipe **char** dapat dipandang sebagai bilangan bulat yang mengodekan karakter **Unicode** (suatu standar industri yang dirancang untuk mengizinkan teks dan simbol dari semua sistem tulisan di dunia untuk ditampilkan dan dimanipulasi secara konsisten oleh komputer. Dikembangkan secara tandem/berpasang-pasang dengan standar Universal Character Set dan dipublikasikan dalam bentuk buku **The Unicode Standard**).

### 3.7.2 Tipe Data Floating-Point

Bilangan floating-point (titik mengambang), disebut bilangan nyata (*real*) untuk mengevaluasi ekspresi bilangan pecahan. Contoh penggunaannya di perhitungan akar bilangan, *sinus*, *cosinus*, dan sebagainya. Java mengimplementasikan standar *floating-point* IEEE-754. Terdapat 2 tipe *floating-point*, yaitu `float` dan `double`. Masing-masing tipe *floating-point* mempunyai kebutuhan memori berbeda. Tipe `float` memerlukan 32-bit, sebagai *single-precision* dan tipe `double` memerlukan 64-bit sebagai *double-precision*. Spesifikasi tipe data *floating-point* dapat dilihat pada Tabel 3.4 di bawah ini.

Tabel 3.4 Spesifikasi Tipe Data Floating-Point

Tipe	Ukuran	Range	Keterangan
float	32-bits	3.4e-038 s/d 3.4e+038	$3.4 * 10^{-038}$ s.d. $3.4 * 10^{038}$
double	64-bits	1.7e-308 s/d 1.7e+308	$1.7 * 10^{-308}$ s.d. $1.7 * 10^{308}$

Dalam beberapa kondisi terdapat suatu angka dengan pecahan yang cukup banyak, dan biasanya disimbolkan dengan e yaitu eksponensial atau kategori jenis ilmiah untuk menggantikan bagian angka dengan E+n, di mana E mengalikan angka sebelumnya dengan 10 pangkat *n*. Misalnya, format 2-desimal **Ilmiah** menampilkan 5000 sebagai 5.00E+03, yang berarti 5 kali 10 pangkat 3.

Komputasi titik mengambang mengikuti spesifikasi IEEE-754. terdapat 3 nilai titik mengambang special yang digunakan untuk menunjukkan overflow dan kesalahan, yaitu:

1. Tak terhingga positif (positive infinity)
2. Tak terhingga negatif (negative infinity)
3. NaN (not a number)

**Contoh:**

Hasil pembagian angka positif dengan 0 adalah *positive infinity*

Hasil komputasi 0/0 atau hasil akar kuadrat bilangan negative adalah NaN

Di Java, terdapat konstanta untuk tipe float dan double seperti berikut:

- ✓ Double.POSITIVE\_INFINITY
- ✓ Double.NEGATIVE\_INFINITY
- ✓ Double.NaN
- ✓ Float.POSITIVE\_INFINITY
- ✓ Float.NEGATIVE\_INFINITY
- ✓ Float.NaN

### 3.7.3 Tipe Data Boolean

Java memiliki tipe sederhana boolean nilai logis. Variabel tipe ini memiliki salah satu dari dua nilai `true/false` (salah/benar). Tipe ini dihasilkan semua operator relasional seperti `a<b`. Tipe Boolean juga merupakan tipe untuk ekspresi kondisional, menuntun kendali `if` atau loop seperti `for` ataupun `while`.

**Contoh:**

```
public class BooleanExample {
    public static void main(String[] args) {
        boolean b;
        b = false;
        System.out.println("b adalah "+b);
        b = true;
        System.out.println("b adalah "+b);
        //Nilai boolean dapat mengendalikan pernyataan if
        if(b){
            System.out.println("pernyataan ini dieksekusi ");
        }
        b = false;
        if(b){
            System.out.println("Pernyataan ini tidak dieksekusi ");
        }
        //hasil dari operator relational
        System.out.println("10 > 7 adalah "+(10 > 7));
    }
}
```

**Output:**

```
b adalah false
b adalah true
pernyataan ini dieksekusi
10 > 7 adalah true
```

### 3.7.4 Tipe Data Character

Tipe karakter character untuk menyimpan karakter Unicode tunggal. Karena karakter Unicode disusun 16-bit, tipe data char adalah 16-bit unsigned integer. Pertama, tanda petik tunggal sebagai penanda karakter (tanda petik tunggal untuk menandai konstanta/literal char). Contohnya: 'H' adalah satu karakter, sementara "H" adalah string yang berisi satu karakter. Keduanya adalah berbeda. Kedua, Unicode (Java menggunakan Unicode merepresentasikan karakter). Unicode mendefinisikan himpunan karakter internasional secara penuh, dapat merepresentasikan semua karakter di semua negara. Unicode ini merupakan penyatuan seluruh himpunan karakter seperti latin, *Greek*, *Arabic*, *Cyrillic*, *Hebrew*, *Katakan*, *hangul*, dan banyak lagi. Unicode dirancang untuk menangani semua karakter di dunia, di dalam kode 2 byte. Kode 2 byte memungkinkan 65.536 karakter. Himpunan karakter ASCII berada di range 0 sampai 127, dan extended 8 bit disebut ISO-Latin-1(ISO 8859-1) di range 0 sampai 255. Karakter Unicode dikodekan 0 sampai 65535, biasanya diekspresikan sebagai nilai heksadesimal '\u0000' sampai '\uFFFF', dimana '\u0000' sampai '\u00FF' adalah himpunan karakter ASCII atau ISO 8859-1. awalan \u mengindikasikan Unicode dan empat digit heksadesimal memberitahukan karakter Unicode yang dimaksud. Contoh: \u2122 adalah symbol merek (™). Selain karakter escape \u yang mengidentifikasi karakter **Unicode**, terdapat beberapa escape untuk karakter-karakter special seperti dijelaskan pada Tabel 3.5 di bawah ini.

Tabel 3.5 Escape Karakter Spesial

Barisan escape	Nama	Nilai Unicode
\b	Backspace	\u0008
\t	Tab	\u0009
\n	Linefeed	\u000a
\r	Carriage return	\u000d
\"	Petik ganda	\u0022
\'	Petik tunggal	\u0027
\\	backslash	\u005c

### 3.8 Casting

Casting merupakan cara mengubah (konversi) suatu tipe data ke tipe data lain. Casting sering diperlukan ketika sebuah fungsi mengirim tipe yang berbeda dengan tipe yang diperlukan pada sebuah operasi maupun statement. Cara melakukan casting dilakukan dengan cara menempatkan tipe yang diharapkan di dalam tanda kurung buka dan tutup, dan diletakkan sebelah kiri dari nilai atau variable yang akan dikonversi. Sintaks casting:

**(target-type) value**

**Contoh:** casting dari `int` ke `byte`

```
int a_int = 2;
byte b_byte = (byte) a_int;
```

- ▶ Syarat melakukan casting:
  - A. Programmer harus memiliki pengetahuan terkait ukuran penyimpanan.
  - B. Tipe target harus memiliki ukuran penyimpanan yang lebih besar dari pada tipe (tipe asal) yang akan dikonversi.
- ▶ Tipe dalam casting tipe data dapat dilihat pada Tabel 3.6 di bawah ini.

Tabel 3.6 Target Casting Tipe Data

Tipe Asal	Tipe Target
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

### 3.9 Variabel

Variabel adalah unit dasar penyimpanan di program Java. Variabel didefinisikan menggunakan kombinasi *identifier*, *type*, dan juga sekaligus diinisialisasi. Semua

variabel memiliki lingkup yang mendefinikan ketampakan dan waktu hidupnya. Di Java, semua variabel harus dideklarasikan sebelum digunakan untuk menciptakan variabel di memori, kita harus mendeklarasikan dengan menyatakan tipe variabel yang dikehendaki serta *identifier* unik yang mengidentifikasi variabel.

Deklarasi variabel memberitahu kompilator untuk menyediakan memori bagi variabel dengan tipe Type serta nama variabel Identifier. Tanda kurung siku menyatakan kita sekaligus dapat mendeklarasikan banyak variabel dengan tipe yang sama dengan memisahkan variabel-variabel dengan tanda koma.

#### Sintaks deklarasi sebagai berikut:

```
Type Identifier [= value][, Identifier [= value]...];
```

Type dapat berupa salah satu dari beberapa hal berikut:

1. Type atomic seperti short, int, long, char, float, double, dan sebagainya
2. Nama kelas
3. Nama interface, dan lain sebagainya

#### Contoh:

```
int angka;  
int angka = 0;  
int angka = 0, usia = 21;  
int angka = 0, usia = 21, tinggi_badan;  
String nama;  
String Nama = "M. Nishom";  
List<MyClass> list;  
Scanner scanner;
```

### 3.10 Konstanta

Konstanta merupakan sebuah variabel yang nilainya tetap dan tidak bisa diubah lagi. Konstanta digunakan untuk menyimpan data yang tidak akan kita ubah. Dalam Bahasa Java, pendeklarasian konstanta ditulis dengan menambahkan kata kunci (*keyword*) *final* sebelum tipe data dari nama konstanta. Penamaan konstanta adalah dengan menggunakan huruf besar semua (*UPPERCASE*).

#### Contoh:

```
public static final double PI = 3.14159265358979323846;
```

Nilai pada variabel *PI* tidak dapat diubah atau diganti dengan nilai baru, karena sudah bersifat final. Sebenarnya, *keyword* final ini juga dapat diimplementasikan ke semua `Type` dalam bahasa Java. Seperti kelas `Math` yang berisi metode untuk melakukan operasi numerik dasar seperti fungsi eksponensial dasar, logaritma, akar kuadrat, dan trigonometri.

**Contoh:**

```
public final class Math {  
    //konstanta  
    private Math() { //konstruktor  
    }  
    //methods lainnya  
}
```

Kelas `Math` ini merupakan kelas yang bersifat final, sehingga tidak dapat diinstansiasi.

### 3.11 Literal

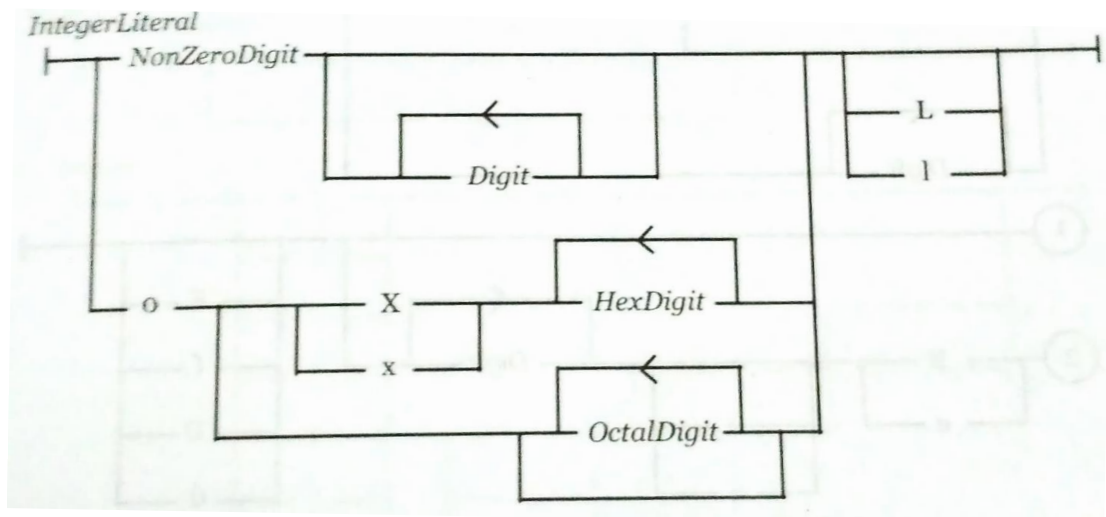
Literal adalah setiap nilai konstan yang dapat ditugaskan ke variabel. Literal juga didefinisikan sebagai representasi sintaksis (*syntactic*) dari data boolean, karakter, numerik, dan string. Literal memberikan sarana untuk mengekspresikan nilai spesifik dalam program. Contoh:

```
int count = 0;
```

dalam pernyataan tersebut, sebuah variabel bilangan bulat bernama `count` dinyatakan dan diberi nilai integer. Literal 0 merepresentasikan nilai. Berikut merupakan representasi sintaks dari berbagai tipe data.

#### 1. Literal Integer

Dalam merepresentasikan nilai pada bilangan bulat dapat dilakukan dengan berbagai cara seperti terlihat pada sintaks di bawah ini.



Gambar 3.4 Sintaks Literal Integer

### Contoh:

Bilangan decimal 77 direpresentasikan:

- 77 dalam decimal
- 0x4D dalam heksadesimal
- 0115 dalam oktal

### Kode Program:

```
public class LiteralInteger {
    int decimal = 77;
    int hexa = 0x4D;
    int oktal = 0115;

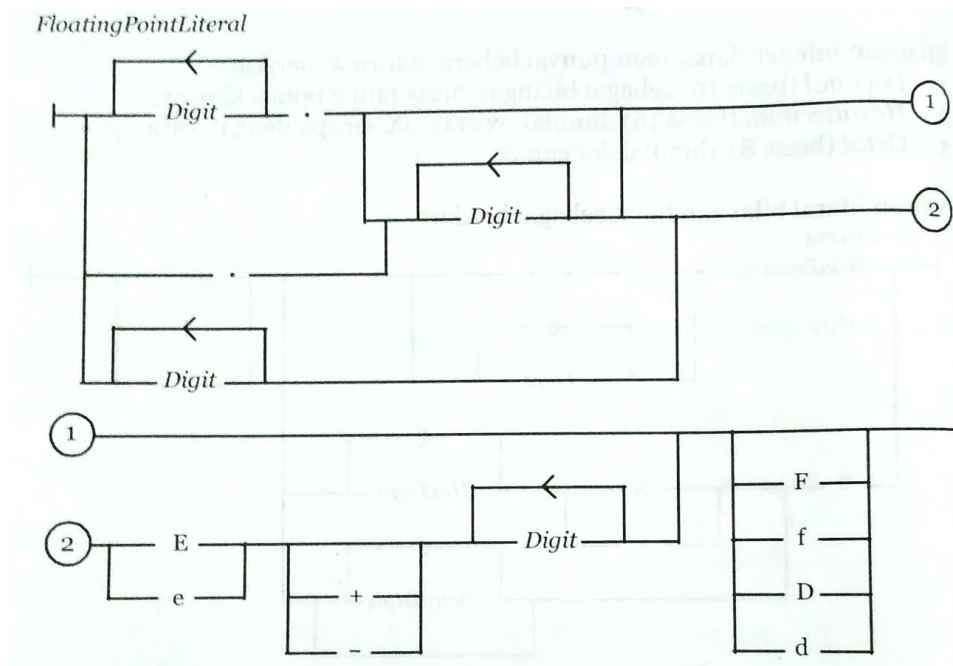
    public LiteralInteger() {
        System.out.println(oktal);
    }

    public static void main(String[] args) {
        LiteralInteger nl = new LiteralInteger();
    }
}
```

## 2. Literal Floating-Point

Dalam merepresentasikan nilai pada bilangan pecahan dapat dilakukan dengan berbagai cara seperti terlihat pada sintaks di bawah ini.





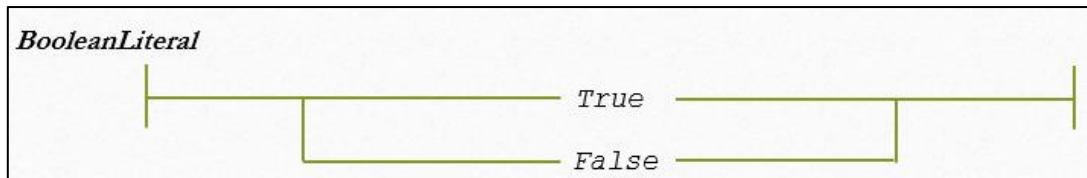
Gambar 3.5 Sintaks Literal Floating-Point

### Contoh Kode Prrogram:

```
public class LiteralFloatingPoint{
    public static void main(String[] args) {
        double decimalNumber = 5.0;
        System.out.println(decimalNumber);
        decimalNumber = 5d;
        System.out.println(decimalNumber);
        decimalNumber = 5D;
        System.out.println(decimalNumber);
        decimalNumber = 0.5;
        System.out.println(decimalNumber);
        decimalNumber = 10f;
        System.out.println(decimalNumber);
        decimalNumber = 10F;
        System.out.println(decimalNumber);
        decimalNumber = 3.14159e0;
        System.out.println(decimalNumber);
        decimalNumber = 2.718281828459045D;
        System.out.println(decimalNumber);
        decimalNumber = 1.0e-6D;
        System.out.println(decimalNumber);
    }
}
```

### 3. Literal Boolean

Java menyediakan tipe boolean dengan nilai: true dan false. Salah satu merepresentasikan salah satu dua kondisi yang mungkin. Sintaks literal Character dapat dilihat pada sintaks di bawah ini.



Gambar 3.6 Sintaks Literal Boolean

- `true` merepresentasikan nilai boolean `true`, contoh: `boolean b = true;`
- `false` merepresentasikan nilai boolean `false`, contoh: `boolean c = false;`

#### Contoh:

```
boolean b = true;
boolean s = false;
```

Representasi nilai pada tipe data `boolean` ini juga dapat dilakukan dengan menggunakan ekspresi tertentu.

#### Contoh:

```
boolean e = 12 > 10;
```

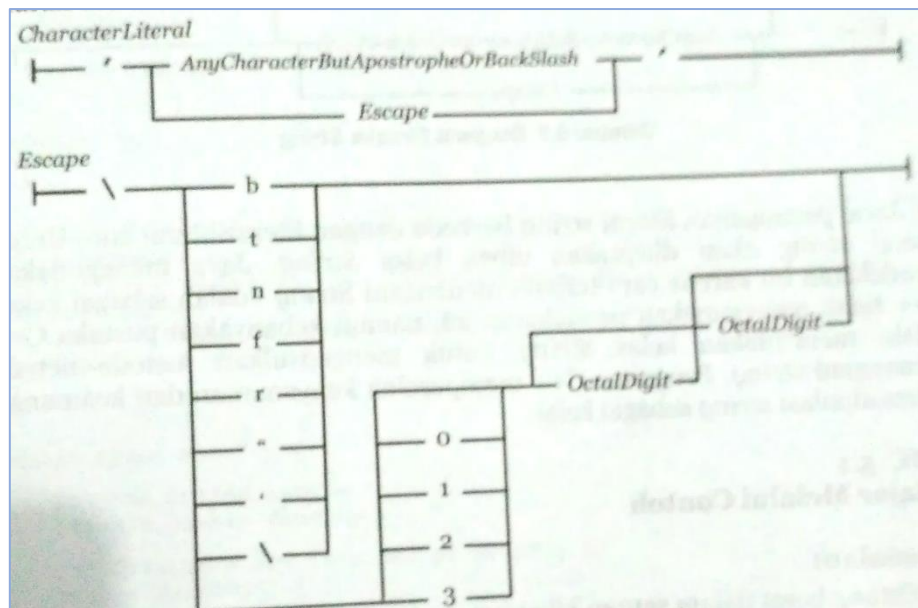
Ekspresi `"12 > 10"` merupakan operasi yang akan menghasilkan nilai `true`.

#### Contoh Kode Program:

```
public class LiteralBoolean {
    public static void main(String[] args) {
        boolean b = true;
        boolean s = false;
        boolean e = 12 > 10;
        System.out.println(e);
    }
}
```

## 4. Literal Character

Berbeda dengan kebanyakan tipe data lainnya pada bahasa Java, tipe data `Character` memiliki beberapa tipe literal unik yang dapat merepresentasikan nilai pada tipe data ini. Misalnya, karakter special `"\n"` adalah nilai untuk perintah ganti baris atau baris baru. Sintaks literal Character dapat dilihat pada Gambar 3.7. sedangkan contoh karakter spesial dapat dilihat pada Tabel 3.7 di bawah ini.



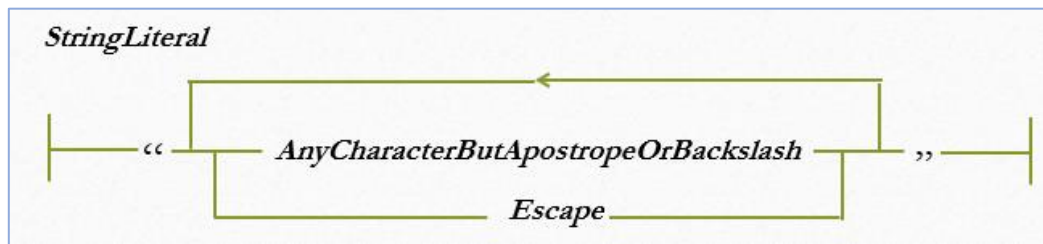
Gambar 3.7 Sintaks Literal Character

Tabel 3.7 Tabel Karakter Spesial

Name	Character	ASCII	hex
Backspace	<code>\b</code>	8	0x08
TAB	<code>\t</code>	9	0x09
NUL character	<code>\0</code>	0	0x00
newline	<code>\n</code>	10	0x0a
carriage control	<code>\r</code>	13	0xd
double quote	<code>\"</code>	34	0x22
single quote	<code>\'</code>	39	0x27
backslash	<code>\\</code>	92	0x5c

## 5. Literal String

Salah satu tipe data yang paling sering digunakan dalam pemrograman Java adalah `String`. Literal dari tipe data `String` merepresentasikan banyak karakter dan nilainya ditulis di dalam pasangan tanda petik ganda. Sintaks literal String dapat dilihat pada Gambar 3.8 di bawah ini.



Gambar 3.8 Sintaks Literal String

#### Contoh:

```
String nama = "M. Nishom";
String detail = "Nama: " + nama;
System.out.println(detail);
```

#### Contoh Kode Program:

```
public class LiteralString {
    public static void main(String[] args) {
        String teks = "Ini adalah contoh literal String \n"
            + "yang mencakup lebih dari satu baris, \n"
            + "yaitu tiga baris teks.\n";

        System.out.println("Deskripsi: " + teks);
    }
}
```

### 3.12 Practice

Soon...

### 3.13 Exercise

Soon...