

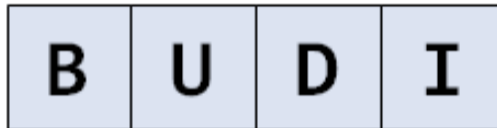
# Bab 3. Input dan Output

## OBJEKTIF :

1. Mahasiswa mampu memahami string, *escape sequence*, mendapatkan *input* dari pengguna, memformat *output*, *style* penulisan program, dan algoritma.
2. Mahasiswa mampu menggunakan *software* IDE NetBeans untuk membuat program dengan menggunakan string, *escape sequence*, mendapatkan *input* dari pengguna, memformat *output*, *style* penulisan program, dan algoritma.

## 3.1 String

Banyak program tidak hanya memproses angka-angka namun juga memproses teks. Program *word processing* dan *search engine* adalah dua contoh program yang memproses teks secara ekstensif. Data berbentuk teks dalam pemrograman disebut sebagai `String`. `String` dalam bahasa Indonesia berarti untai. Data teks disebut string karena teks merupakan untaian karakter-karakter yang berupa huruf-huruf, angka-angka, tanda-tanda baca, spasi, dan sebagainya.



Kita mendeklarasikan sebuah variabel bertipe `String` seperti contoh berikut:

```
String nama;
```

### Catatan:

Tipe data `String` harus dituliskan dengan huruf pertama berupa huruf `S` besar.

Setelah dideklarasikan variabel ini dapat kita tugaskan dengan literal string seperti pada contoh berikut:

```
nama = "Budi Susilo";
```

Pada statement di atas, `"Budi Susilo"` adalah literal string yang kita tugaskan ke variabel `nama`. Literal string diawali dan diakhiri dengan tanda kutip ganda.

Kita juga dapat menyatukan deklarasi dan penugasan nilai awal seperti berikut:

```
String nama = "Budi Susilo";
```

# Konkatenasi String

Operator `+` jika digunakan dengan dua `String` akan mengkonkatenasi (menyambung) kedua `String` tersebut. Sebagai contoh:

```
String namaDepan = "Budi";  
String namaBelakang = "Susilo";  
String namaLengkap = namaDepan + namaBelakang; // namaLengkap = "BudiSusilo"
```

Variabel `namaLengkap` akan menyimpan string `"BudiSusilo"`. Lebih lanjut, jika kita ingin memisahkan nama depan dan nama belakang dengan spasi, kita dapat menuliskan:

```
String namaLengkap = namaDepan + " " + namaBelakang; // namaLengkap = "Budi Susilo"
```

Kita dapat juga mengkonkatenasi sebuah variabel `String` dengan variabel bertipe lain. Nilai dari variabel bertipe lain tersebut akan secara otomatis diubah menjadi string dan disambung ke variabel `String`. Rangkaian statement berikut mencontohkan konkatenasi string dengan tipe data lain:

```
String pekerjaan = "Agen";  
int idPegawai = 7;  
String bond = pekerjaan + idPegawai; // bond = "Agen7"
```

Karena `pekerjaan` adalah sebuah `String` dan `idPegawai` adalah sebuah `integer`, maka nilai `idPegawai` dikonversi ke sebuah `String` (dari `7` dikonversi ke `"7"`). Lalu, kedua string `"Agen"` dan `"7"` dikonkatenasi sehingga menghasilkan `String` `"Agen7"`.

Konkatenasi string sangat berguna untuk mengurangi penulisan statement `System.out.print`. Sebagai contoh, kita dapat mengkombinasi dua statement berikut:

```
System.out.print("Jumlahnya adalah ");  
System.out.println(jumlah);
```

menjadi satu statement berikut:

```
System.out.println("Jumlahnya adalah " + jumlah);
```

Program berikut mendemonstrasikan penggunaan konkatenasi string.

**Program (DemoString.java)**

```

/*
    Program ini mendemonstrasikan konkatenasi string.
*/
public class DemoString
{
    public static void main(String[] args)
    {
        String sapa = "Selamat pagi, ";
        String nama = "Budi Susilo";

        System.out.println(sapa + nama + "!");
    }
}

```

### Ouput Program (DemoString.java)

```
Selamat pagi, Budi Susilo!
```

## Class String

String bukanlah tipe data primitif namun merupakan tipe data *class*. Untuk membuat variabel yang menyimpan sebuah string, kita membuat sebuah *object* dari *class* `String`. *Class* `String` adalah *class* yang telah ditulis sebelumnya oleh pengembang Java dan dapat langsung digunakan dalam program.

Variabel dengan tipe data primitif menyimpan nilai data yang ditugasinya pada alamat memori yang dilabelinya. Misalkan, statement berikut menginisialisasi variabel `number` bertipe `int` dengan nilai 25:

```
int number = 25;
```

Bagaimana variabel ini menyimpan nilai data dapat diilustrasikan pada gambar berikut:

Variabel bertipe primitif menyimpan nilai data sebenarnya pada alamat memori yang dilabelinya.

number = 25

Berbeda dengan variabel dengan tipe data primitif, variabel dengan tipe data *class* tidak menyimpan nilai data pada alamat memori yang dilabelinya namun menyimpan alamat memori dari *object* yang menyimpan datanya. Misalkan, *statement* inisialisasi variabel string berikut:

```
String nama = "Budi Susilo"
```

Ketika *statement* inisialisasi di atas dieksekusi, sebuah *object* dari *class* `String` diciptakan dan *object* `String` tersebut diberikan nilai data `"Budi Susilo"`, lalu alamat memori dari *object* tersebut ditugaskan ke variabel `nama`. Bagaimana variabel bertipe string ini menyimpan alamat memori dari *object* yang menyimpan data string dapat diilustrasikan seperti gambar berikut:

Variabel bertipe *class* (seperti *string*) menyimpan alamat dari *object* yang menyimpan nilai data pada alamat memori yang dilabelinya.



Karena variabel tipe *class* menyimpan alamat dari sebuah *object*, kita tidak menyebut variabel tipe *class* menyimpan suatu nilai data, namun kita menyebut variabel tipe *class* mereferensikan suatu *object* yang menyimpan nilai data. Oleh karena ini, variabel tipe *class* juga dikenal sebagai variabel referensi.

## Method-Method String

Sebuah *class* selain menyimpan data umumnya menyediakan *method-method* untuk melakukan operasi-operasi terhadap data yang disimpannya. Pada *class* `String` terdapat *method-method* yang dapat digunakan untuk bekerja dengan `String` yang disimpannya.

Salah satu *method class* `String` yang sering digunakan adalah *method* `length` yang digunakan untuk mendapatkan panjang string yang disimpannya. Panjang string adalah banyaknya karakter yang terdapat dalam string. Potongan kode berikut mendemonstrasikan cara mendapatkan panjang `String`:

```
String nama = "Budi Susilo";
int panjangString;

panjangString = nama.length(); // panjangString akan menyimpan 11
```

Setelah kode di atas dieksekusi variabel `panjangString` akan menyimpan nilai 11 karena banyaknya karakter dalam string `"Budi Susilo"` adalah 11 karakter. Pada kode di atas, `nama.length()` adalah pemanggilan *method* `length` yang dilakukan pada *object* dari *class* `String` yang direferensikan oleh variabel `nama`.

Secara umum *syntax* pemanggilan *method* dari sebuah *object* adalah seperti berikut:

```
variabelReferensi.namaMethod(argument...)
```

Di mana `variabelReferensi` adalah variabel yang mereferensikan sebuah *object*, `namaMethod` adalah nama *method* yang ingin dipanggil, dan `argument...` adalah nol atau lebih *argument* yang diberikan ke *method*. Jika tidak ada *argument* yang diberikan, seperti pada pemanggilan *method* `length`, kita menuliskan tanda kurung kosong.

Program berikut mendemonstrasikan penggunaan *method* `length`:

### Program (PanjangString.java)

```
/*
    Program ini mendemonstrasikan penggunaan method length
    dari class String.
*/
public class PanjangString
{
```

```

public static void main(String[] args)
{
    String nama = "Budi Susilo";
    int panjangString;

    panjangString = nama.length();
    System.out.println(nama + " mempunyai panjang " + panjangString + "
    karakter.");
}
}

```

### Output Program (PanjangString.java)

```
Budi Susilo mempunyai panjang 11 karakter.
```

Selain *method* `length` terdapat banyak *method-method* class `String` lainnya. Kita tidak akan membahas semua *method-method* dari class `String`, namun kita akan membahas dua *method* class `String` lainnya yang sering digunakan yaitu: *method* `charAt` dan *method* `substring`.

### Method `charAt`

Karakter-karakter yang membentuk sebuah string mempunyai nomor urut yang menandakan posisinya dalam string. Nomor urut ini disebut dengan indeks. Indeks dari string dimulai dari 0, sehingga karakter pertama dari sebuah string mempunyai indeks 0, karakter kedua mempunyai indeks 1, dan seterusnya. Gambar berikut mengilustrasikan sebuah string beserta indeksnya.

"Hello world!"											
	H	e	l	l	o		w	o	r	l	d !
indeks	0	1	2	3	4	5	6	7	8	9	10 11

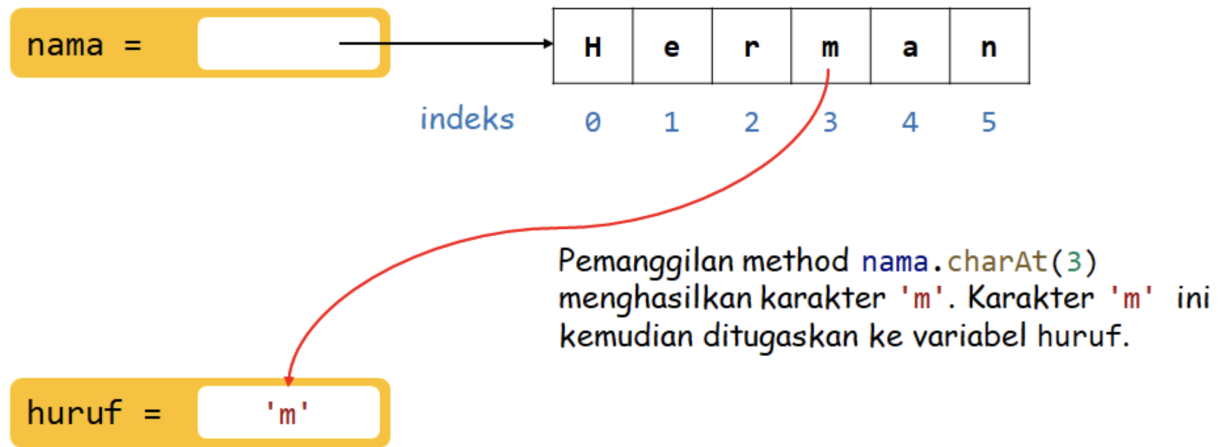
*Method* `charAt` digunakan untuk mencari karakter dalam string berdasarkan indeks. *Method* `charAt` memerlukan sebuah argument berupa integer yang menyatakan indeks dari karakter yang ingin dicari. Sebagai contoh, perhatikan potongan kode berikut:

```

char huruf;
String nama = "Herman";
huruf = nama.charAt(3);

```

Pemanggilan *method* `nama.charAt(3)` mengembalikan karakter `'m'` yang merupakan karakter indeks 3 dari string "Herman" yang direferensikan oleh variabel `nama`. Karakter `m` ini kemudian ditugaskan ke variabel `huruf`. Sehingga, setelah kode ini dieksekusi, variabel `huruf` akan menyimpan karakter `m`. Gambar berikut mengilustrasikan eksekusi dari potongan kode di atas.



## Method `substring`

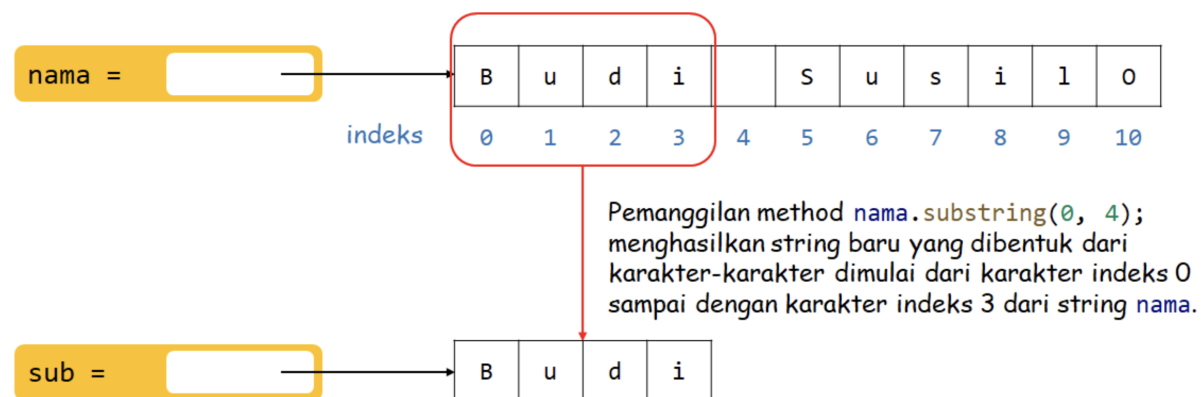
Kita dapat mengambil bagian dari string (substring) dari sebuah string dengan menggunakan *method* `substring`. Pemanggilan *method*:

```
str.substring(awal, batasAkhir)
```

mengembalikan sebuah string yang dibentuk dari karakter-karakter dalam string `str`, dimulai dari karakter dengan indeks `awal` sampai dengan karakter dengan satu indeks sebelum indeks `batasAkhir`. Sebagai contoh, perhatikan potongan kode berikut:

```
String nama = "Budi Susilo";  
String sub = nama.substring(0, 4);    // sub mereferensikan "Budi"
```

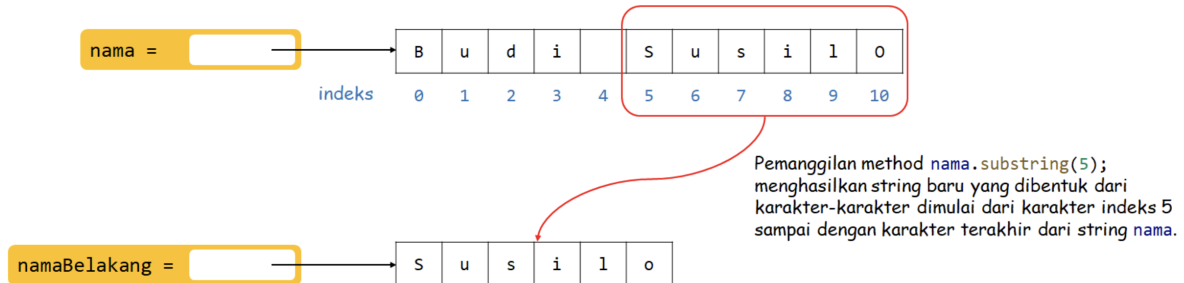
Pemanggilan *method* `nama.substring(0, 4)` mengembalikan string `"Budi"`, yang merupakan sebuah string yang dibentuk dari karakter indeks 0 sampai dengan karakter indeks 3 dari string `"Budi Susilo"`. Sehingga, setelah *statement-statement* di atas dieksekusi, variabel `sub` akan mereferensikan string `"Budi"`.



*Method* `substring` dapat dipanggil dengan satu *argument*. Jika kita hanya memberikan satu *argument* pada pemanggilan *method* `substring`, maka *method* tersebut mengembalikan substring yang dibentuk dari karakter yang dimulai dari indeks sesuai dengan *argument* yang diberikan sampai dengan karakter terakhir. Sebagai contoh, perhatikan potongan kode berikut:

```
String nama = "Budi Susilo";
String namaBelakang = nama.substring(5);
```

Pemanggilan *method* `nama.substring(5)` akan mengembalikan sebuah string yang dibentuk dari karakter indeks 5 sampai dengan karakter terakhir dari string "Budi Susilo". Sehingga, setelah *statement-statement* di atas dieksekusi, variabel `namaBelakang` akan mereferensikan string "Susilo".



## 3.2 Escape Sequence

Kita menggunakan tanda kutip ganda untuk menuliskan literal string. Bagaimana jika kita ingin menuliskan string yang mengandung tanda kutip ganda? Misalkan kita ingin menampilkan *output* berikut ke layar:

```
Dia berkata, "Selamat pagi".
```

*Output* di atas dapat kita dapatkan dengan menuliskan *statement* berikut:

```
system.out.println("Dia berkata, \"Selamat pagi\"");
```

Perhatikan kita menuliskan `\` sebelum kata `Selamat` dan sesudah kata `pagi`. Kita tidak bisa menuliskan tanda kutip ganda langsung `"` di dalam string, karena tanda kutip ganda menandai awal dan akhir dari literal string. Sehingga, kita menggunakan alternatif penulisan tanda kutip ganda. Kombinasi `\` dengan `"` disebut dengan *escape sequence* untuk tanda kutip ganda `"`. *Escape Sequence* adalah kombinasi karakter `\` dengan sebuah karakter tertentu yang digunakan sebagai alternatif penulisan suatu karakter.

Terdapat sejumlah *escape sequence* lainnya selain `\`. Tabel berikut mendaftarkan sejumlah *escape sequence* yang sering digunakan.

<i>Escape Sequence</i>	Keterangan
<code>\"</code>	Digunakan sebagai alternatif penulisan tanda kutip ganda
<code>\'</code>	Digunakan sebagai alternatif penulisan tanda kutip tunggal
<code>\\</code>	Digunakan sebagai alternatif penulisan <i>backslash</i> <code>\</code>
<code>\t</code>	Digunakan untuk mencetak <i>tab</i> (indentasi)

Escape Sequence	Keterangan
\n	Digunakan untuk mencetak baris baru

Escape sequence lainnya yang sering digunakan adalah escape sequence \n untuk mencetak baris baru dan juga escape sequence \t untuk mencetak tab (indentasi). Program berikut mencontohkan penggunaan kedua escape sequence tersebut.

#### Program (EscapeSequence.java)

```

/*
    Program yang mendemonstrasikan escape sequence \t dan \n.
*/
public class EscapeSequence
{
    public static void main(String[] args)
    {
        System.out.print("Berikut adalah penjualan terlaris:\n");
        System.out.print("\tKopi\n");
        System.out.print("\tEs Teh\n");
        System.out.println("\tSoda");
    }
}

```

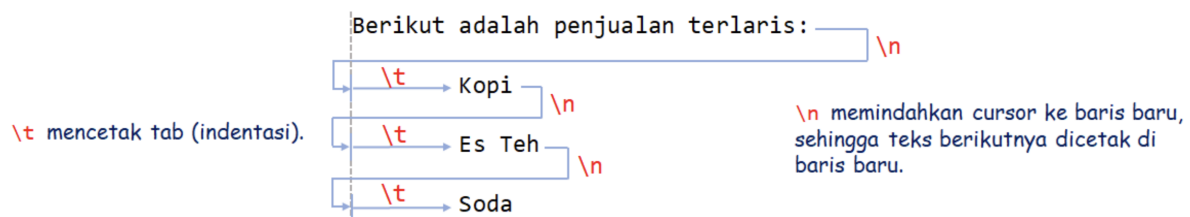
#### Output Program (EscapeSequence.java)

```

Berikut adalah penjualan terlaris:
    Kopi
    Es Teh
    Soda

```

Gambar berikut mengilustrasikan bagaimana escape-escape sequence dari program di atas dicetak.



Jika di dalam string terdapat karakter \ maka compiler akan menganggapnya tanda tersebut adalah bagian dari suatu escape sequence. Bagaimana jika kita ingin mencetak teks yang mengandung tanda \? Misalkan kita ingin mencetak teks berikut C:\temp\test.txt.

Kita tidak dapat menggunakan statement berikut:

```
System.out.println("C:\temp\test.txt"); // mencetak C: emp est.txt
```

\t akan dicetak oleh Java sebagai tab (indentasi). Sehingga, output statement di atas adalah:



```
C:    emp    est.txt
```

Untuk menuliskan `\` dalam sebuah string, kita menggunakan *escape sequence* `\\`, seperti pada *statement* berikut:

```
System.out.println("C:\\temp\\test.txt");
```

String di atas jika dicetak ke layar akan ditampilkan seperti berikut:

```
C:\temp\test.txt
```

### 3.3 Mendapatkan *Input* dari Pengguna

Program-program yang kita tuliskan sebelumnya hanya melakukan komputasi terhadap nilai-nilai yang kita tuliskan di dalam program. Kita dapat membuat program kita menjadi interaktif dengan meminta pengguna program memasukkan *input* data. Misalkan kita membuat sebuah program yang menghitung luas dari bujur sangkar dengan meminta pengguna untuk memasukkan sisi bujur sangkar yang ingin dihitung luasnya. *Output* program yang akan kita buat adalah sebagai berikut:

Masukkan sisi bujur sangkar (cm):     
Luas bujur sangkar adalah 1156 cm<sup>2</sup>.

Pada *output* di atas angka 34 adalah angka-angka yang diketikkan pengguna dan tombol  digunakan untuk mengakhiri *input*. Program ini meminta pengguna memasukkan *input* dengan pertama menampilkan sebuah pesan yang menanyakan pengguna *input* apa yang kita harapkan. Pesan yang meminta pengguna memasukkan *input* ini disebut dengan **prompt**. Teks `Masukkan sisi bujur sangkar (cm):` adalah *prompt*.

Kita menampilkan *prompt* ke pengguna dengan menuliskan *statement*:

```
System.out.print("Masukkan sisi bujur sangkar (cm): ");
```

Kita menggunakan *method* `print` karena kita ingin teks *input* yang diketikkan pengguna tampil di sebelah *prompt* bukan di baris setelah *prompt*.

Untuk membaca *input* dari *keyboard* kita harus menggunakan sebuah *class* bernama `Scanner`. Kita membuat *object* dari *class* `Scanner` dengan menggunakan *statement* berikut:

```
Scanner keyboard = new Scanner(System.in);
```

Kita akan mempelajari arti *statement* di atas nanti saat pembahasan *object* dan *class* pada beberapa bab selanjutnya. Untuk saat ini, kita akan selalu menggunakan *statement* ini saat kita ingin membaca *input* dari *keyboard*.

Setelah mempunyai *object* `Scanner`, kita dapat menggunakan *method* `nextInt` untuk membaca nilai integer yang dimasukkan pengguna dan menugaskannya ke sebuah variabel bertipe `int` dengan *statement* berikut:

```
sisi = keyboard.nextInt();
```

Selanjutnya, kita dapat menuliskan *statement* yang menghitung luas bujur sangkar dan *statement* yang menampilkan luas bujur sangkar tersebut.

Satu hal penting mengenai *class* `Scanner` sebelum kita dapat menggunakannya dalam program kita adalah *class* ini tidak langsung tersedia ke program. Semua program yang menggunakan *class* `Scanner` harus mempunyai *statement* berikut yang dituliskan di bagian awal program sebelum definisi *class*:

```
import java.util.Scanner;
```

*Statement* di atas memberitahukan *compiler* bahwa program kita menggunakan *class* `Scanner` yang terdapat dalam *package* `java.util`. *Package* adalah kumpulan *class-class* yang mempunyai kegunaan sejenis. *Package* `java.util` adalah *package* yang ditulis oleh pengembang bahasa Java dan disertakan dalam JDK.

Berikut adalah kode program lengkap untuk menghitung luas bujur sangkar yang sisinya dari input pengguna:

#### **Program (LuasBujurSangkar.java)**

```
import java.util.Scanner;

/*
    Program ini menghitung luas bujur sangkar dengan
    sisi yang diinput pengguna.
*/
public class LuasBujurSangkar
{
    public static void main(String[] args)
    {
        int sisi;    // Untuk menyimpan input sisi bujur sangkar dari pengguna
        int luas;    // Luas bujur sangkar

        // Buat sebuah object Scanner untuk membaca input keyboard
        Scanner keyboard = new Scanner(System.in);

        // Tampilkan prompt ke pengguna untuk memasukkan input sisi
        System.out.print("Masukkan sisi (cm): ");

        // Baca input berupa integer dari pengguna dan simpan ke sisi
        sisi = keyboard.nextInt();

        // Hitung luas bujur sangkar dengan sisi yang dimasukkan pengguna
        luas = sisi * sisi;
```

```
// Tampilkan luas bujur sangkar
System.out.println("Luas bujur sangkar adalah " + luas + " cm2.");
}
}
```

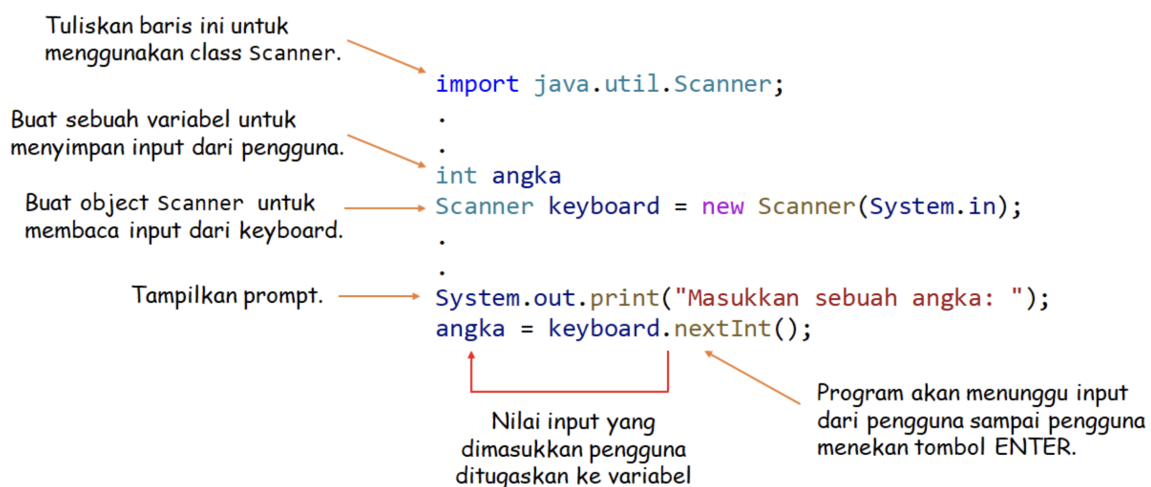
Program di atas pertama akan menampilkan *prompt*:

Masukkan sisi (cm):

Lalu program menunggu *input* dari pengguna. Setelah pengguna memasukkan *input keyboard* dan mengakhiri *input* dengan menekan tombol `Enter`, nilai yang dimasukkan pengguna akan ditugaskan ke variabel `sisi`. Selanjutnya, program menghitung `luas` dengan mengalikan `sisi` dengan `sisi` dan menampilkan teks dari hasil luas. Jika pengguna memasukkan nilai `34` maka *output* program akan seperti berikut:

Masukkan sisi (cm):    
Luas bujur sangkar adalah 1156 cm2.

Gambar berikut mengilustrasikan *statement-statement* yang diperlukan untuk mendapatkan *input* dari pengguna.



## Method-method dalam Class Scanner

Selain *method* `nextInt` yang digunakan untuk membaca *input* berupa integer dari pengguna, *class* `Scanner` memiliki sejumlah *method-method* lain untuk mendapatkan data bertipe selain integer. Tabel berikut mendaftarkan *method-method* dari *class* `Scanner`.

Method	Keterangan
<code>nextByte</code>	Membaca <i>input</i> dalam tipe <code>byte</code>
<code>nextDouble</code>	Membaca <i>input</i> dalam tipe <code>double</code>
<code>nextFloat</code>	Membaca <i>input</i> dalam tipe <code>float</code>

Method	Keterangan
<code>nextInt</code>	Membaca <i>input</i> dalam tipe <code>int</code>
<code>nextLine</code>	Membaca <i>input</i> dalam tipe <code>String</code>
<code>nextLong</code>	Membaca <i>input</i> dalam tipe <code>long</code>
<code>nextShort</code>	Membaca <i>input</i> dalam tipe <code>short</code>

Program berikut mendemonstrasikan penggunaan sejumlah *method* `Scanner` dalam sebuah program.

**Program (Biodata.java)**

```
import java.util.Scanner;

/*
    Program ini meminta pengguna memasukkan biodata.
*/
public class Biodata
{
    public static void main(String[] args)
    {
        String nama;          // Untuk menyimpan nama
        int umur;              // Untuk menyimpan umur
        double beratBadan;    // Untuk menyimpan berat badan
        double tinggiBadan;   // Untuk menyimpan tinggi badan

        // Buat sebuah object Scanner untuk membaca input.
        Scanner keyboard = new Scanner(System.in);

        // Minta nama pengguna
        System.out.print("Masukkan nama Anda: ");
        nama = keyboard.nextLine();

        // Minta umur pengguna
        System.out.print("Masukkan umur Anda: ");
        umur = keyboard.nextInt();

        // Minta berat badan pengguna
        System.out.print("Masukkan berat badan Anda: ");
        beratBadan = keyboard.nextDouble();

        // Minta tinggi badan pengguna
        System.out.print("Masukkan tinggi badan Anda: ");
        tinggiBadan = keyboard.nextDouble();

        // Tampilkan biodata pengguna
        System.out.print("Biodata Anda:\n");
        System.out.println("Nama: " + nama);
        System.out.println("Umur: " + umur + " tahun");
    }
}
```

```

        System.out.println("Berat: " + beratBadan + " kg");
        System.out.println("Tinggi: " + tinggiBadan + " cm");
    }
}

```

### Output Program (Biodata.java)

```

Masukkan nama Anda: Budi Susilo
Masukkan umur Anda: 23
Masukkan berat badan Anda: 75.7
Masukkan tinggi badan Anda: 172.5
Biodata Anda:
Nama: Budi Susilo
Umur: 23 tahun
Berat: 75.7 kg
Tinggi: 172.5 cm

```

## Membaca *Input String* setelah *Input Tipe Numerik*

Method `nextLine` yang membaca *input* string tidak dapat langsung digunakan setelah program membaca *input* dari tipe data berbentuk angka. Sebagai contoh, perhatikan program berikut:

### Program (PermasalahanInput.java)

```

import java.util.Scanner;

/*
    Program berikut mendemonstrasikan permasalahan input
    ketika kita membaca input string setelah input tipe data lain.
*/
public class PermasalahanInput
{
    public static void main(String[] args)
    {
        String nama;    // Untuk menyimpan nama pengguna
        int umur;        // Untuk menyimpan umur pengguna

        // Buat sebuah object Scanner untuk membaca input.
        Scanner keyboard = new Scanner(System.in);

        // Minta umur pengguna
        System.out.print("Berapa umur Anda: ");
        umur = keyboard.nextInt();

        // Minta nama pengguna
        System.out.print("Siapa nama Anda: ");
        nama = keyboard.nextLine();

        // Tampilkan informasi yang dimasukkan ke pengguna
        System.out.println("Halo, " + nama + ". Umur Anda adalah " +
            umur + " tahun.");
    }
}

```

```
}  
}
```

### Output Program (PermasalahanInput.java)

```
Berapa umur Anda: 23  
Siapa nama Anda: Halo, . Umur Anda adalah 23 tahun.
```

Perhatikan pada *output*, program menampilkan *prompt* untuk memasukkan umur dan pengguna dapat memasukkan angka umurnya, namun ketika program meminta pengguna memasukkan nama, program tidak menunggu *input* dari pengguna sehingga pengguna tidak dapat memasukkan namanya. Ini terjadi karena ketika pengguna mengakhiri *input* untuk `umur` dengan menekan `Enter` sebenarnya program membaca *input* tombol `Enter` tersebut sebagai karakter baris baru. Karakter baris baru ini kemudian yang dibaca oleh *method* `nextLine` (baris 24) sebagai karakter yang mengakhiri *input*. Oleh karena itu, program tidak menunggu *input* dari pengguna saat mengeksekusi *statement* baris 24.

Program berikut memperbaiki contoh program di atas:

### Program (SolusiPermasalahanInput.java)

```
import java.util.Scanner;  
  
/*  
    Program berikut mendemonstrasikan solusi dari permasalahan input  
    ketika kita membaca input string setelah input tipe data lain.  
*/  
public class SolusiPermasalahanInput  
{  
    public static void main(String[] args)  
    {  
        String nama;    // Untuk menyimpan nama pengguna  
        int umur;        // Untuk menyimpan umur pengguna  
  
        // Buat sebuah object Scanner untuk membaca input.  
        Scanner keyboard = new Scanner(System.in);  
  
        // Minta umur pengguna  
        System.out.print("Berapa umur Anda: ");  
        umur = keyboard.nextInt();  
  
        // Baca karakter baris baru dan abaikan  
        keyboard.nextLine();  
  
        // Minta nama pengguna  
        System.out.print("Siapa nama Anda: ");  
        nama = keyboard.nextLine();  
  
        // Tampilkan informasi yang dimasukkan ke pengguna  
        System.out.println("Halo, " + nama + ". Umur Anda adalah " +  
            umur + " tahun.");  
    }  
}
```

```
}  
}
```

### Output Program (SolusiPermasalahanInput.java)

```
Berapa umur Anda: 23  
Siapa nama Anda: Budi Susilo  
Halo, Budi Susilo. Umur Anda adalah 23 tahun.
```

Dapat dilihat pada *output* dari program di atas, pengguna sekarang dapat memasukkan *input* namanya. Perbedaan program `SolusiPermasalahanInput.java` dan `PermasalahanInput.java` adalah tambahan *statement* pemanggilan *method* `keyboard.nextLine()` pada baris 23. Pemanggilan *method* ini membaca karakter baris baru dari tombol `Enter` yang diketik pengguna namun karena kita tidak membutuhkannya kita tidak menugaskannya ke variabel apapun. Tujuan dari *statement* pada baris 23 adalah untuk mengabaikan karakter baris baru dari *input* sebelumnya.

## 3.4 Memformat Output

Ketika kita menampilkan angka dengan *method* `System.out.println` atau `System.out.print`, kita tidak dapat mengatur bagaimana format tampilan angka tersebut. Misalkan, nilai dari tipe `double` dapat ditampilkan dengan 15 tempat desimal, seperti yang didemonstrasikan oleh kode berikut:

```
double number = 10.0 / 6.0;  
System.out.println(number);
```

Kode di atas akan menampilkan *output*:

```
1.666666666666667
```

Sering kali kita ingin mengatur format tampilan angka. Misalkan, kita ingin tampilan angka *floating point* dengan presisi dua tempat desimal. Kita dapat melakukannya dengan menggunakan *method* `System.out.printf` dengan format *specifier*, seperti yang dicontohkan oleh kode berikut:

```
double number = 10.0 / 6.0;  
System.out.printf("%.2f", number);
```

Kode di atas akan menampilkan *output*:

```
1.67
```

Pada contoh di atas, `%.2f` disebut sebagai format *specifier*. Ketika *method* `System.out.printf` di atas dieksekusi, `%.2f` tidak ditampilkan di layar, namun, nilai dari variabel `number` dengan dua tempat desimal yang akan ditampilkan.

Kita dapat juga menuliskan format *specifier* sebagai bagian dari suatu teks, seperti berikut:

```
double number = 10.0 / 6.0;  
System.out.printf("Hasil pembagian adalah %.2f.\n", number);
```

Kode di atas akan menampilkan *output*:

```
Hasil pembagian adalah 1.67.
```

Lebih lanjut, kita dapat mempunyai lebih dari satu format *specifier*, seperti pada contoh berikut:

```
double temp1 = 30.8, temp2 = 29.84;  
System.out.printf("Temperatur kedua kota adalah %.2f dan %.2f\n", temp1, temp2);
```

*Output* dari kode di atas:

```
Temperatur kedua kota adalah 30.80 dan 29.84
```

Pada contoh di atas, setiap format *specifier* yang kita tuliskan dalam string akan digantikan dengan nilai dari variabel-variabel yang ditulis setelahnya dengan dipisahkan koma secara berurutan. Gambar berikut mengilustrasikan bagaimana kaitan format *specifier* dengan variabel-variabel yang ditulis setelah string.

Diagram illustrating the mapping of the format specifier `%.2f` to the variable `number` in the `System.out.printf` statement. A blue circle with the number 1 is positioned above the `%.2f` specifier, and an orange arrow points from it to the variable `number`.

```
System.out.printf("Hasil pembagian adalah %.2f.\n", number);
```

Diagram illustrating the mapping of two format specifiers `%.2f` and `%.2f` to the variables `temp1` and `temp2` in the `System.out.printf` statement. A blue circle with the number 1 is positioned above the first `%.2f` specifier, and an orange arrow points from it to the variable `temp1`. A blue circle with the number 2 is positioned below the second `%.2f` specifier, and an orange arrow points from it to the variable `temp2`.

```
System.out.printf("Temperatur kedua kota adalah %.2f dan %.2f\n", temp1, temp2);
```

Secara umum penulisan *method* untuk menampilkan nilai yang terformat adalah sebagai berikut:

```
System.out.printf(StringBerformat, DaftarArgument);
```

dimana *StringBerformat* adalah literal string yang mengandung format *specifier* dan *DaftarArgument* adalah *argument-argument* yang ditulis dipisahkan koma. Kita dapat menuliskan berapapun format *specifier* dalam *StringBerformat* selama terdapat jumlah *argument* yang sama dalam *DaftarArgument*.



## Syntax Format Specifier

Format *specifier* yang digunakan untuk menampilkan angka mempunyai *syntax* sebagai berikut:

```
%[flag][lebar][.presisi]konversi
```

Penjelasan bagian-bagian dari *syntax* di atas:

- **%** -- semua format *specifier* dimulai dengan karakter persen **%**.
- **flag** -- Setelah karakter **%**, satu atau lebih *flag* opsional dapat dituliskan. Flag menyebabkan nilai diformat dalam berbagai cara.
- **lebar** -- Setelah *flag*, kita dapat menentukan lebar minimum *field* untuk nilai tersebut.
- **.presisi** -- Jika nilai yang dicetak adalah angka *floating point*, kita dapat membulatkan nilai tersebut ke suatu presisi. Presisi adalah banyaknya angka di tempat desimal.
- **konversi** -- Semua format *specifier* berakhir dengan karakter konversi, seperti **f** untuk *floating point*, atau **d** untuk integer.

## Karakter Konversi

Huruf **f** pada format *specifier* **%.2f** berarti angka yang ingin kita tampilkan adalah *floating point*. Kita dapat menampilkan angka integer dengan menggunakan format *specifier* **%d**. Contoh berikut mendemonstrasikan format *specifier* untuk integer:

```
int kucing = 2;  
int bebek = 5;  
System.out.printf("Saya mempunyai %d kucing dan %d bebek.\n", kucing, bebek);
```

Output dari kode di atas adalah:

```
Saya mempunyai 2 kucing dan 5 bebek.
```

## Menentukan Presisi

Kita dapat mengubah banyaknya tempat desimal untuk angka *floating point* yang tercetak dengan mengubah **.presisi**. Sebagai contoh, format *specifier* **%.3f** menyebabkan nilai *floating point* tercetak dibulatkan ke tiga tempat desimal.

```
double temp = 78.42869;  
System.out.printf("Temperatur: %.3f.\n", temp);
```

Output dari kode di atas:

```
Temperatur: 78.429.
```

Pada contoh di atas, nilai **78.42869** dibulatkan ke tiga tempat desimal sehingga nilai yang ditampilkan adalah **78.429**.

Berikut adalah contoh lain dengan berbagai presisi format *specifier*:

```
double value1 = 123.45678;  
double value2 = 123.45678;  
double value3 = 123.45678;  
System.out.printf("%.1f %.2f %.3f\n", value1, value2, value3);
```

Pada contoh di atas, nilai `value1` akan ditampilkan dengan pembulatan satu tempat desimal, nilai `value2` akan ditampilkan dengan pembulatan dua tempat desimal, dan nilai `value3` akan ditampilkan dengan pembulatan tiga desimal. Sehingga, output dari kode di atas akan seperti berikut:

```
123.5 123.46 123.457
```

Perlu diperhatikan, kita hanya dapat menentukan presisi pada format *specifier* untuk angka *floating point* (`%f`). Jika kita menggunakannya dengan format *specifier* untuk angka integer (`%d`) maka kita akan mendapatkan *error*.

## Menentukan Lebar Minimum *Field*

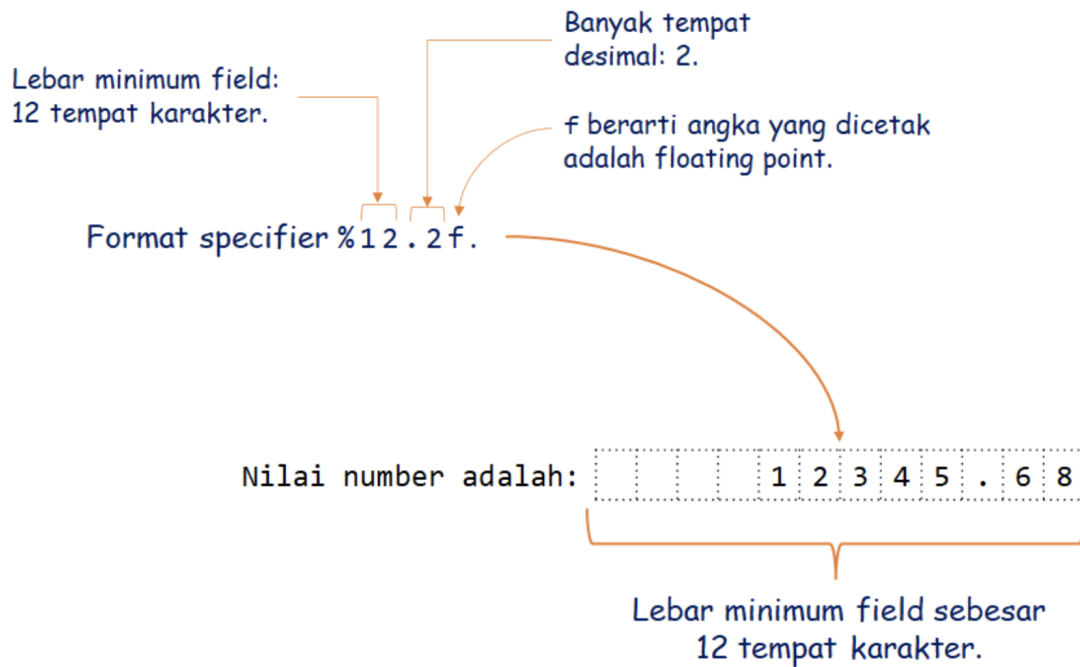
Format *specifier* dapat juga digunakan untuk menentukan lebar minimum *field* dari angka tercetak. Misalkan contoh berikut mencetak angka *floating point* dengan lebar minimum *field* sebanyak 12 tempat karakter:

```
double number = 12345.6789;  
System.out.printf("Nilai number adalah: %12.2f\n", number);
```

mencetak:

```
Nilai number adalah:      12345.68
```

Pada contoh di atas, kita menentukan lebar minimum *field* sebesar 12 tempat karakter. Jumlah karakter dari nilai `number` dengan pembulatan dua tempat desimal termasuk tanda `.` adalah 8 karakter. Karena kita menentukan *field* minimum sebesar 12 karakter, maka angka yang dicetak ditambahkan spasi sebanyak empat kali. Gambar berikut mengilustrasikan lebar minimum *field*.



Menentukan lebar minimum *field* berguna untuk mencetak angka-angka dengan rata kanan dalam sebuah kolom. Program berikut mendemonstrasikan penggunaan minimum *field* untuk mencetak angka-angka rata kanan dalam sebuah kolom.

#### Program (Kolom.java)

```
/*
    Program ini mencetak angka-angka floating point
    dalam sebuah kolom dengan penulisan rata kanan
*/
public class Kolom
{
    public static void main(String[] args)
    {
        // Deklarasi sejumlah variabel double.
        double num1 = 127.889;
        double num2 = 3465.148;
        double num3 = 3.776;
        double num4 = 264.821;
        double num5 = 88.081;
        double num6 = 1799.999;

        // Tampilkan setiap variabel dalam sebuah field
        // dengan 8 tempat dengan 2 tempat desimal.
        System.out.printf("%8.2f\n", num1);
        System.out.printf("%8.2f\n", num2);
        System.out.printf("%8.2f\n", num3);
        System.out.printf("%8.2f\n", num4);
        System.out.printf("%8.2f\n", num5);
        System.out.printf("%8.2f\n", num6);
    }
}
```

### Output Program (Kolom.java)

```
127.89
3465.15
  3.78
264.82
 88.08
1800.00
```

## Mencetak Pemisah Ribuan

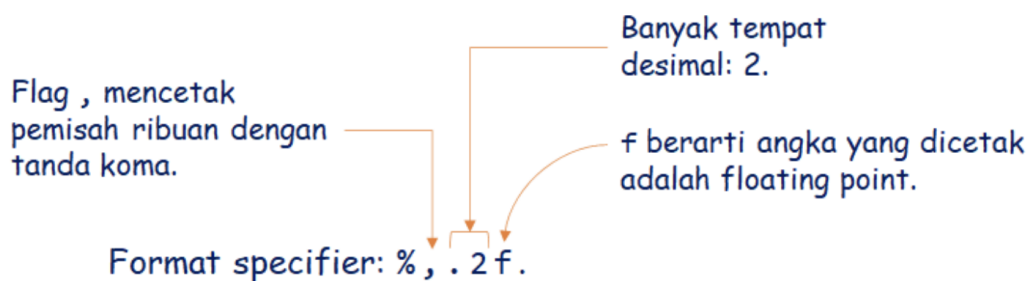
Kita dapat menambahkan menampilkan pemisah ribuan dengan menggunakan *flag* koma (,). Rangkaian *statement* berikut mencontohkan format *specifier* untuk mencetak pemisah ribuan:

```
double jumlah = 1234567.89;
System.out.printf("%,.2f", jumlah);
```

Output dari rangkaian kode di atas:

```
1,234,567.89
```

Gambar berikut menjelaskan bagian-bagian format *specifier* dari contoh di atas.



Mencetak pemisah ribuan umumnya digunakan ketika kita memformat angka untuk format mata uang. Program berikut mendemonstrasikan penggunaan pemisah ribuan untuk memformat mata uang.

### Program (MataUang.java)

```
/*
    Program ini mendemonstrasikan method printf
    untuk memformat mata uang.
*/
public class MataUang
{
    public static void main(String[] args)
    {
        double harga = 450000;
        int banyak = 5;
```

```

double total = harga * banyak;

System.out.printf("Total pembelian: Rp.%,.2f", total);
}
}

```

### Output Program (MataUang.java)

```
Total pembelian: Rp.2,250,000.00
```

## Menambahkan Awalan 0

Flag pada format *specifier* selain mencetak pemisah ribuan juga dapat digunakan untuk mencetak awalan 0 pada angka yang tercetak. Sebagai contoh:

```

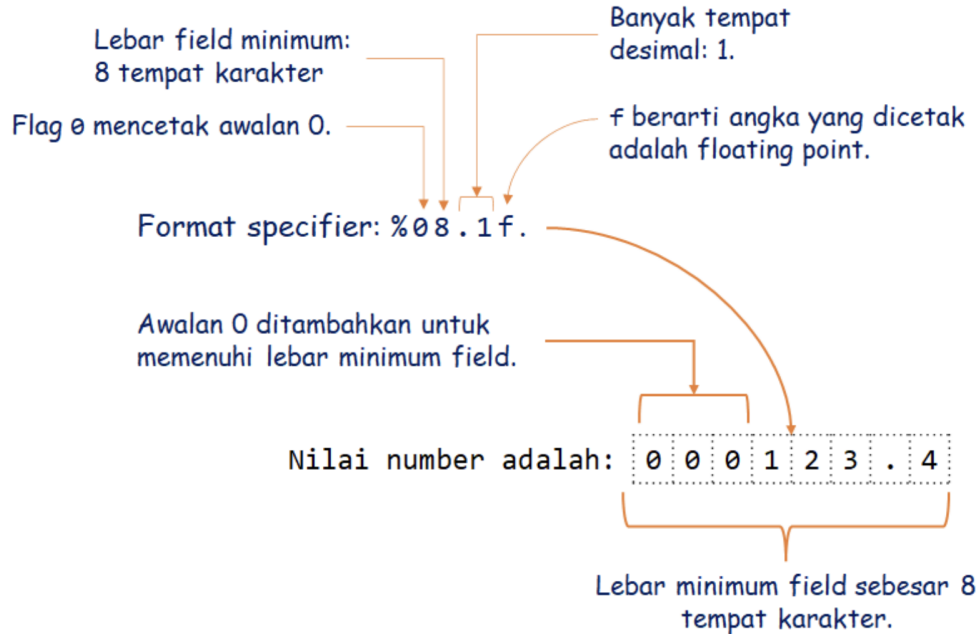
double number = 123.4;
System.out.printf("Nilai number adalah: %08.1f\n", number);

```

akan memberikan *output* berikut:

```
Nilai number adalah: 000123.4
```

Gambar berikut menjelaskan bagian-bagian dari format *specifier* dari contoh di atas.



## Format *Specifier* untuk String

Kita dapat mencetak string dengan `System.out.printf` dengan menggunakan format *specifier* `%s`. Potongan kode berikut mencontohkannya:

```
String nama = "Budi";
System.out.printf("Nama Anda adalah %s\n", nama);
```

Kode di atas akan menghasilkan *output*:

```
Nama Anda adalah Budi
```

Kita dapat juga menambahkan lebar minimum *field* ketika mencetak string. Sebagai contoh, perhatikan kode berikut:

```
String nama1 = "Budi";
String nama2 = "Susilo";
String nama3 = "Herman";
String nama4 = "Donny";
String nama5 = "Wati";
String nama6 = "Wanda";
System.out.printf("%10s%10s\n", nama1, nama2);
System.out.printf("%10s%10s\n", nama3, nama4);
System.out.printf("%10s%10s\n", nama5, nama6);
```

Format *specifier* `%10s` mencetak sebuah string dalam *field* sepuluh karakter. Kode di atas menampilkan nilai-nilai dari variabel dalam sebuah *table* dengan tiga baris dan dua kolom. Setiap kolom mempunyai lebar tiga karakter. Berikut adalah *output* dari kode di atas:

Budi	Susilo
Herman	Donny
Wati	Wanda

Perhatikan pada *output* di atas string-string dicetak rata kanan. Kita dapat menggunakan *flag* `(-)` untuk membuat string-string dicetak rata kiri. Kode berikut mendemonstrasikan ini:

```
String nama1 = "Budi";
String nama2 = "Susilo";
String nama3 = "Herman";
String nama4 = "Donny";
String nama5 = "Wati";
String nama6 = "Wanda";
System.out.printf("%-10s%-10s\n", nama1, nama2);
System.out.printf("%-10s%-10s\n", nama3, nama4);
System.out.printf("%-10s%-10s\n", nama5, nama6);
```

Kode di atas menghasilkan *output* berikut:

Budi	Susilo
Herman	Donny
Wati	Wanda

Kita juga dapat mencetak tipe-tipe data berbeda menggunakan *method* `printf` seperti pada contoh potongan kode berikut:

```
int jamKerja = 40;
double honor = jamKerja * 50000;
String nama = "Budi";
System.out.printf("Nama: %s, Jam Kerja: %d, Honor: Rp.%,.2f\n", nama, jamKerja, honor);
```

Pada potongan kode di atas kita menampilkan sebuah `String`, sebuah `int`, dan sebuah `double`. *Output* dari potongan kode di atas adalah sebagai berikut:

```
Nama: Budi, Jam Kerja: 40, Honor: Rp.2,000,000.00
```

## 3.5 Style Penulisan Program

*Syntax* bahasa Java mengatur urutan penulisan *keyword*, kurung kurawal, titik koma, *identifier*, dan elemen-elemen bahasa Java lainnya. Ketika kita menjalankan *compiler*, *compiler* akan mengecek apakah terdapat *error syntax* dalam program, dan jika tidak ada maka *compiler* akan membuat *bytecode* dari program.

Saat membaca program, *compiler* mengabaikan baris baru, indentasi, spasi-spasi yang memisahkan operator dari *operand*, ataupun spasi-spasi tambahan lainnya. Sebagai contoh kita dapat menuliskan program Java seperti berikut:

### **Program (Ringkas.java)**

```
public class Ringkas {public static void main(String [] args){int
num1=22; double num2=2.5; System.out.println("Nilai num1 adalah "+num1+" dan nilai
num2 adalah "+num2);}}
```

### **Output Program (Ringkas.java)**

```
Nilai num1 adalah 22 dan nilai num2 adalah 2.5
```

Program di atas secara *syntax* benar sehingga dapat dikompilasi dan dijalankan. Namun program tersebut sulit dibaca oleh manusia.

Dalam menulis program, kita biasanya mengikuti *style* penulisan program yang memberikan cara-cara pengaturan penulisan kode sehingga kode dapat mudah dibaca oleh manusia. Program berikut adalah versi dari program `Ringkas.java` yang ditulis mengikuti suatu *style* penulisan program.

### **Program (Terbaca.java)**

```
/*
    Program ini menulis ulang program Ringkas.java
    dengan style pemrograman sehingga kode program
```

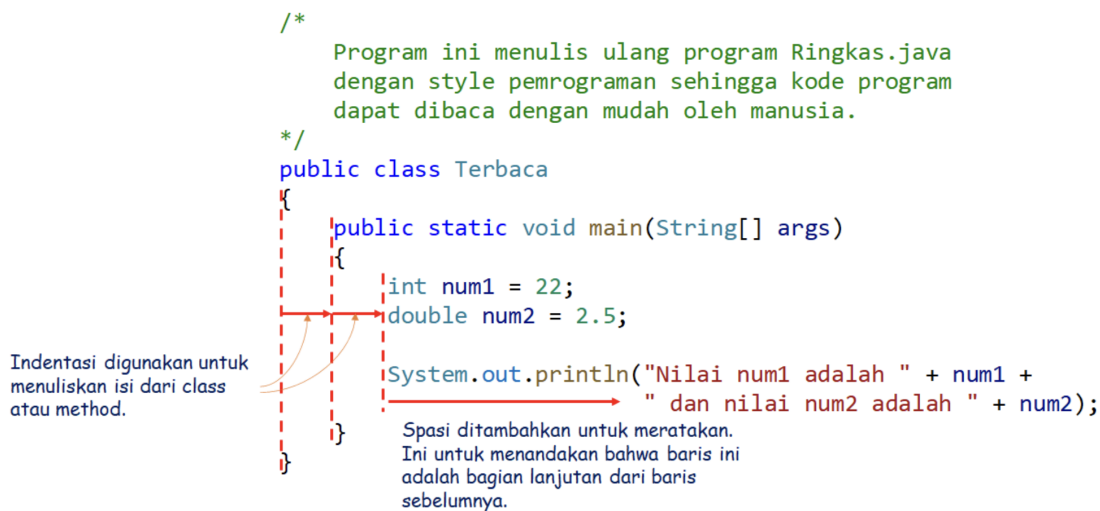
```

    dapat dibaca dengan mudah oleh manusia.
*/
public class Terbaca
{
    public static void main(String[] args)
    {
        int num1 = 22;
        double num2 = 2.5;

        System.out.println("Nilai num1 adalah " + num1 +
                           " dan nilai num2 adalah " + num2);
    }
}

```

Gambar berikut mengilustrasikan penggunaan indentasi dan spasi yang mengikuti suatu *style* penulisan program sehingga program mudah dibaca.



```

/*
    Program ini menulis ulang program Ringkas.java
    dengan style pemrograman sehingga kode program
    dapat dibaca dengan mudah oleh manusia.
*/
public class Terbaca
{
    public static void main(String[] args)
    {
        int num1 = 22;
        double num2 = 2.5;

        System.out.println("Nilai num1 adalah " + num1 +
                           " dan nilai num2 adalah " + num2);
    }
}

```

Indentasi digunakan untuk menuliskan isi dari class atau method.

Spasi ditambahkan untuk meratakan. Ini untuk menandakan bahwa baris ini adalah bagian lanjutan dari baris sebelumnya.

## 3.6 Algoritma

Kita menuliskan program komputer dengan menuliskan suatu prosedur yang berisi urutan langkah-langkah detail instruksi ke komputer. Prosedur yang berisi urutan langkah-langkah detail ini disebut sebagai algoritma.

Kita dapat membayangkan algoritma seperti sebuah resep. Misalkan algoritma untuk membuat teh dapat dituliskan seperti berikut:

1. Masukkan teh celup ke cangkir
2. Isi teko dengan air
3. Masak air dalam teko sampai mendidih
4. Tuangkan air mendidih ke cangkir
5. Tambahkan gula
6. Aduk teh



Terdapat dua alat bantu yang biasanya digunakan untuk menuliskan algoritma: *pseudocode* dan *flowchart*.

## ***Pseudocode***

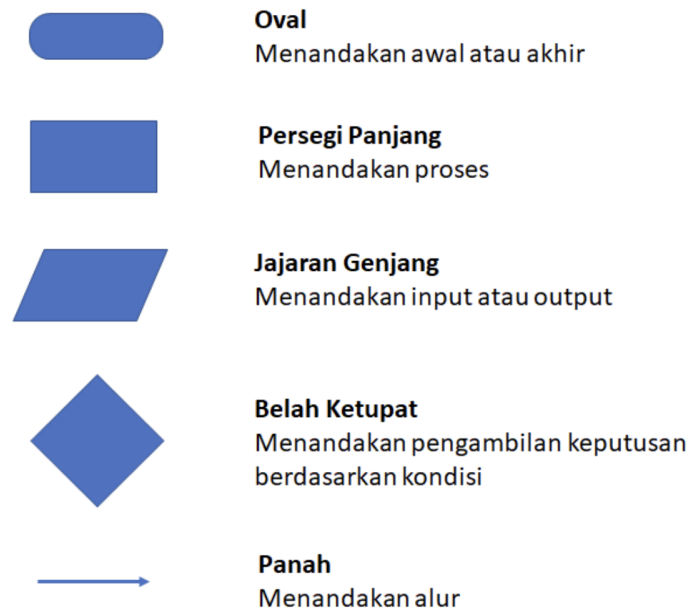
*Pseudo* berarti semu atau tidak nyata. *Pseudocode* (kode tidak nyata) adalah penjelasan langkah-langkah dari suatu algoritma yang menggunakan bahasa informal dan tidak mengikuti *syntax* dari bahasa pemrograman tertentu. Berikut adalah contoh dari *pseudocode* untuk mengonversi temperatur dalam fahrenheit ke celcius:

```
Input temperatur dalam fahrenheit
kalkulasi konversi fahrenheit ke celcius dengan rumus:
    celcius = (5/9) * (fahrenheit - 32)
Tampilkan temperatur dalam celcius
```

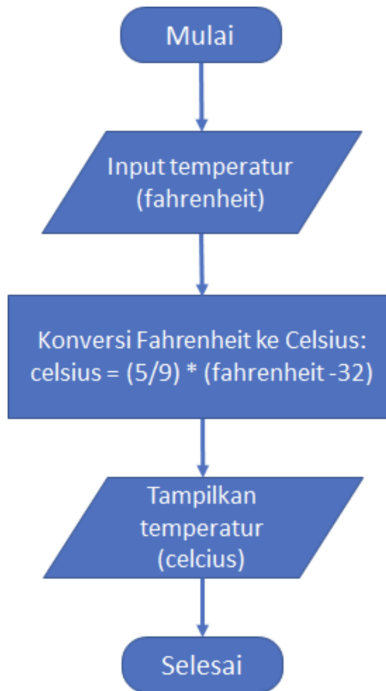
Tidak ada aturan bahasa khusus dalam menulis *pseudocode*. *Pseudocode* dapat dituliskan dalam bahasa apapun selama dapat dibaca dan dimengerti oleh manusia.

## ***Flowchart***

*Flowchart* adalah diagram yang menjelaskan langkah-langkah dari algoritma. *Flowchart* dibentuk dari komponen-komponen dalam gambar berikut:



Flowchart dari program yang mengonversi temperatur dalam fahrenheit ke celcius digambarkan pada gambar berikut:



## REFERENSI

- [1] Horstmann, Cay S. 2012. *Big Java: Late Objects, 1st Edition*. United States of America: John Wiley & Sons, Inc.
- [2] Gaddis, Tony. 2016. *Starting Out with Java: From Control Structures through Objects (6th Edition)*. Boston: Pearson.