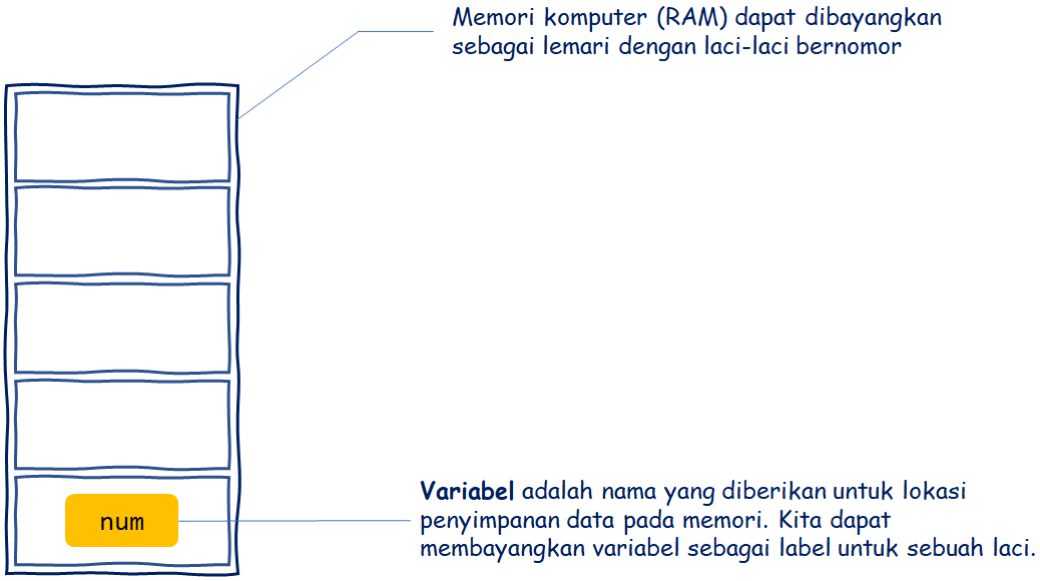
Bab 2. Dasar-dasar Program Java

OBJEKTIF :

1. Mahasiswa mampu memahami variabel, tipe data primitif, aritmatika, *statement assignment*, konstanta, dan konversi antar tipe data.
2. Mahasiswa mampu menggunakan *software* IDE Netbeans untuk membuat program dengan menggunakan variabel, tipe data primitif, aritmatika, *statement assignment*, konstanta, dan konversi antar tipe data.

2.1 Variabel

Variabel adalah nama untuk suatu lokasi penyimpan data dalam memori komputer. Seperti pada pembahasan sebelumnya, memori dapat kita bayangkan sebagai lemari dengan laci-laci bernomor. Kita dapat membayangkan variabel sebagai label untuk nomor laci tersebut. Gambar berikut mengilustrasikan analogi memori sebagai lemari berlaci dan variabel sebagai label untuk suatu lokasi laci.



Variabel memungkinkan kita untuk menyimpan dan melakukan pengolahan data. Program berikut adalah sebuah contoh program Java dengan sebuah variabel.

Program (ContohVariabel.java)

```
// Program ini mendemonstrasikan variabel.

public class ContohVariabel
{
    public static void main(String[] args)
    {
        int num;

        num = 5;
        System.out.print("Nilai variabel num adalah ");
        System.out.println(num);
    }
}
```

Output Program (ContohVariabel.java)

```
Nilai variabel num adalah 5
```

Berikut adalah penjelasan dari masing-masing *statement* dalam *method* `main` pada program `ContohVariabel.java` di atas.

Pada baris 7:

```
int num;
```

statement ini disebut ***statement* deklarasi variabel**. Variabel harus dideklarasikan sebelum dapat digunakan. Deklarasi variabel memberitahu *compiler* bahwa terdapat sebuah variabel dengan nama tertentu yang digunakan untuk menyimpan suatu data dengan tipe tertentu. Kata `int` merupakan singkatan dari *integer* (yang berarti bilangan bulat). Kata `int` memberitahukan *compiler* bahwa tipe data yang akan disimpan dalam variabel adalah angka bulat. Sedangkan kata `num` adalah nama variabel. Sehingga, *statement* deklarasi di atas memberitahukan *compiler* bahwa kita mempunyai sebuah variabel bernama `num` yang akan digunakan untuk menyimpan data bertipe `int` atau *integer* (bilangan bulat).

Pada baris 9:

```
num = 5;
```

statement ini disebut dengan ***statement assignment*** (penugasan). *statement assignment* ini memberitahukan *compiler* bahwa kita menugaskan (memberikan) nilai `5` ke variabel `num`. Sehingga, setelah *statement* ini dieksekusi, variabel `num` akan menyimpan nilai 5. Tanda sama dengan (`=`) adalah operator *assignment* yang menandakan bahwa nilai dari ruas kanannya (dalam contoh ini 5) ditugaskan ke ruas kirinya. Angka `5` pada *statement* ini dalam pemrograman disebut sebagai *literal integer*. ***literal*** adalah nilai yang kita tuliskan pada program.

Pada baris 10:

```
System.out.print("Nilai variabel num adalah ");
```

statement pada baris 10 adalah *statement* pemanggilan *method* `System.out.print` dengan *argument* berupa *literal string* `"Nilai variabel num adalah "`. *statement* ini mencetak: `Nilai variabel num adalah` ke layar.

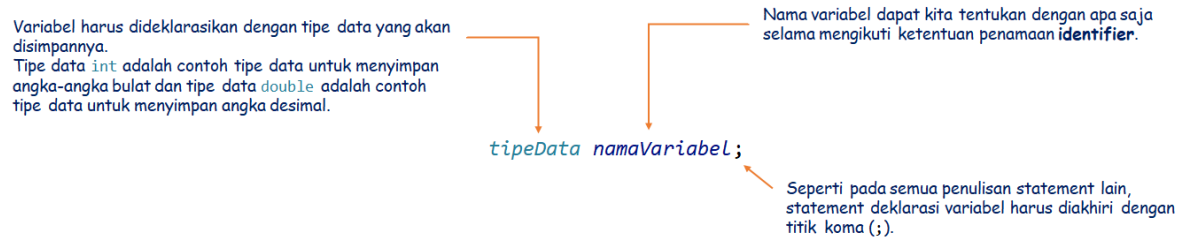
Pada baris 11:

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

Pada baris ini kita memberikan nama variabel sebagai *argument* ke *method* `System.out.println`. Dengan *argument* berupa nama variabel, *method* `System.out.println` (dan juga *method* `System.out.print`) akan mencetak nilai yang disimpan pada variabel tersebut. Pada contoh, `num` menyimpan nilai `5`, sehingga *statement* ini akan mencetak `5` ke layar.

Deklarasi Variabel

Variabel harus dideklarasikan sebelum dapat digunakan. Deklarasi variabel memberitahu *compiler* bahwa kita mempunyai sebuah variabel dengan nama tertentu yang akan digunakan untuk menyimpan tipe (jenis) data tertentu. Kita mendeklarasikan variabel dengan menuliskan *statement* deklarasi variabel. *Syntax* dari *statement* deklarasi variabel dijelaskan pada gambar berikut.



Berikut adalah contoh-contoh *statement* deklarasi variabel.

```
int jumlahHari;
double penjualan;
```

Pada *statement* deklarasi baris pertama di atas, kita mendeklarasikan sebuah variabel bernama `jumlahHari` dengan tipe data `int`. Tipe data `int` digunakan untuk menyimpan *integer* (angka-angka bulat). Pada *statement* deklarasi baris kedua, kita mendeklarasikan variabel bernama `penjualan` dengan tipe data `double`. Tipe data `double` digunakan untuk menyimpan data berupa angka-angka desimal (angka-angka dengan pecahan, seperti: 3.14). Kita akan membahas tipe-tipe data apa saja yang terdapat dalam Java pada bagian selanjutnya.

Nama yang kita tentukan untuk variabel harus mengikuti aturan penamaan *identifier* . Kita akan membahas aturan penamaan *identifier* pada bagian selanjutnya.

Kita dapat mendeklarasikan lebih dari satu variabel yang bertipe sama dalam satu baris. Sebagai contoh, *statement* berikut mendeklarasikan tiga variabel bernama `num1`, `num2`, dan `num3` yang ketiganya bertipe `int` dalam satu baris.

```
int num1, num2, num3;
```

statement di atas ekuivalen dengan *statement-statement* berikut.

```
int num1;
int num2;
int num3;
```

Perhatikan pada penulisan deklarasi tiga variabel dalam satu baris, kita menuliskan tipe data yang diikuti dengan nama-nama variabel dengan setiap masing-masing nama variabelnya dipisahkan koma, lalu kita mengakhirinya dengan titik koma.

Penamaan Variabel

Kita dapat menamakan variabel dengan nama yang kita inginkan selama mengikuti ketentuan penamaan ***identifier***. *identifier* (pengindentifikasi) adalah nama-nama yang programmer tentukan untuk entitas-entitas program seperti variabel, *class* , atau *method*. Pada contoh program di atas, nama *class* `ContohVariabel` dan nama variabel `num` adalah *identifier* .

Salah satu ketentuan penamaan *identifier* adalah nama *identifier* tidak dapat menggunakan *keyword*. *Keyword* adalah kata-kata yang sudah menjadi bagian dari bahasa Java itu sendiri. Kata-kata seperti `public`, `private`, atau `class` adalah contoh-contoh dari *keyword*.

Tabel berikut mendaftar semua *keyword* dalam Java.

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

Ketentuan penamaan *identifier* dalam Java adalah sebagai berikut:

- Tidak menggunakan *keyword*.
- Karakter pertama harus berupa huruf a s.d z atau A s.d Z, *underscore*(`_`), atau tanda dolar (`$`).
- Setelah karakter pertama, kita dapat menggunakan huruf-huruf a s.d z atau A s.d Z, digit-digit 0 s.d 9, *underscore*(`_`), atau tanda dolar(`$`).
- Karakter huruf besar dan huruf kecil membedakan. Ini berarti `barangDipesan` tidak sama dengan `barangdipesan`.
- Tidak mengandung spasi.

Tabel berikut memperlihatkan contoh-contoh nama variabel yang valid atau tidak valid dalam Java.

NamaVariabel	Valid/Tidak Valid	Catatan
jarak_1	Valid	Kita dapat menggunakan *underscore* pada karakter pertama.
x	Valid	Dalam matematika, kita menggunakan variabel dengan nama singkat seperti x atau y. Penggunaan nama-nama singkat ini valid dalam Java, namun tidak umum. *Programmer* umumnya menggunakan nama variabel yang menjelaskan kegunaan variabel tersebut.
6pack	Tidak Valid	Variabel tidak dapat dimulai dengan angka.
juni1997	Valid	Hanya karakter pertama yang tidak boleh berupa angka. Karakter-karakter selanjutnya dapat berupa angka.
nilai ujian	Tidak Valid	Variabel tidak dapat terdiri dari spasi.
double	Tidak Valid	Variabel tidak dapat menggunakan *keyword*.
ltr/botol	Tidak Valid	Variabel tidak dapat terdiri dari simbol-simbol selain *underscore* (`_`) atau tanda dolar (`$`).

Perlu diperhatikan, bahasa Java bersifat *case-sensitive* yang berarti huruf besar dan huruf kecil membedakan. Misalkan, variabel `pencacah` dan `Pencacah` adalah dua variabel berbeda.

Dalam menamakan variabel, sangat dianjurkan untuk memilih nama yang menjelaskan kegunaan variabel tersebut. Misalkan jika kita membuat variabel untuk menyimpan jumlah mahasiswa yang lulus, menamakan variabel tersebut dengan `jumlahKelulusan2020` lebih baik dibandingkan menamakannya dengan `x`. Oleh karena ini, kita akan seringkali menamakan variabel dengan nama yang terdiri dari lebih dari satu kata. Umumnya, dalam menamakan variabel dengan nama lebih dari satu kata, kita menggunakan penulisan yang disebut dengan *camel case*, yaitu huruf pertama dari kata pertama ditulis dengan huruf kecil, dan huruf pertama dari kata-kata selanjutnya ditulis dengan huruf besar. Contoh penulisan dengan *camel case*: `jumlahKelulusan` , `nilaiUjian` , atau `hargaPerUnit`.

Mencetak Nilai Variabel

Kita dapat mencetak nilai yang disimpan oleh variabel ke *output console* dengan memberikan *argument* nama variabel ke *method* `System.out.println` atau `System.out.print`. Sebagai contoh, *statement* berikut.

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

statement diatas akan mencetak nilai yang disimpan oleh variabel `nilaiUjian`. Perhatikan pada *statement* untuk mencetak nilai dari variabel, kita tidak menggunakan tanda kutip ganda di awal dan di akhir nama variabel.

2.2 Tipe Data Primitif

Ketika kita mendeklarasikan sebuah variabel selain menentukan nama untuk variabel tersebut kita juga harus menentukan tipe data yang akan disimpannya. Tipe data menentukan banyaknya ruang memori yang digunakan untuk menyimpan nilai dalam variabel tersebut dan cara data dalam variabel diformat. Di dalam Java terdapat banyak tipe data, namun hanya terdapat delapan tipe data primitif (primitif berarti dasar). Tipe-tipe data lainnya merupakan pengembangan dari tipe-tipe data primitif ini.

Kita mengklasifikasikan tipe-tipe data primitif menjadi empat klasifikasi: tipe-tipe data *integer* yang terdiri dari empat tipe data, tipe-tipe data *floating-point* yang terdiri dari dua tipe data, tipe data `boolean`, dan tipe data `char`.

Tipe Data *Integer*

integer dalam bahasa Indonesia berarti bilangan bulat. Bilangan bulat adalah bilangan tanpa pecahan. Angka -5, 0, 99, dan 10000 adalah contoh-contoh bilangan bulat. Dalam Java, terdapat empat tipe data yang dapat digunakan untuk menyimpan *integer*. Empat tipe data *integer* ini dibedakan berdasarkan ruang penyimpanan memori yang digunakan dan jangkauan dari angka bulat yang dapat disimpan. Tabel berikut mendaftar empat tipe-tipe data *integer*.

Tipe Data	Ukuran	Keterangan
`byte`	1 byte	Digunakan untuk menyimpan *integer* dengan jangkauan -128 s.d $+127$
`short`	2 byte	Digunakan untuk menyimpan *integer* dengan jangkauan $-32,768$ s.d $+32,767$
`int`	4 byte	Digunakan untuk menyimpan *integer* dengan jangkauan $-2,147,483,648$ s.d $+2,147,483,647$
`long`	8 byte	Digunakan untuk menyimpan *integer* dengan jangkauan $-9,223,372,036,854,775,808$ s.d $+9,223,372,036,854,775,807$

Dari keempat tipe data *integer* di atas, tipe data `int` yang biasanya digunakan. Kita menggunakan tipe data `long` ketika kita menulis program yang membutuhkan variabel untuk menyimpan *integer* di luar jangkauan `int`, misalkan program yang menghitung jarak bintang-bintang di tata surya. Kita akan jarang sekali menggunakan tipe data *integer*, `byte` atau `short`. Kedua tipe data tersebut digunakan dulu kala ketika memori komputer masih berkapasitas kecil dan program yang kita tulis harus melakukan efisiensi penggunaan memori.

Literal Integer

Ketika kita memberikan nilai ke variabel kita harus menuliskan *literal-literal* yang sesuai dengan tipe datanya. *Literal* adalah nilai yang kita tuliskan dalam program.

Literal integer ditulis tanpa pemisah ribuan ataupun titik desimal. Penulisan *literal integer* dengan pemisah ribuan seperti berikut akan menghasilkan *error*.

```
int num;
num = 1,257,649;        // ERROR!
```

Rangkaian *statement* di atas harus dituliskan seperti berikut.

```
int num;
num = 1257649;
```

Literal integer dianggap sebagai tipe `int` oleh Java. Untuk menuliskan *literal integer* untuk tipe `long`, kita harus menambahkan akhiran huruf `L` atau `l`, seperti berikut.

```
long bigNum;
bigNum = 1234L;
```

atau

```
long bigNum;
bigNum = 1234l;
```

Akhiran `L` (huruf l besar) yang biasanya digunakan karena huruf `l` terlihat seperti angka `1`.

Kita tetap dapat menggunakan *integer literal* (tanpa akhiran L) untuk memberikan nilai ke variabel tipe `byte` dan `short` selama nilainya masih di dalam jangkauan masing-masing tipe.

Program berikut mendemonstrasikan penggunaan variabel *integer*.

Program (Jumlah.java)

```
/*
    Program ini mendemonstrasikan penggunaan variabel.
*/
public class Jumlah
{
    public static void main(String[] args)
    {
        int num1, num2, jumlah;

        num1 = 45;
        num2 = 98;
        jumlah = num1 + num2;
        System.out.println(jumlah);
    }
}
```

Output Program (Jumlah.java)

```
143
```

Pada baris 5, kita mendeklarasikan tiga variabel bertipe *integer* masing-masing bernama `num1`, `num2`, dan `jumlah`.

Pada baris 7, kita menugaskan nilai `45` ke variabel `num1`.

Pada baris 8, kita menugaskan nilai `98` ke variabel `num2`.

Pada baris 9, kita menugaskan hasil dari penjumlahan nilai yang disimpan `num1`, yaitu `45`, dengan nilai yang disimpan `num2`, yaitu `98`. Tanda `+` adalah operator penjumlahan yang kita gunakan untuk menjumlahkan dua nilai. Setelah *statement* ini dieksekusi, variabel `jumlah` akan menyimpan nilai `143`.

Pada baris 10, kita mencetak nilai yang disimpan oleh variabel `jumlah`. *statement* ini menyebabkan `143` tampil pada layar.

Tipe Data *Floating Point*

Dalam matematika, kita mengenal bilangan desimal. Bilangan desimal adalah bilangan dengan pecahan seperti 10.45, 459.34599, atau 34234234.123123. Dalam komputer, bilangan desimal diistilahkan dengan *floating point* (diterjemahkan ke Bahasa Indonesia menjadi titik mengambang). Dinamakan *floating point* karena titik pemisah pecahan dapat berada di antara digit-digit manapun.

> **Catatan.** Amerika Serikat menggunakan titik sebagai pemisah pecahan dan koma sebagai pemisah ribuan, sedangkan Indonesia menggunakan koma sebagai pemisah pecahan dan titik sebagai pemecah ribuan. Java mengikuti standar Amerika Serikat.

Terdapat dua tipe data *floating point* seperti terlihat pada tabel di bawah.

Tipe Data	Ukuran	Keterangan
`float`	4 byte	Digunakan untuk menyimpan ***floating-point*** dalam jangkauan $\pm 3.4 \times 10^{-38}$ s.d $\pm 3.4 \times 10^{38}$ dengan akurasi hingga 7 digit
`double`	8 byte	Digunakan untuk menyimpan ***floating-point*** dalam jangkauan $\pm 1.7 \times 10^{-308}$ s.d $\pm 1.7 \times 10^{308}$ dengan akurasi hingga 15 digit

Tipe data *floating-point* yang umumnya digunakan dalam program adalah `double`. Sama seperti pada alasan penggunaan tipe data yang lebih kecil dari `int`, tipe data `float` digunakan dahulu ketika kapasitas memori komputer masih terbatas dan program harus mengefisiensi penggunaan ruang memori.

Literal Floating Point

Literal floating point ditulis dengan pemisah desimal titik. Kita tidak dapat menuliskan pecahan ke variabel bertipe `double` atau `float`. Rangkaian *statement* berikut akan menghasilkan *error*.

```
double angka;
angka = 3 1/2;      // ERROR !
```

Rangkaian *statement* di atas harus dituliskan sebagai berikut.

```
double angka;
angka = 3.5;
```

Khusus untuk tipe `float` kita menuliskan *literal*nya dengan akhiran huruf `F` atau `f`. *statement* berikut akan menghasilkan *error*.

```
float jarak;
jarak = 23.5;     // ERROR !
```

Rangkaian *statement* di atas harus dituliskan sebagai berikut.

```
float jarak;
jarak = 23.5F;
```

Kita dapat menuliskan tipe data `double` dengan notasi eksponensial. Misalkan untuk memberikan nilai 1000000.00 ke variabel `double`, kita dapat menuliskan *statement* berikut.

```
total = 1E6;
```

`1E6` berarti 1×10^6.

Program berikut mendemonstrasikan penggunaan tipe data *floating-point*.

Program (Penjualan.java)

```
/*
    Program ini mendemonstrasikan tipe data double.
*/
public class Penjualan
{
    public static void main(String[] args)
    {
        double harga1, harga2, total;

        harga1 = 250000.00;
        harga2 = 125750.25;
        total = harga1 + harga2;
        System.out.print("Total belanja Rp.");
        System.out.println(total);
    }
}
```

Output Program (Penjualan.java)

```
Total belanja Rp.375750.25
```

Tipe Data `boolean`

Tipe data `boolean` adalah tipe data yang hanya dapat bernilai `true` atau `false`. Program berikut mendemonstrasikan deklarasi dan penugasan dari variabel `boolean`.

Program (TrueFalse.java)

```
/*
    Program yang mendemostrasikan variabel boolean
*/
public class TrueFalse
{
    public static void main(String[] args)
    {
        boolean bool;

        bool = true;
        System.out.println(bool);
        bool = false;
        System.out.println(bool);
    }
}
```

Output Program (TrueFalse.java)

```
true
false
```

Variabel dari tipe data `boolean` berguna untuk mengevaluasi kondisi yang bersifat benar atau salah. Kita akan membahas kondisi nanti pada pembahasan struktur keputusan. Untuk saat ini yang perlu kita ketahui mengenai tipe data *boolean*:

- Variabel `boolean` hanya dapat menyimpan nilai `true` atau `false`.
- Isi dari variabel `boolean` tidak dapat disalin ke variabel yang mempunyai tipe data berbeda dengan `boolean`.

Tipe Data `char`

Tipe data `char` digunakan untuk menyimpan karakter. Variabel dengan tipe data `char` dapat menyimpan satu karakter pada satu waktu. *Literal* karakter ditulis dengan mengapitnya dengan tanda kutip tunggal. Berikut adalah contoh pendeklarasian dan pemberian nilai variabel bertipe `char`:

```
char huruf;
huruf = 'A';
```

Pada contoh di atas kita mendeklarasikan variabel `huruf` dengan tipe `char` lalu kita menugaskan variabel tersebut dengan nilai `A`.

Program berikut mencontohkan penggunaan variabel bertipe `char`:

Program (Huruf.java)

```
/*
    Program ini mendemonstrasikan tipe data char.
*/
public class Huruf
{
    public static void main(String[] args)
    {
        char huruf;

        huruf = 'A';
        System.out.println(huruf);

        huruf = 'B';
        System.out.println(huruf);
    }
}
```

Output Program (Huruf.java)

```
A
B
```

Unicode

Karakter secara internal di dalam komputer direpresentasikan dengan angka-angka. Setiap karakter mempunyai representasi angka. Java menggunakan Unicode, yaitu salah satu standard yang digunakan untuk mengkodekan karakter-karakter dalam angka. Dalam Unicode, angka 65 adalah kode untuk huruf `A` (A besar) dan angka 66 adalah kode untuk huruf `B`, dan seterusnya. Untuk daftar lengkap kode-kode untuk semua karakter dalam Unicode dapat dilihat di https://en.wikipedia.org/wiki/List_of_Unicode_characters.

Program berikut mendemonstrasikan Unicode dalam Java.

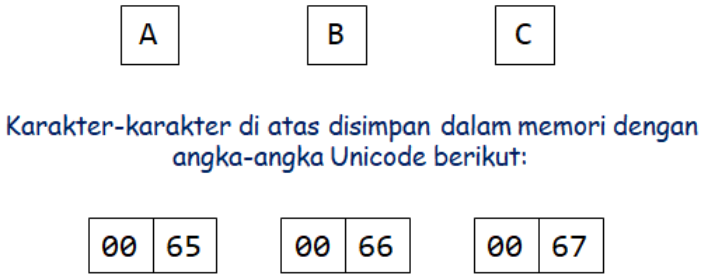
Program (Huruf2.java)

```
/*
    Program ini mendemonstrasikan Unicode
    yang disimpan dalam tipe char.
*/
public class Huruf2
{
    public static void main(String[] args)
    {
        char huruf;

        huruf = 65;
        System.out.println(huruf);

        huruf = 66;
        System.out.println(huruf);
    }
}
```

Output Program (Huruf2.java)

```
A
B
```

Di dalam memori tipe data, `char` disimpan dalam bentuk angka-angka. Diilustrasikan sebagai berikut.

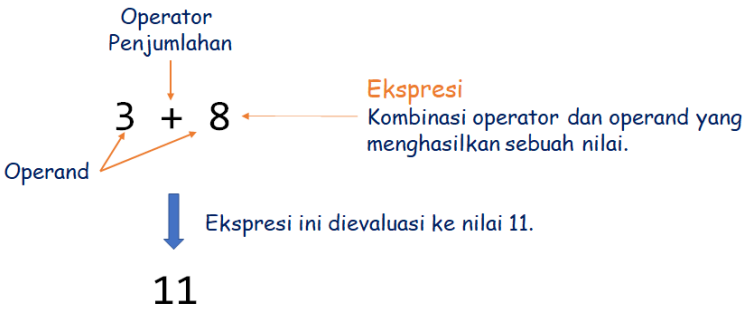


2.3 Aritmatika

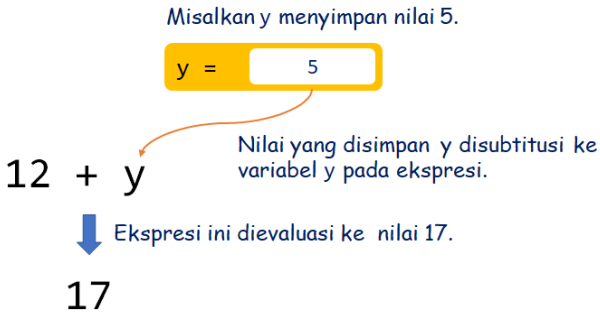
Dalam menulis program, kita akan banyak melakukan operasi-operasi aritmatika seperti penjumlahan, pengurangan, perkalian, atau pembagian terhadap data-data berbentuk angka. Untuk melakukan operasi-operasi aritmatika pada program, kita menuliskan ekspresi.

Ekspresi

Ekspresi adalah kombinasi operator dan *operand* . Operator adalah simbol tertentu untuk menandakan suatu operasi. Sedangkan *operand* adalah nilai yang dikenakan operasi. Sebagai contoh, `3 + 8` adalah sebuah ekspresi dengan operator `+` yang menandakan penjumlahan dan dua *operand* , 3 dan 8. Ekspresi dievaluasi ke sebuah nilai. Misalkan, pada contoh, ekspresi `3 + 8` akan dievaluasi ke nilai `11`.



Selain berupa nilai, *operand* dalam ekspresi dapat juga berupa variabel, seperti `12 + y`. Ekspresi `12 + y` dievaluasi dengan mensubtitusi variabel `y` dengan nilai yang disimpannya. Misalkan, `y` menyimpan nilai 5, maka ekspresi `12 + y` akan dievaluasi ke nilai `17`. Gambar berikut mengilustrasikan proses evaluasi ekspresi dengan *operand* berupa variabel.



Umumnya dalam menulis program, kita menyimpan hasil evaluasi ekspresi ke sebuah variabel. Sehingga, kita umumnya menuliskan *statement assignment* yang ruas kanannya berupa ekspresi, seperti contoh-contoh berikut.

```
jumlah = 3 + 8;    // tugaskan nilai 11 ke jumlah
total = 12 + y;    // tugaskan hasil evaluasi 12 + y ke total
```

Operator Aritmatika

Terdapat lima operator aritmatika yang terdapat dalam Java, seperti terlihat pada tabel berikut.

Operator	Arti	Contoh
`+`	Penjumlahan	`num + 7`
`-`	Pengurangan	`skor - penalti`
`*`	Perkalian	`banyak * harga`
`/`	Pembagian	`total / harga`
`%`	Modulus (Sisa hasil bagi)	`num % 3`

Dari kelima operator di atas, operator penjumlahan, pengurangan, perkalian, dan pembagian tidak terlalu asing bagi kita karena merupakan operator aritmatika yang umum. Operator terakhir yaitu operator modulus, `%`, butuh penjelasan lanjutan. Operator `%` menghitung sisa bagi dari dua nilai. Misalkan: `10 % 3` yang dibaca 10 modulus 3 akan dievaluasi menjadi `1`. Ini karena 10 dibagi 3 menghasilkan 3 dengan sisa 1. Sisa 1 ini yang merupakan hasil dari 10 modulus 3.

Satu hal lain yang perlu diketahui mengenai operator aritmatika adalah operator `-` dapat juga digunakan untuk menuliskan *literal* negatif atau untuk menegasi. Kita menuliskan *literal* negatif seperti berikut: `-8`. Kita juga dapat menegasi nilai yang disimpan variabel, misalkan variabel `num` menyimpan nilai `15`, ekspresi `-num` akan dievaluasi ke `-15`.

Program berikut mendemonstrasikan penggunaan operator-operator aritmatika.

Program (DemoAritmatika.java)

```
/*
    Program ini mendemokan penggunaan operator aritmatika.
*/
public class DemoAritmatika
{
    public static void main(String[] args)
    {
        int op1, op2;                                // Operand1 dan Operand2
        int jumlah, selisih, kali, bagi, modulus; // Untuk menyimpan hasil
operasi aritmatika

        // Tugaskan nilai ke operand1 dan operand2
        op1 = 74;
        op2 = 31;

        // Cetak nilai variabel
        System.out.print("op1 = ");
        System.out.println(op1);
        System.out.print("op2 = ");
        System.out.println(op2);

        // Operasi Penjumlahan
        jumlah = op1 + op2;
        System.out.print("op1 + op2 = ");
        System.out.println(jumlah);
```

```
        // Operasi Pengurangan
        selisih = op1 - op2;
        System.out.print("op1 - op2 = ");
        System.out.println(selisih);

        // Operasi Perkalian
        kali = op1 * op2;
        System.out.print("op1 * op2 = ");
        System.out.println(kali);

        // Operasi Pembagian
        bagi = op1 / op2;
        System.out.print("op1 / op2 = ");
        System.out.println(bagi);

        // Operasi Modulus
        modulus = op1 % op2;
        System.out.print("op1 % op2 = ");
        System.out.println(modulus);

        // Operasi Negasi
        System.out.print("-op1 = ");
        System.out.println(-op1);
    }
}
```

Output Program (DemoAritmatika.java)

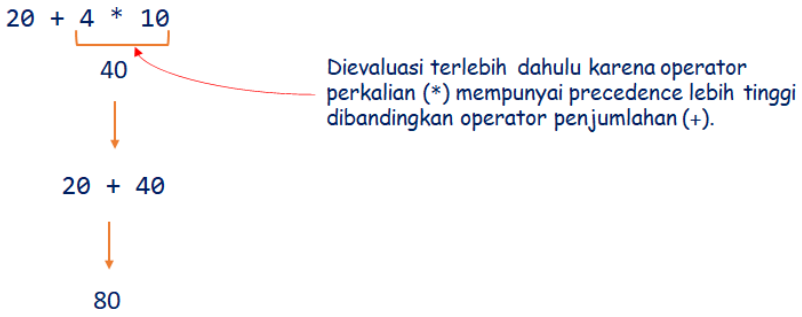
```
op1 = 74
op2 = 31
op1 + op2 = 105
op1 - op2 = 43
op1 * op2 = 2294
op1 / op2 = 2
op1 % op2 = 12
-op1 = -74
```

Operator *Precedence* (Keutamaan Operator)

Kita dapat menuliskan ekspresi dengan lebih dari satu operator. Sebagai contoh.

```
20 + 4 * 10
```

Ekspresi di atas akan dievaluasi ke `60`. Ketika kita mempunyai ekspresi yang terdiri lebih dari satu operator, Java mengevaluasi ekspresi tersebut berdasarkan urutan *precedence* (keutamaan) dari operator-operator tersebut. Operator `*` mempunyai *precedence* lebih tinggi dibandingkan operator `+`, maka operasi perkalian dievaluasi terlebih dahulu sebelum operasi penjumlahan. Sehingga langkah evaluasi ekspresi pada contoh, Ekspresi `4 * 10` dievaluasi ke `40` dan menghasilkan ekspresi `20 + 40` yang dievaluasi ke `60`. Gambar berikut mengiluastrasikan proses evaluasi ekspresi di atas.



Tingkat *precedence* dari operator-operator aritmatika dapat dilihat pada tabel berikut:

Tingkat precedence	Operator
Precendence tertinggi	`*`, `/`, `%`
Precendence terendah	`+` , `-`

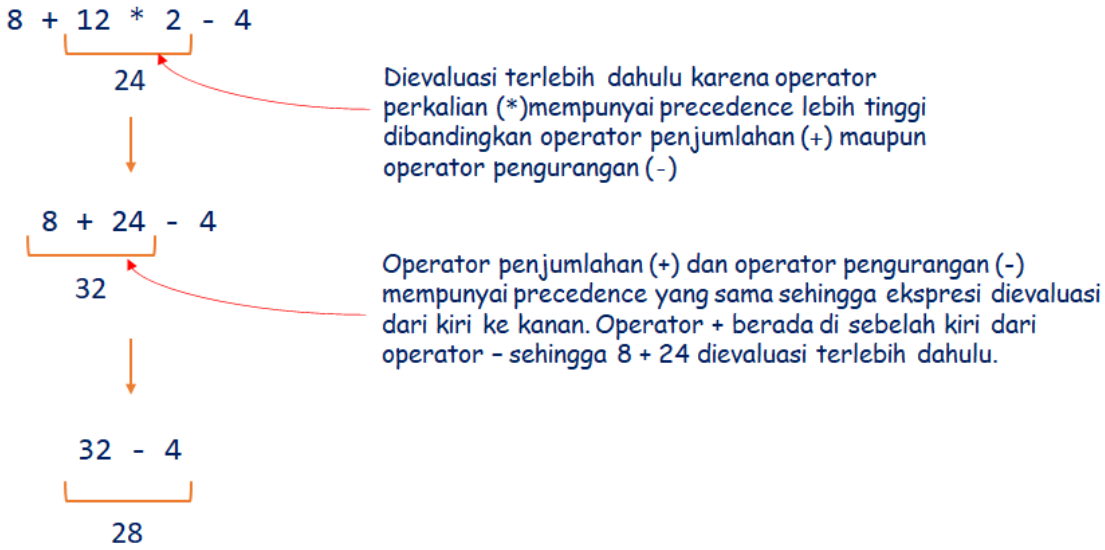
Pada tabel dapat kita lihat, operator perkalian, operator pembagian, dan operator modulus mempunyai tingkat *precedence* yang sama. Operator penjumlahan dan operator pengurangan juga mempunyai tingkat *precedence* yang sama. Jika dalam sebuah ekspresi terdapat dua operator dengan tingkat *precedence* yang sama ekspresi tersebut dievaluasi dari kiri ke kanan.
Misalkan ekspresi,

```
8 + 12 * 2 - 4
```

dievaluasi ke nilai `28` berdasarkan tahapan evaluasi berikut:

1. Pada ekpresi `8 + 12 * 2 - 4`, bagian ekspresi `12 * 2` dievaluasi terlebih dahulu karena operator `*` memiliki *precedence* lebih tinggi dibandingkan operator `+` ataupun `-`, sehingga menghasilkan ekspresi `8 + 24 - 4`.
2. Pada ekspresi `8 + 24 -4`, bagian ekspresi `8 + 24` dievaluasi terlebih dahulu karena operator `+` dan `-` mempunyai *precedence* yang sama, sehingga ekpsresi dievaluasi dari kiri terlebih dahulu. Karena ekspresi `8 + 24` adalah bagian ekspresi paling kiri maka ekspresi ini dievaluasi terlebih dahulu, sehingga menghasilkan ekspresi `32 - 4`.
3. Terakhir, ekspresi `32 - 4` dievaluasi ke `28`.

Ilustrasi tahapan evaluasi ekspresi di atas diilustrasikan pada gambar di bawah:



Tanda kurung untuk Mengutamakan Operator

Kita dapat mengutamakan bagian dari ekspresi dievaluasi terlebih dahulu sebelum bagian lain dengan menuliskan bagian tersebut dalam tanda kurung. Sebagai contoh,

```
(a + b + c + d) / 4.0
```

menyebabkan `(a + b + c + d)` dievaluasi terlebih dahulu lalu hasilnya dibagi dengan `4.0`. Tanpa tanda kurung ekspresi di atas dievaluasi dengan mengevaluasi `d / 4.0` terlebih dahulu (karena operator `/` mempunyai *precedence* lebih tinggi dibandingkan operator `+`) lalu hasilnya ditambahkan ke `a`, `b`, dan `c`. Tabel berikut berisi contoh-contoh penggunaan tanda kurung pada ekspresi dan nilai hasil evaluasi ekspresi tersebut.

Ekspresi	Dievaluasi ke
`(5 + 2) * 4`	28
`10 / (5 -3)`	5
`8 + 12 * (6 - 2)`	56
`(4 + 17) % 2 - 1`	0
`(6 - 3) * (2 + 7) / 3`	9

Class `Math`

Seringkali dalam menulis program kita memerlukan operasi matematika yang lebih kompleks daripada penjumlahan, pengurangan, pembagian, atau perkalian. Misalkan kita memerlukan perhitungan pemangkatan atau pengakaran. Dalam Java tidak ada simbol operator untuk melakukan pemangkatan atau pengakaran. Untuk mengkalkulasi pemangkatan atau pengakaran, kita harus memanggil *method-method* dari *class* `Math`. *class* `Math` dan *method-method*nya telah tersedia dalam Java dan bisa langsung kita gunakan. Sebagai contoh, misalkan untuk menghitung $\sqrt{36}$ kita menuliskan `Math.sqrt(36)` dan untuk menghitung 3^5 kita menuliskan `Math.pow(3, 5)`.

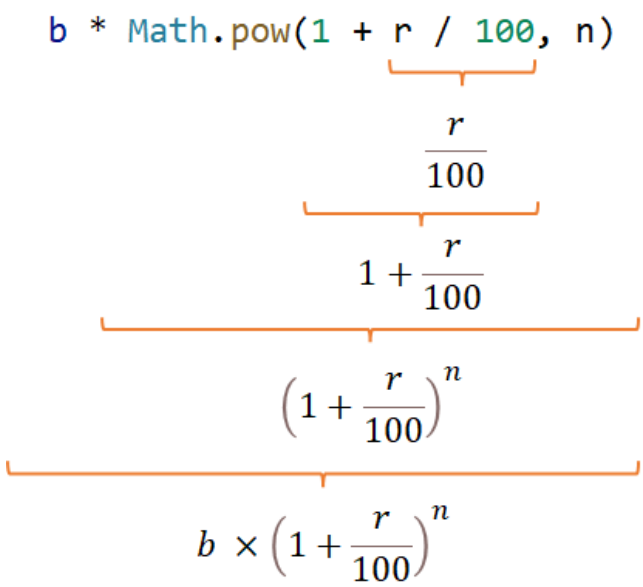
> **Catatan.** `sqrt` adalah singkatan dari *square root* yang berarti akar kuadrat sedangkan `pow` adalah singkatan dari *power* yang berarti pangkat.

Kita dapat mengkombinasi operator aritmatika dan pemanggilan *method* dari *class* `Math` untuk menuliskan ekspresi matematika yang kompleks. Misalkan, ekspresi matematika $b \times \left(1 + \frac{r}{100}\right)^n$

kita tuliskan dengan ekspresi dalam bahasa Java maka akan sebagai berikut.

```
b * Math.pow(1 + r / 100, n)
```

Gambar berikut menunjukkan bagaimana menganalisa ekspresi di atas:



Selain pemangkatan dan pengakaran, *class* `Math` mempunyai *method-method* lain yang beberapa diantaranya dapat dilihat pada tabel di bawah.

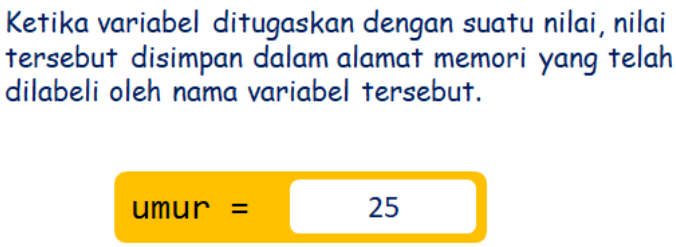
method	**Menghitung**
`Math.sqrt(x)`	Akar kuadrat dari x
`Math.pow(x, y)`	x^y
`Math.sin(x)`	Sinus dari x (x dalam radian)
`Math.cos(x)`	Cosinus dari x (x dalam radian)
`Math.exp(x)`	e^x
`Math.log(x)`	Logaritma natural dari x ($ln(x)$)
`Math.log10(x)`	Logaritma basis 10 dari x ($log_{10}(x)$)
`Math.round(x)`	Pembulatan x ke *integer* terdekat
`Math.abs(x)`	Nilai mutlak dari x ($\|x\|$)
`Math.max(x, y)`	Maksimum dari x dan y
`Math.min(x, y)`	Minimum dari x dan y

2.4 *Statement Assignment*

Kita menuliskan *statement assignment* (penugasan) untuk menugaskan sebuah variabel dengan suatu nilai.

```
umur = 25;
```

Operator `=` disebut sebagai operator *assignment*. Ketika *statement* di atas dieksekusi, komputer menyimpan nilai `25` ke lokasi dalam memori yang telah dilabeli dengan variabel `umur`. Kita dapat mengilustrasikan variabel dan nilai yang disimpan dalam alamat memori yang dilabelinya seperti gambar berikut.



Dalam *statement assignment*, nama variabel yang ditugaskan sebuah nilai harus dituliskan dalam ruas kiri dari operator `=`. *statement* berikut adalah contoh *statement assignment* yang salah.

```
25 = umur;   // ERROR!
```

Selain menuliskan *statement assignment* dengan sebuah *literal*, kita dapat menuliskan *statement assignment* dengan sebuah ekspresi di ruas kanan operator `=`.

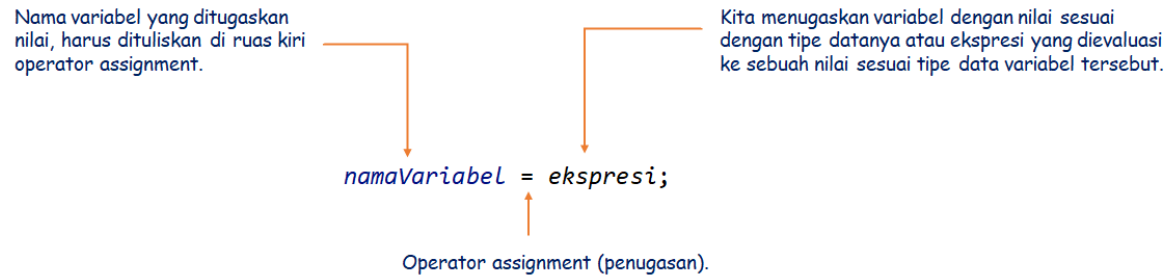
```
num = 80 + (8 * 15);
```

Nilai `num` akan ditugaskan dengan nilai hasil evaluasi ekspresi pada sebelah kanan.

Perlu diperhatikan bahwa `=` bukanlah sama dengan seperti dalam matematika. Pada pemrograman, kita tidak dapat menuliskan ekspresi pada ruas kiri dari operator `=`.

```
num + 10 = 80;      // Menghasilkan ERROR!
```

Secara umum, *statement assignment* mempunyai *Syntax* seperti berikut:



Inisialisasi

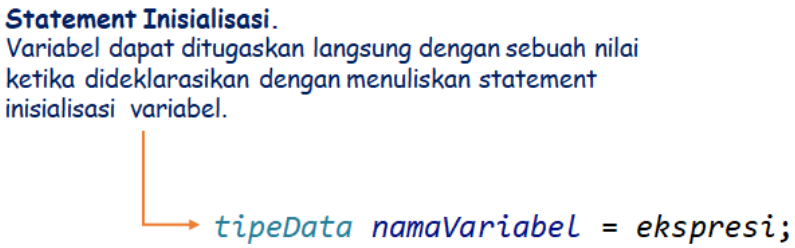
Variabel harus dideklarasikan dengan menuliskan *statement* deklarasi sebelum dapat digunakan. Setelah dideklarasikan, kita dapat menyimpan nilai ke dalam variabel dengan menuliskan *statement assignment*. Rangkaian *statement* berikut adalah contoh *statement* deklarasi dan *statement assignment*.

```
int umur;      // statement deklarasi
umur = 25;     // statement assignment
```

Kita dapat juga menugaskan sebuah nilai ke variabel sebagai bagian dari *statement* deklarasi seperti berikut.

```
int umur = 25;
```

statement di atas disebut *statement* Inisialisasi. Inisialisasi berarti pemberian nilai awal. *Syntax* dari *statement* inisialisasi adalah seperti berikut.

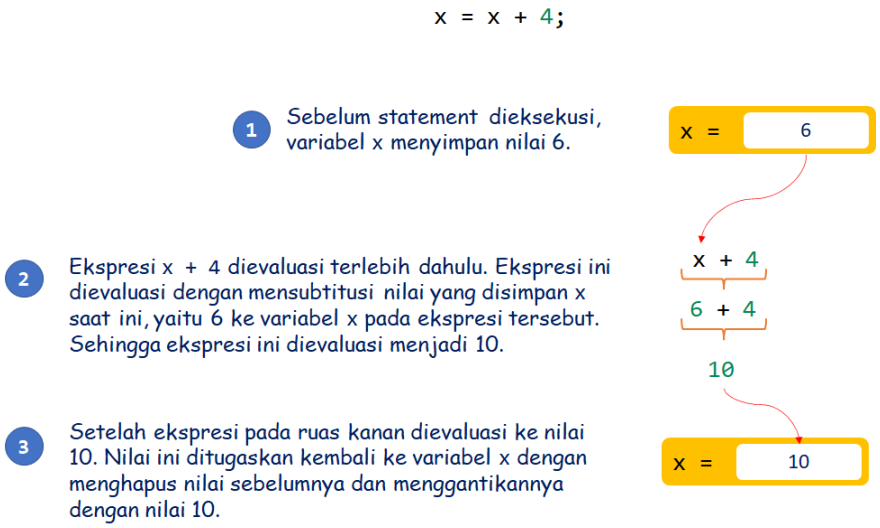


Statement Assignment dengan Variabel yang Sama pada Kedua Ruas

Kita dapat menuliskan *statement assignment* pada kedua ruasnya yang terdapat variabel yang sama seperti berikut.

```
x = x + 4;
```

statement assignment dieksekusi dengan ekspresi pada ruas kanan dievaluasi dengan mensubtitusi x dengan nilainya lalu ditambahkan 4. Kemudian, hasilnya ditugaskan ke x, menggantikan nilai yang sebelumnya disimpan. Misalkan, nilai yang disimpan `x` adalah `6` sebelum *statement* ini, maka setelah *statement* ini dieksekusi nilai yang `x` adalah `10`. Gambar berikut mengilustrasikan eksekusi *statement* di atas.



Program berikut mencontohkan *statement assignment* yang ruas kanannya mengandung variabel yang ditugasinya.

Program (Umur.java)

```
/*
    Program ini mendemonstrasikan penugasan
    yang kedua ruasnya terdapat variabel yang sama
*/
public class Umur
{
    public static void main(String[] args)
    {
        int umur = 25;

        System.out.print("Umur Anda sekarang: ");
        System.out.println(umur);

        umur = umur + 12;

        System.out.print("Umur Anda dalam 12 tahun mendatang: ");
        System.out.println(umur);
    }
}
```

Output Program (Umur.java)

```
Umur Anda sekarang: 25
Umur Anda dalam 12 tahun mendatang: 37
```

Operator *Assignment* Teraugmentasi

Seperti yang telah kita lihat sebelumnya, kita dapat menugaskan kembali suatu variabel dengan nilai yang didapat dari hasil operasi aritmatika yang melibatkan nilai sebelumnya dari variabel tersebut, seperti `x = x + 4`. Java menyediakan operator yang dapat digunakan untuk menyingkat penulisan *statement* ini, yaitu dengan menggunakan operator *assignment* teraugmentasi. Dinamakan operator *assignment* teraugmentasi karena operator ini ditulis dengan menggabungkan dua jenis operator, yaitu operator aritmatika dengan operator *assignment*.

Misalkan *statement*,

```
x = x + 4;
```

dapat disingkat penulisannya menjadi

```
x += 4;
```

Operator `+=` adalah operator assigment teraugmentasi. Tidak hanya untuk operator `+`, semua operator aritmatika juga mempunyai operator *assignment* teraugmentasi terkait, seperti terlihat pada tabel di bawah.

Operator *Assignment* Teraugmentasi	Contoh *Statement*	*Statement* Ekuivalen
`+=`	`c += 7`	`c = c + 7`
`-=`	`d -= 4`	`d = d - 4`
`*=`	`e *= 5`	`e = e * 5`
`/=`	`g /= 2`	`g = g / 2`
`%=`	`i %= 9`	`i = i % 9`

Program berikut mendemonstrasikan penggunaan operator *assignment* teraugmentasi.

Program (Deposito.java)

```
/*
    Program ini mendemonstrasikan operator assignment teraugmentasi
*/
public class Deposito
{
    public static void main(String[] args)
    {
        double deposito = 125500.00;
        double bunga = 0.0475;
```

```
        System.out.print("Saldo deposito awal: ");
        System.out.println(deposito);

        deposito *= (1 + bunga); // deposito = deposito * (1 + bunga);

        System.out.print("Saldo deposito setelah 1 tahun: ");
        System.out.println(deposito);

        deposito *= (1 + bunga); // deposito = deposito * (1 + bunga);

        System.out.print("Saldo deposito setelah 2 tahun: ");
        System.out.println(deposito);
    }
}
```

Output Program (Deposito.java)

```
Saldo deposito awal: 125500.0
Saldo deposito setelah 1 tahun: 131461.25
Saldo deposito setelah 2 tahun: 137705.65937500002
```

Operator *Inkrement* dan *Dekrement*

Dalam menulis program, kita akan sering menuliskan *statement* yang menambahkan nilai 1 ke nilai variabel seperti berikut.

```
counter = counter + 1;
```

statement di atas dapat disingkat penulisannya menjadi seperti berikut.

```
counter++;
```

Operator `++` disebut sebagai operator inkrementasi. Inkrementasi berarti menambah 1. Selain operator `++`, dalam Java juga terdapat operator penyingkat yang digunakan untuk mengurangi nilai 1 dari nilai variabel. Seperti *statement* berikut,

```
counter = counter - 1;
```

dapat dituliskan sebagai berikut.

```
counter--;
```

Operator `--` disebut sebagai operator dekrementasi. Dekrementasi berarti mengurangi nilai 1.

2.5 Konstanta

Nilai dalam variabel dapat diubah sepanjang program. Kita dapat membuat sebuah variabel hanya mempunyai satu nilai sepanjang program dengan menambahkan *keyword* `final` pada deklarasi variabel. Variabel yang tidak dapat diubah nilainya ini disebut sebagai konstanta dan umumnya namanya dituliskan menggunakan huruf besar semua. Contoh berikut memperlihatkan deklarasi dan inisialisasi konstanta.

```
final double BUNGA_BANK = 0.05;
```

Setelah kita menginisialisasi konstanta, jika setelahnya terdapat *statement* yang mengubah nilai konstanta tersebut, *compiler* akan memberikan *error*. Misalkan,

```
final double BUNGA_BANK = 0.05;
BUNGA_BANK = 0.06;     // ERROR!
```

statement kedua yang menugaskan konstanta dengan nilai baru akan menghasilkan *error*.

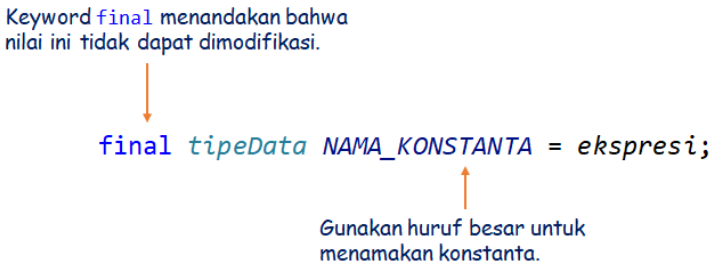
Salah satu kegunaan konstanta dalam program adalah untuk memperjelas program. Misalkan *statement* berikut,

```
jumlah = saldo * 0.05;
```

dapat dituliskan dengan menggantikan nilai 0.05 menggunakan konstanta yang memperjelas arti dari ekspresi

```
jumlah = saldo * BUNGA_BANK;
```

Secara umum, *Syntax* deklarasi konstanta adalah seperti berikut.



Konstanta dalam *class* `Math`

Selain *method-method*, *class* `Math` juga menyediakan konstanta matematika:

- `Math.PI` untuk konstanta π.
- `Math.E` untuk konstanta e.

Salah satu contoh penggunaan `Math.PI` adalah untuk menghitung volume dari bola yang dalam ekspresi matematika ditulis seperti berikut:

$$\frac{4}{3}\pi r^3$$

Ekspresi Java untuk ekspresi matematika di atas dapat kita tuliskan seperti berikut.

```
(4.0 / 3.0) * Math.PI * Math.pow(r, 3)
```

2.6 Konversi antar Tipe Data

Terkadang, kita memerlukan konversi dari satu tipe data ke tipe data lain. Misalkan, ketika kita perlu menugaskan nilai *floating point* ke variabel bertipe *integer* atau sebaliknya. Proses konversi antar tipe data ada yang dilakukan otomatis oleh Java dan ada yang harus dilakukan secara manual dengan menggunakan operator tertentu.

Konversi Otomatis

Java akan mengkonversi otomatis nilai *floating point* yang ditugaskan ke variabel bertipe *integer*. Misalkan, perhatikan rangkaian *statement* berikut.

```
int myInt = 9;
double myDouble = myInt;
```

Pada rangkaian *statement* di atas, kita menugaskan nilai dari variabel `myInt` yang bertipe `int`, ke variabel `myDouble` yang bertipe `double`. Rangkaian *statement* ini tidak menyebabkan *error*. Setelah rangkaian *statement* di atas dieksekusi, variabel `myDouble` akan menyimpan nilai `9.0`. Ketika kita menugaskan nilai *integer* ke variabel bertipe *floating-point*, Java akan secara otomatis mengkonversi nilai *integer* tersebut ke nilai *floating-point* dengan menambahkan `.0`.

Program berikut mendemonstrasikan konversi otomatis dari tipe `int` ke tipe `double`.

Program (KonversiDouble.java)

```
/*
    Program ini mendemonstrasikan konversi otomatis
    dari tipe int ke tipe double.
*/
public class KonversiDouble
{
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Output Program (KonversiDouble.java)

```
9
9.0
```

Konversi Manual (*Casting*)

Konversi secara otomatis hanya dapat dilakukan jika kita mengkonversi tipe data *integer* ke *floating point*. Jika kita melakukan sebaliknya kita akan mendapatkan *error*. Misalkan,

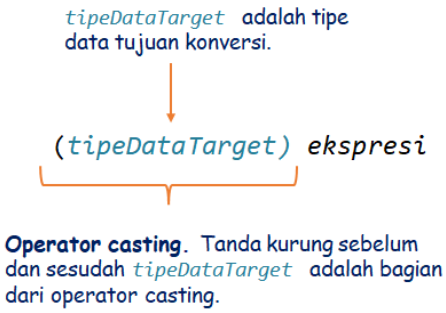
```
double myDouble = 9.76;
int myInt = myDouble;    // ERROR!!
```

Java tidak memperbolehkan konversi data tipe *floating point* ke *integer* karena bagian desimal dari data akan hilang (pada contoh `.76` akan hilang). Kita dapat memaksakan konversi dari *floating point* ke *integer* dengan konsekuensi bagian desimal dari data hilang dengan melakukan ***casting***. Contoh di atas dapat kita perbaiki seperti berikut.

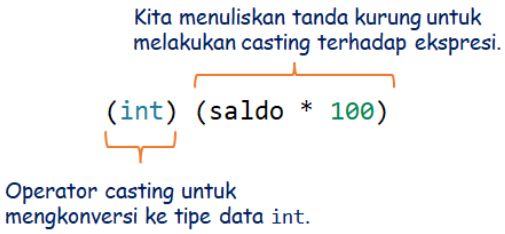
```
double myDouble = 9.76;
int myInt = (int) myDouble;    // myInt menyimpan 9
```

Operator `(int)` disebut operator *casting* untuk konversi ke tipe `int`.

Syntax penulisan ekspresi operasi *casting* adalah sebagai berikut.



Berikut adalah contoh operasi *casting* terhadap ekspresi.



Program berikut mendemonstrasikan operasi *casting* dari *floating point* ke *integer*.

Program (Casting.java)

```java
/*
    Program ini mendemonstrasikan operasi casting.
*/
public class Casting
{
  public static void main(String[] args) {
    double myDouble = 9.78;
    int myInt = (int) myDouble; // Manual casting: double ke int

    System.out.println(myDouble);   // Output 9.78
    System.out.println(myInt);      // Output 9
  }
}
```

Output Program (Casting.java)

```
9.78
9
```

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.