2.7 Arithmetic

Most programs perform arithmetic calculations. The **arithmetic operators** are summarized in Fig. 2.11. Note the use of various special symbols not used in algebra. The **asterisk** (*****) indicates multiplication, and the percent sign (**%**) is the **remainder operator**, which we'll discuss shortly. The arithmetic operators in Fig. 2.11 are *binary* operators, because each operates on *two* operands. For example, the expression `f + 7` contains the binary operator `+` and the two operands `f` and `7`.

Java operation	Operator	Algebraic expression	Java expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	bm	`b * m`
Division	`/`	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	`%`	$r \bmod s$	`r % s`

Fig. 2.11

Arithmetic operators.

Integer division yields an integer quotient. For example, the expression `7 / 4` evaluates to `1`, and the expression `17 / 5` evaluates to `3`. Any fractional part in integer division is simply *truncated* (i.e., *discarded*)—no *rounding* occurs. Java provides the remainder operator, `%`, which yields the remainder after division. The expression `x % y` yields the remainder after `x` is divided by `y`. Thus, `7 % 4` yields `3`, and `17 % 5` yields `2`. This operator is most commonly used with integer operands but it can also be used with other arithmetic types. In this chapter's exercises and in later chapters, we consider several interesting applications of the remainder operator, such as determining whether one number is a multiple of another.

Arithmetic Expressions in Straight-Line Form

Arithmetic expressions in Java must be written in **straight-line form** to facilitate entering programs into computers. Thus, expressions such as "`a` divided by `b`" must be written as `a/b`, so that all constants, variables and operators appear in a straight line. The following algebraic notation is generally not acceptable to compilers:

$$\frac{a}{b}$$

Parentheses for Grouping Subexpressions

Parentheses are used to group terms in Java expressions in the same manner as in algebraic expressions. For example, to multiply a times the quantity `b + c`, we write

```
a * (b + c)
```

If an expression contains **nested parentheses**, such as

```
((a + b) * c)
```

the expression in the *innermost* set of parentheses (`a + b` in this case) is evaluated *first*.

Rules of Operator Precedence

Java applies the arithmetic operators in a precise sequence determined by the **rules of operator precedence**, which are generally the same as those followed in algebra:

1. Multiplication, division and remainder operations are applied first. If an expression contains several such operations, they’re applied from left to right. Multiplication, division and remainder operators have the same level of precedence.

2. Addition and subtraction operations are applied next. If an expression contains several such operations, the operators are applied from left to right. Addition and subtraction operators have the same level of precedence.

These rules enable Java to apply operators in the correct *order*.1 When we say that operators are applied from left to right, we're referring to their **associativity**. Some associate from right to left. Figure 2.12 summarizes these rules of operator precedence. A complete precedence chart is included in Appendix A.

1. We use simple examples to explain the *order of evaluation*. Subtle order-of-evaluation issues occur in the more complex expressions. For more information, see Chapter 15 of *The Java™ Language Specification* (`https://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html`).

Operator(s)	Operation(s)	Order of evaluation (precedence)
* / %	Multiplication Division Remainder	Evaluated first. If there are several operators of this type, they's evaluated from *left to right*.
+ −	Addition Subtraction	Evaluated next. If there are several operators of this type, they's evaluated from *left to right*.
=	Assignment	Evaluated last.

Fig. 2.12

Precedence of arithmetic operators.

Sample Algebraic and Java Expressions

Let’s consider several sample expressions. Each example shows an algebraic expression and its Java equivalent. The following is an example of an average of five terms:

$$\begin{aligned} &\textit{Algebra:} && m = \frac{a+b+c+d+e}{5} \\ &\textit{Java:} && \mathtt{m = (a + b + c + d + e)/5;} \end{aligned}$$

The parentheses are required because division has higher precedence than addition. The entire quantity `(a + b + c + d + e)` is to be divided by `5`. If the parentheses are erroneously omitted, we obtain `a + b + c + d + e / 5`, which evaluates to the different expression

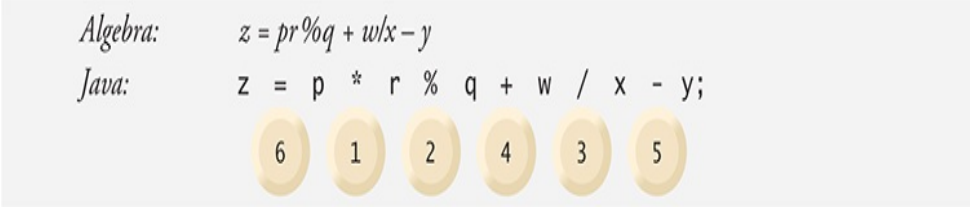
$$a + b + c + d + \frac{e}{5}$$

Here’s an example of the equation of a straight line:

$$\begin{aligned} &\textit{Algebra:} && y = mx + b \\ &\textit{Java:} && \mathtt{y = m * x + b;} \end{aligned}$$

No parentheses are required. The multiplication operator is applied first because multiplication has a higher precedence than addition. The assignment occurs last because it has a lower precedence than multiplication or addition.

The following example contains remainder (%), multiplication, division, addition and subtraction operations:

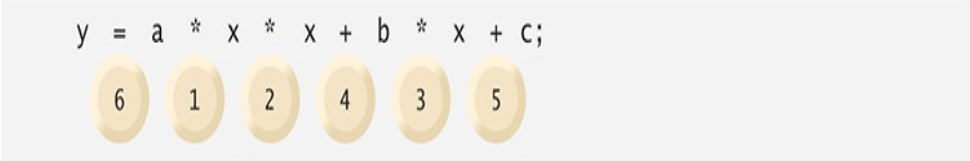


Description

The circled numbers under the statement indicate the *order* in which Java applies the operators. The `*`, `%` and `/` operations are evaluated first in *left-to-right* order (i.e., they associate from left to right), because they have higher precedence than `+` and `-`. The `+` and `-` operations are evaluated next. These operations are also applied from *left to right*. The assignment (=) operation is evaluated last.

Evaluation of a Second-Degree Polynomial

To develop a better understanding of the rules of operator precedence, consider the evaluation of an assignment expression that includes a second-degree polynomial $ax^2 + bx + c$:



Description

The multiplication operations are evaluated first in left-to-right order (i.e., they associate from left to right), because they have

higher precedence than addition. (Java has no arithmetic operator for exponentiation, so x^2 is represented as `x * x`. Section 5.4 shows an alternative for performing exponentiation.) The addition operations are evaluated next from *left to right*. Suppose that `a`, `b`, `c` and `x` are initialized (given values) as follows: `a = 2`, `b = 3`, `c = 7` and `x = 5`. Figure 2.13 illustrates the order in which the operators are applied.

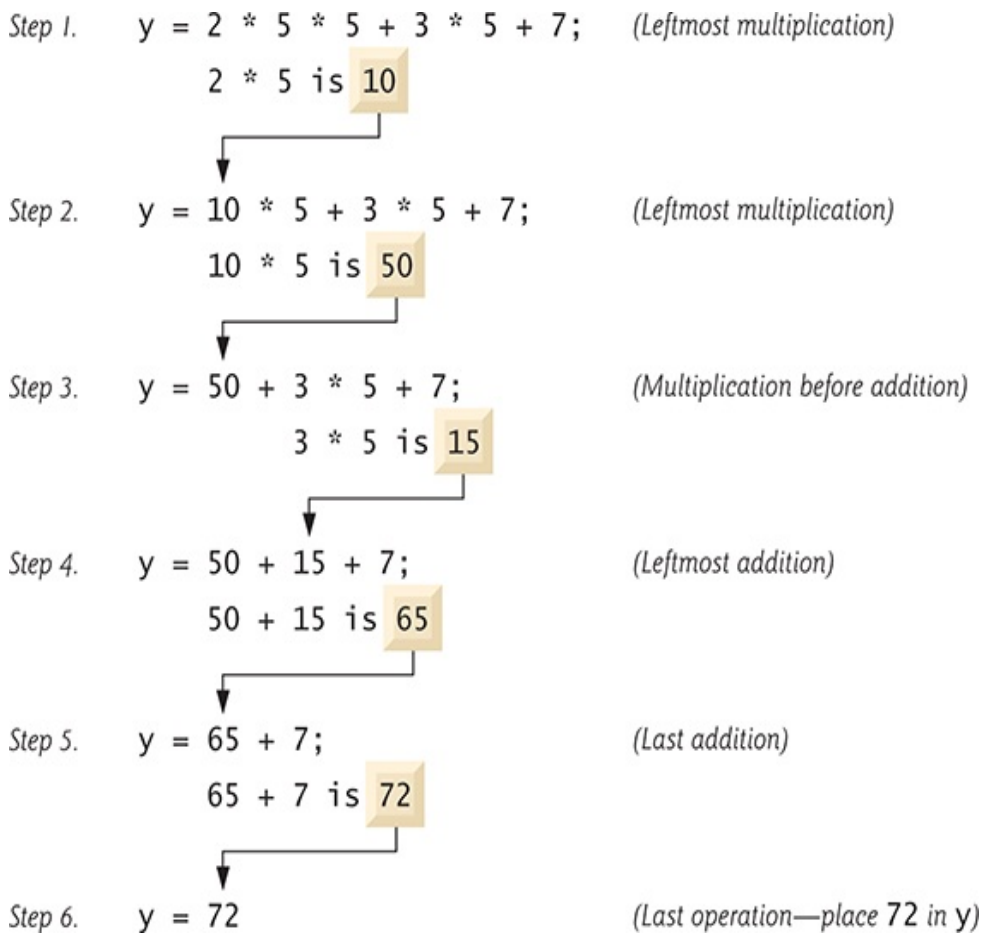


Fig. 2.13

Order in which a second-degree polynomial is evaluated.

Description

You can use redundant parentheses to make an expression clearer. For example, the preceding statement might be parenthesized as follows:

```
y = (a * x * x) + (b * x) + c;
```