4.13 Increment and Decrement Operators

Java provides two unary operators (summarized in Fig. 4.14) for adding 1 to or subtracting 1 from the value of a numeric variable. These are the unary **increment operator**, **`++`**, and the unary **decrement operator**, **--**. A program can increment by 1 the value of a variable called `c` using the increment operator, `++`, rather than the expression `c = c + 1` or `c+= 1`. An increment or decrement operator that's prefixed to (placed before) a variable is referred to as the **prefix increment** or **prefix decrement operator**, respectively. An increment or decrement operator that's postfixed to (placed after) a variable is referred to as the **postfix increment** or **postfix decrement operator**, respectively.

Operator	Sample expression	Explanation
`++` (prefix increment)	`++a`	Increment `a` by `1`, then use the new value of `a` in the expression in which `a` resides.
`++` (postfix increment)	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by `1`.
`--` (prefix decrement)	`--b`	Decrement `b` by `1`, then use the new value of `b` in the expression in which `b` resides.
`--` (postfix decrement)	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by `1`.

Fig. 4.14

Increment and decrement operators.

Using the prefix increment (or decrement) operator to add 1 to (or subtract 1 from) a variable is known as **preincrementing** (or **predecrementing**). This causes the variable to be incremented (decremented) by 1; then the new value of the variable is used in the expression in which it appears. Using the postfix increment (or decrement) operator to add 1 to (or subtract 1 from) a variable is known as **postincrementing** (or **postdecrementing**). This causes the current value of the variable to be used in the expression in which it appears; then the variable’s value is incremented (decremented) by 1.

Good Programming Practice 4.4

Unlike binary operators, the unary increment and decrement operators should be placed next to their operands, with no intervening spaces.

Difference Between Prefix Increment and Postfix Increment Operators

Figure 4.15 demonstrates the difference between the prefix increment and postfix increment versions of the ++ increment operator. The decrement operator (--) works similarly.

```
// Fig. 4.15: Increment.java
// Prefix increment and postfix increment operato

public class Increment {
   public static void main(String[] args) {
      // demonstrate postfix increment operator
      int c = 5;
      System.out.printf("c before postincrement:
      System.out.printf("    postincrementing c:
      System.out.printf(" c after postincrement:

      System.out.println(); // skip a line

      // demonstrate prefix increment operator
      c = 5;
      System.out.printf(" c before preincrement:
      System.out.printf("     preincrementing c:
      System.out.printf("  c after preincrement:
   }
}
```

```
c before postincrement: 5
    postincrementing c: 5
 c after postincrement: 6

 c before preincrement: 5
     preincrementing c: 6
  c after preincrement: 6
```

Fig. 4.15

Prefix increment and postfix increment operators.

Line 7 initializes the variable `c` to `5`, and line 8 outputs `c`'s initial value. Line 9 outputs the value of the expression `c++`. This expression postincrements the variable `c`, so `c`'s *original* value (`5`) is output, then `c`'s value is incremented (to 6). Thus, line 9 outputs `c`'s initial value (`5`) again. Line 10 outputs `c`'s new value (`6`) to prove that the variable's value was indeed incremented in line 9.

Line 15 resets `c`'s value to `5`, and line 16 outputs `c`'s value. Line 17 outputs the value of the expression `++c`. This expression preincrements `c`, so its value is incremented; then the *new* value (`6`) is output. Line 18 outputs `c`'s value again to show that the value of `c` is still `6` after line 17 executes.

Simplifying Statements with the Arithmetic Compound Assignment, Increment and Decrement Operators

The arithmetic compound assignment operators and the increment and decrement operators can be used to simplify program statements. For example, the three assignment statements in Fig. 4.12 (lines 23, 26 and 30)

```
passes = passes + 1;
failures = failures + 1;
studentCounter = studentCounter + 1;
```

can be written more concisely with compound assignment operators as

```
passes += 1;
failures += 1;
studentCounter += 1;
```

with prefix increment operators as

```
++passes;
++failures;
++studentCounter;
```

or with postfix increment operators as

```
passes++;
failures++;
studentCounter++;
```

When incrementing or decrementing a variable in a statement by itself, the prefix increment and postfix increment forms have the *same* effect, and the prefix decrement and postfix decrement forms have the *same* effect. It’s only when a variable appears in the context of a larger expression that preincrementing and postincrementing the variable have different effects (and similarly for predecrementing and postdecrementing).

Common Programming Error 4.8

Attempting to use the increment or decrement operator on an expression other than one to which a value can be assigned is a syntax error. For example, writing `++(x + 1)` *is a syntax error, because* `(x + 1)` *is not a variable.*

Operator Precedence and Associativity

Figure 4.16 shows the precedence and associativity of the operators we've introduced. They're shown from top to bottom in decreasing order of precedence. The second column describes the operators' associativity. The conditional operator (`?:`); the unary operators increment (`++`), decrement (`--`), plus (`+`) and minus (`-`); the cast operators and the assignment operators `=`, `+=`, `-=`, `*=`, `/=` and `%=` associate from *right to left*. The other operators associate from left to right. The third column lists the type of each group of operators.

Error-Prevention Tip 4.6

Refer to the operator precedence and associativity chart (Appendix A) when writing expressions containing many operators. Confirm that the operators in the expression are performed in the order you expect. If you’re uncertain about the order of evaluation in a complex expression, break the expression into smaller statements or use parentheses to force the order of evaluation, exactly as you’d do in an algebraic expression. Be sure to observe that some operators such as assignment (=) associate right to left rather than left to right.

Operators	Associativity	Type
`++` `--`	right to left	unary postfix
`++` `--` `+` `-` `(`*type*`)`	right to left	unary prefix
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<` `<=` `>` `>=`	left to right	relational
`==` `!=`	left to right	equality
`?:`	right to left	conditional
`=` `+=` `-=` `*=` `/=` `%=`	right to left	assignment

Fig. 4.16

Precedence and associativity of the operators discussed so far.