Algoritma dan Pemrograman

Leon Andretti Abdillah

05-01

Expressions, Statements, and Blocks

Introduction

- Now that you understand variables and operators, it's time to learn about *expressions*, *statements*, and *blocks*.
- Operators may be used in building expressions, which compute values; expressions are the core components of statements; statements may be grouped into blocks.

Expressions

- An expression is a syntactic construction that has a value.
- Expressions are the basic way to create values. Expressions are created by combining literals (constants), variables, and method calls by using operators. Parentheses can be used to control the order of evaluation.
- An *expression* is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value.

Assignment expression

- An assignment expression has the following form.
  - ***variable-expression assignment-operator expression***
- The variable expression can be just the name of a variable, or it can be an expression that selects a variable using array indices. The value type of the right-hand-side expression must be compatible with the variable type. An assignment expression is most often used for its side effect: it changes the value of the variable selected by the variable expression to the value of the expression on the right-hand side. The value of the assignment expression is the value that is assigned to the selected variable.
- In most common assignment expressions, the assignment operator is **=**. Then the assignment expression has the following form.
  - ***variable-expression = expression***

Expressions examples

- Somes examples of expressions, illustrated in bold below:
  - int **cadence = 0**;
  - **anArray[0] = 100**;
  - System.out.println("**Element 1 at index 0: " + nArray[0]**);
  - int **result = 1 + 2**; // result is now 3
  - if (**value1 == value2**)
    - System.out.println("**value1 == value2**");

Compound expression

- In this particular example, the order in which the expression is evaluated is unimportant because the result of multiplication is independent of order; the outcome is always the same, no matter in which order you apply the multiplications.
  - `1 * 2 * 3`
- However, this is not true of all expressions. For example, the following expression gives different results, depending on whether you perform the addition or the division operation first:
  - `x + y / 100` // ambiguous

Compound expression using parenthesis

- You can specify exactly how an expression will be evaluated using balanced parenthesis: **(** and **)**. For example, to make the previous expression unambiguous, you could write the following:
  - `(x + y) / 100` // unambiguous, recommended

Compound expression using parenthesis

- If you don't explicitly indicate the order for the operations to be performed, the order is determined by the precedence assigned to the operators in use within the expression. Operators that have a higher precedence get evaluated first. For example, the division operator has a higher precedence than does the addition operator. Therefore, the following two statements are equivalent:
  - `x + y / 100`
  - `x + (y / 100)` // unambiguous, recommended
- When writing compound expressions, be explicit and indicate with parentheses which operators should be evaluated first. This practice makes code easier to read and to maintain.

Statements

- Statements are roughly equivalent to sentences in natural languages.
- A *statement* forms a complete unit of execution.
- A Java statement is the smallest unit that is a complete instruction.
- Statements must end with a semi-colon (`;`).
- Statements generally contain expressions (expressions have a value)

Statements

- The following types of expressions can be made into a statement by terminating the expression with a semicolon (`;`).
  - Assignment expressions
  - Any use of ++ or --
  - Method invocations
  - Object creation expressions
- Such statements are called *expression statements*.

Statements

- In programming, a statement is an instruction to do something.
- It controls the sequence of execution of a program.
- In Java, a statement is terminated with a semicolon and multiple statements can be written on a single line.
  - x = y + 1; z = y + 2;

Assignment statement

- One of the simplest is the Assignment Statement
  - <variable> = <expression>;
- For Example:
  - **int height;**
    height = 34;

Assignment statement

- The left-hand side (LHS) must name a single memory location.
- The right hand side (RHS) must have a value;
- The value of the expression on *RHS* is placed in the memory location named on the *LHS*.
- CORRECT: <variable> = <expression>;
  - **int width, height;**
    width = 476;
    height = 23;
- INCORRECT: <expression> = <variable>;
  - 2 = width;
  - width + 2 = height;
  - width + height = 43

Expression statements 1/2

- Here are some examples of expression statements.
  - // assignment statement
    - `aValue = 8933.234;`
  - // increment statement
    - `aValue++;`
  - // method invocation statement
    - `System.out.println("Hello World!");`
  - // object creation statement
    - `Bicycle myBike = new Bicycle();`

Expression statements 2/2

- In addition to expression statements, there are two other kinds of statements: *declaration statements* and *control flow statements*. A *declaration statement* declares a variable. You've seen many examples of declaration statements already:
- // declaration statement
  - **`double aValue = 8933.234;`**
- Finally, *control flow statements* regulate the order in which statements get executed.

Example If

```
package Package04;

import java.util.Scanner;

public class Decision1 {

    public static void main(String[] args) {

        Scanner scInput = new Scanner(System.in);
        String ket = "";

        System.out.print("Input bilangan : ");
        int bil = scInput.nextInt();

        if (bil >= 55) {
            ket = "Lulus";
            }

        System.out.println("Dengan Nilai " + bil + ", Anda dinyatakan " + ket);
    }
}
```

Blocks

- A *block* is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. The following example, BlockDemo, illustrates the use of blocks:

```
class BlockDemo {
    public static void main(String[] args) {
        boolean condition = true;
        if (condition) { // begin block 1
            System.out.println("Condition is true.");
        } // end block one
        else { // begin block 2
            System.out.println("Condition is false.");
        } // end block 2
    }
}
```