JAVA Textbook

Chapter 4

Writing Java Programs that Make Decisions

Objectives

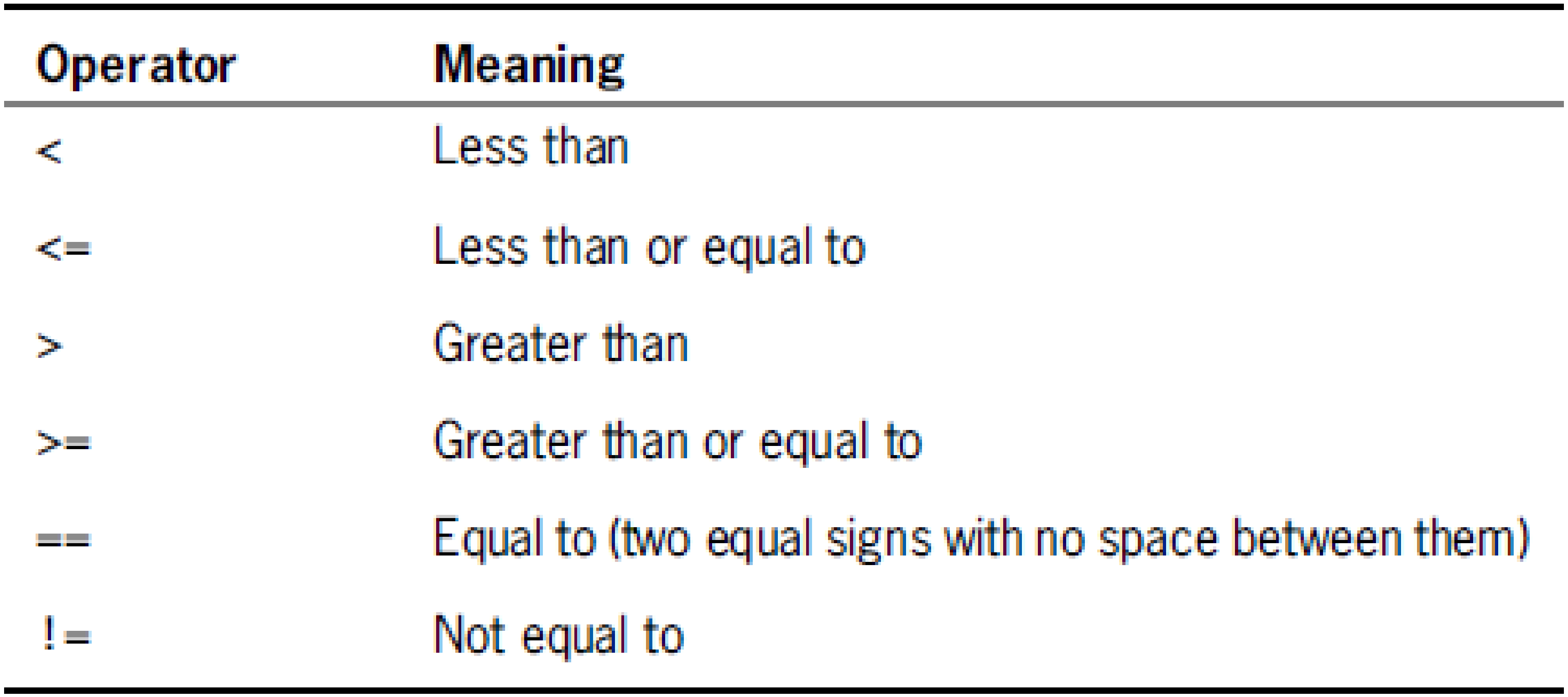
In this chapter, you will learn about:

- Using relational and logical Boolean operators to make decisions in Java programs.
- Comparing String objects.
- Writing decision statements in Java using if statement, if-else statement, nested if statements, and the switch statement.
- Using decision statements to make multiple comparisons by using AND logic and OR logic.

Boolean Operators

- Boolean operators are used in expressions that perform comparisons.
- Such an expression results in a value of true or false.
- Two groups of Boolean operators in Java
  - Relational operators
  - Logical operators

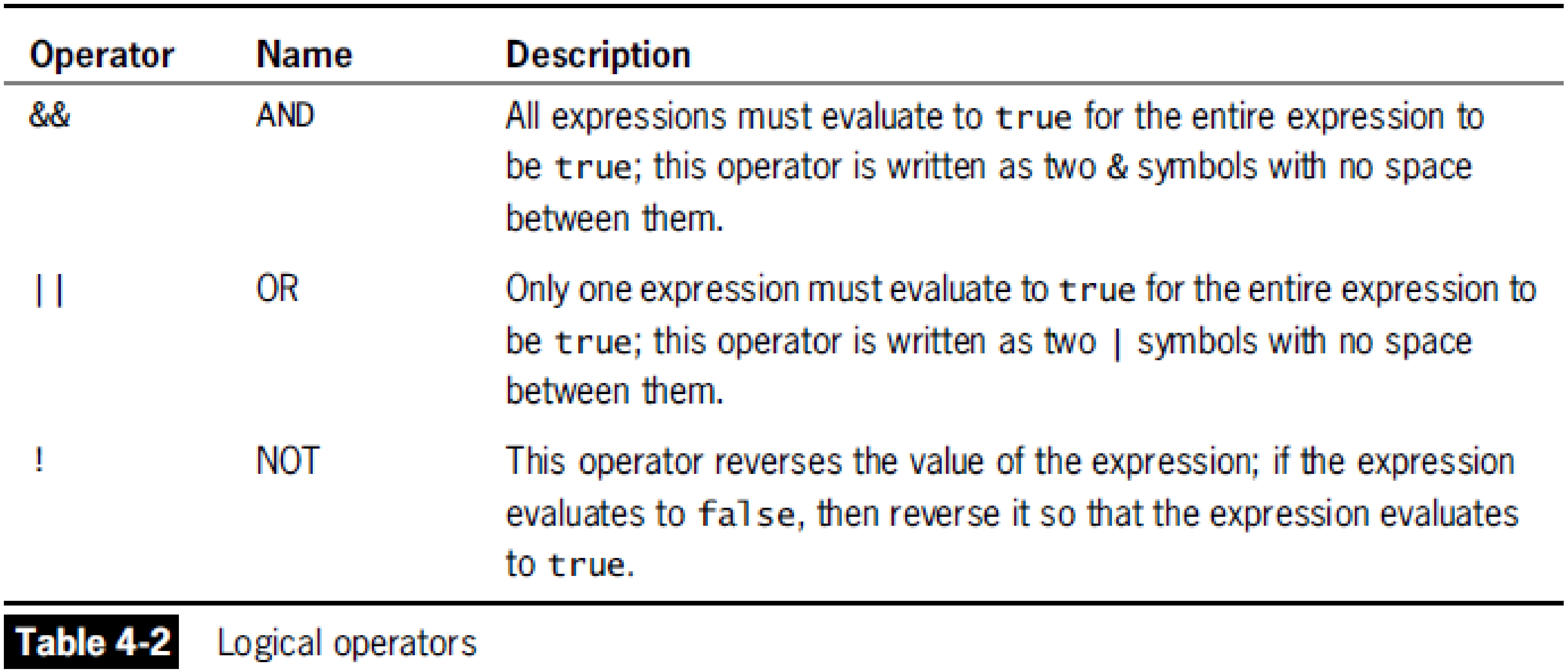
Relational Operators



Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to (two equal signs with no space between them)
!=	Not equal to

Table 4-1 Relational operators

Logical Operators



Operator	Name	Description
&&	AND	All expressions must evaluate to `true` for the entire expression to be `true`; this operator is written as two & symbols with no space between them.
||	OR	Only one expression must evaluate to `true` for the entire expression to be `true`; this operator is written as two | symbols with no space between them.
!	NOT	This operator reverses the value of the expression; if the expression evaluates to `false`, then reverse it so that the expression evaluates to `true`.

Table 4-2 Logical operators

- Perform more than one comparison, but receive only one answer.

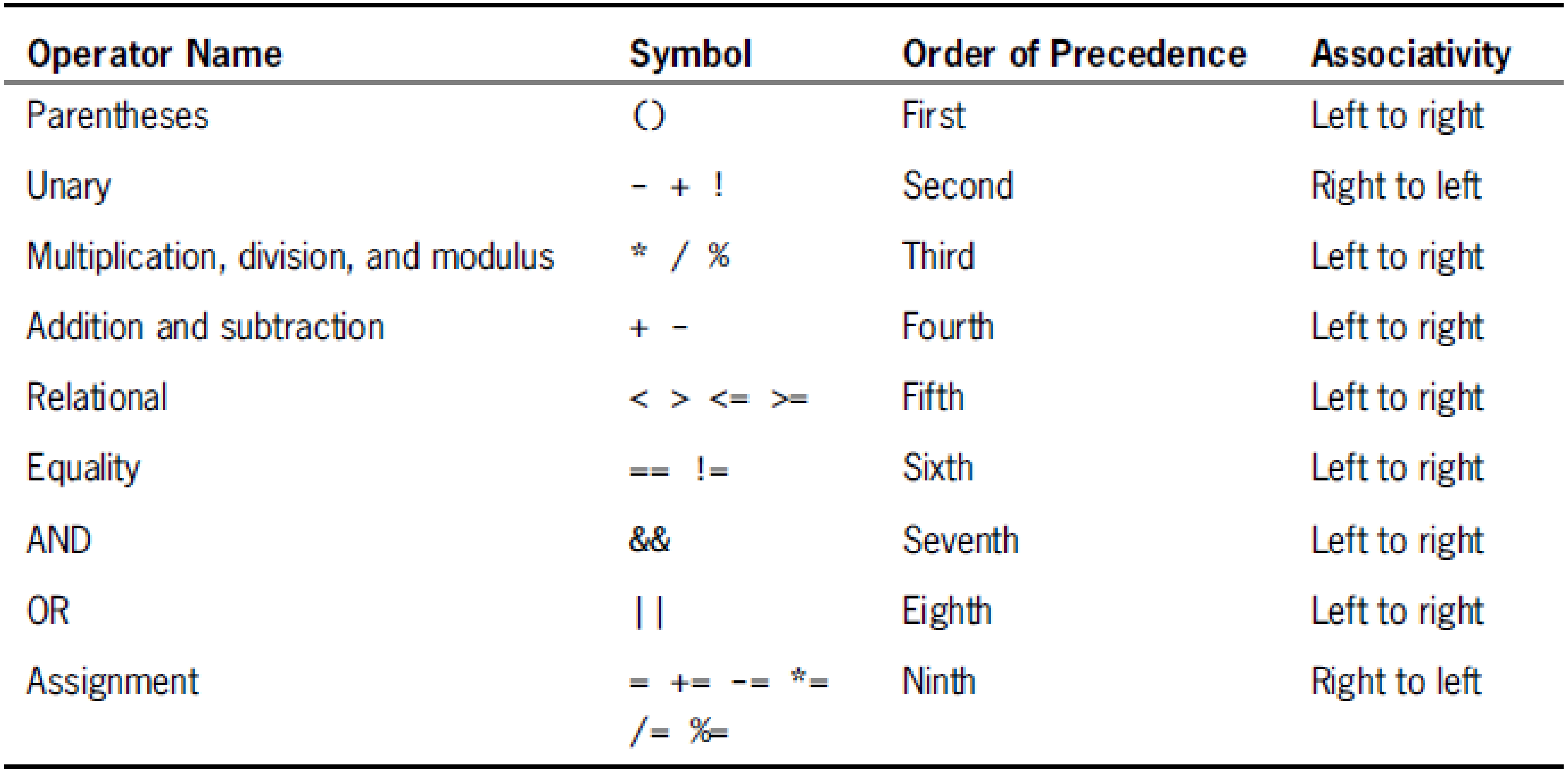
Logical Operators

Example:

int number1 = 10, number2 = 15;

- (number1 > number2) || (number1 == 10) evaluates to true
- (number1 > number2) && (number1 == 10) evaluates to false
- (number1 != number2) && (number1 == 10) evaluates to true
- !(number1 == number2) evaluates to true

Precedence & Associativity



Operator Name	Symbol	Order of Precedence	Associativity
Parentheses	()	First	Left to right
Unary	- + !	Second	Right to left
Multiplication, division, and modulus	* / %	Third	Left to right
Addition and subtraction	+ -	Fourth	Left to right
Relational	< > <= >=	Fifth	Left to right
Equality	== !=	Sixth	Left to right
AND	&&	Seventh	Left to right
OR	||	Eighth	Left to right
Assignment	= += -= *= /= %=	Ninth	Right to left

Table 4-3 Order of precedence and associativity

Precedence & Associativity

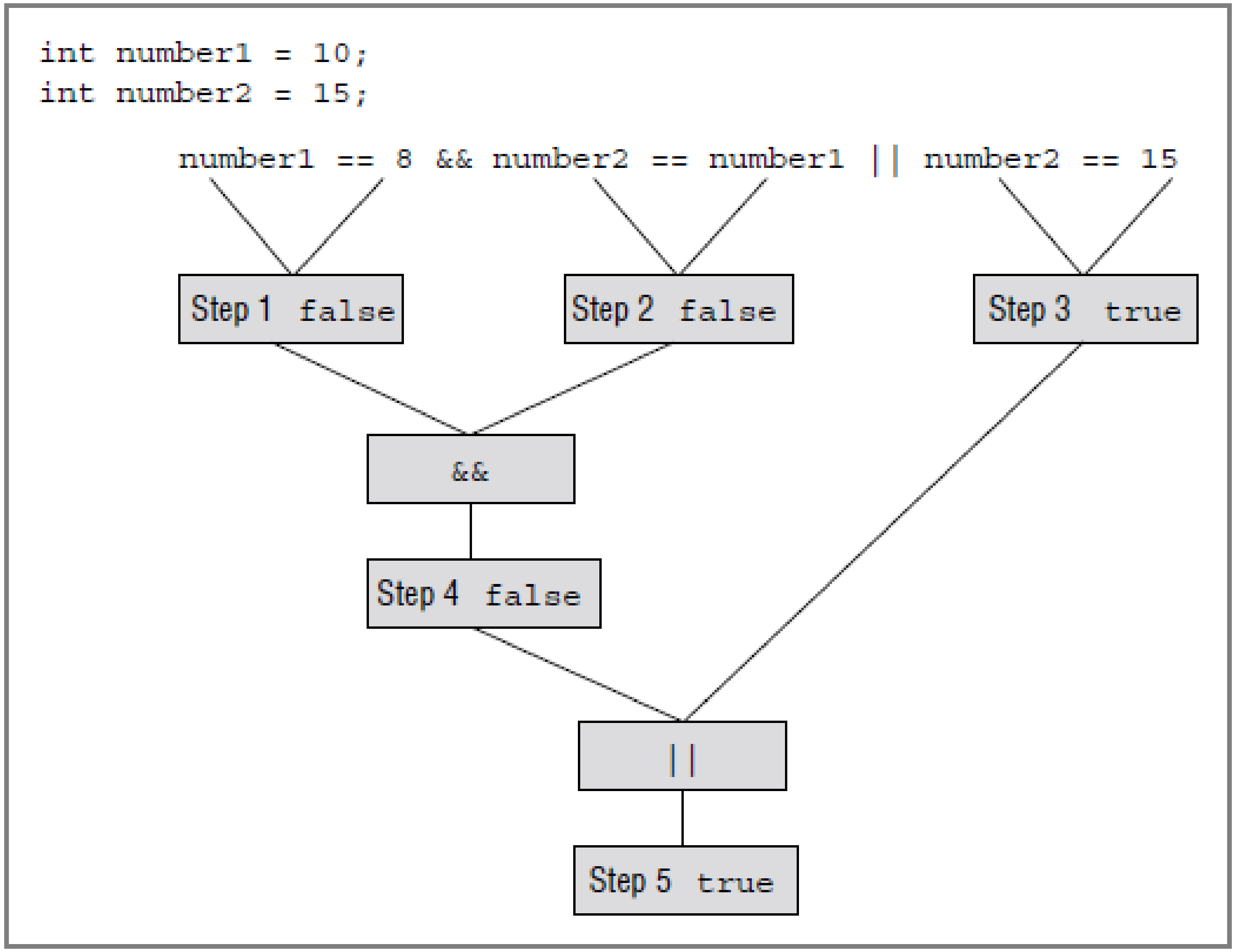
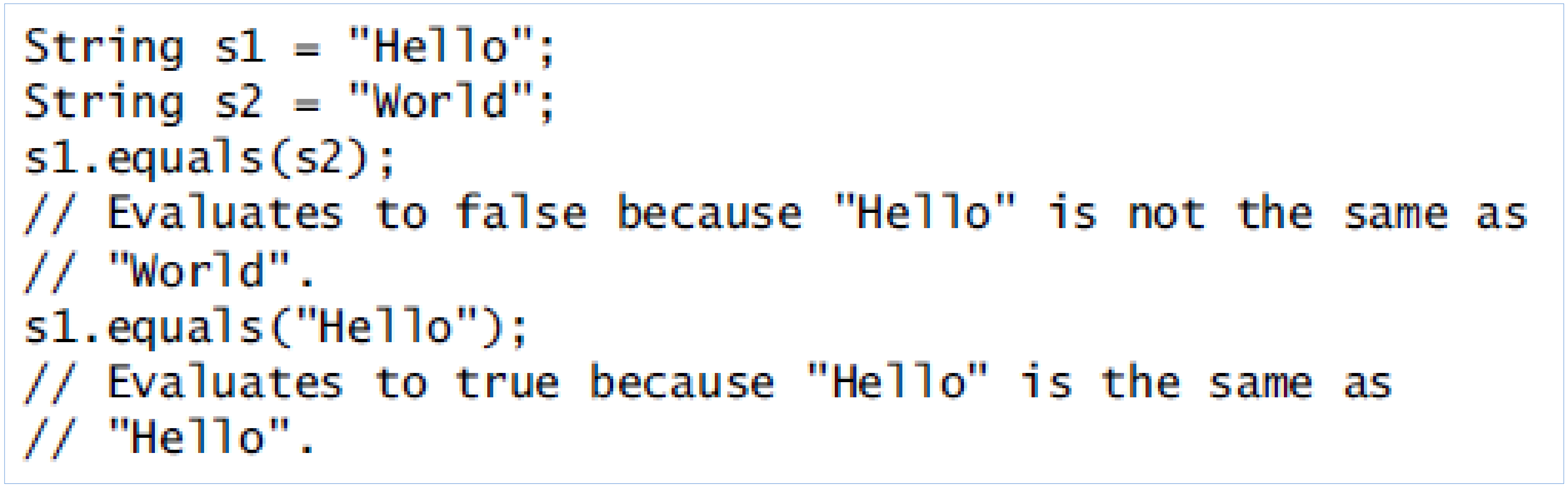

```
int number1 = 10;
int number2 = 15;
```

Figure 4-1 Evaluation of expression using relational and logical operators

Comparing Strings

- Relational operators are used to compare primitive data types such as ints and doubles, but not Strings.
- String objects need to be compared in terms of the contents, not the references (that is, locations in memory).
  - In Java, **DO NOT** use the == operator to compare String objects.
  - Although doing so will not generate a syntax error, it will cause a logical error, as the computer will test to see if two String objects are the same object (i.e., have identical references) instead of whether they have the

Comparing Strings

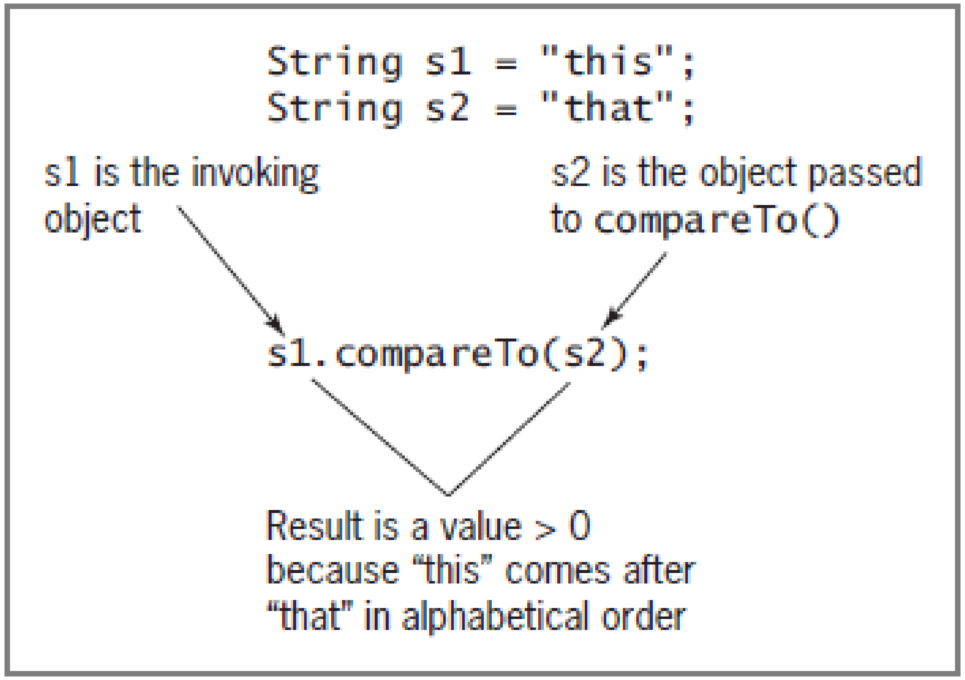


```
String s1 = "Hello";
String s2 = "World";
s1.equals(s2);
// Evaluates to false because "Hello" is not the same as
// "World".
s1.equals("Hello");
// Evaluates to true because "Hello" is the same as
// "Hello".
```

- Use equals() to test two String objects for equality.
- The equals() method returns true if the two String objects are equal, and false if

Comparing Strings

- Can also use compareTo() to compare two String objects.
  - Returns a 0 if two String objects are equal;
  - Returns a value less than 0 if the invoking String object is less than the String object passed to the method;
  - Returns a value greater than 0 if the invoking String object is greater



```
String s1 = "this";
String s2 = "that";
```

s1 is the invoking object
s2 is the object passed to compareTo()
s1.compareTo(s2);
Result is a value > 0 because "this" comes after "that" in alphabetical order

Figure 4-2 Using the compareTo() method

Comparing Strings

```
String s1 = "whole";
String s2 = "whale";
// The next statement evaluates to a value greater than
// 0 because the contents of s1, "whole", are greater
// than the contents of s2, "whale."
s1.compareTo(s2);
// The next statement evaluates to a value less than 0
// because the contents of s2, "whale", are less than the
// contents of s1, "whole."
s2.compareTo(s1);
```

```
String s1 = "whole";
s1.compareTo("whole"); // Evaluates to 0, because
                       // they are equal.
```

Decision Statements

- Used to change the flow of control (order of execution) in a program.
- Also known as branching statements, because they cause the computer to choose from one or more branches (or paths) to continue.
- Several types:
  - If statement
  - If-else statement
  - Nested if statements
  - Switch statement

If Statement

```
if(expression)
  statementA;
```

```
int customerAge = 53;
int discount, numUnder = 0;
if(customerAge < 65)
{
   discount = 0;
   numUnder += 1;
}
System.out.println("Discount : " + discount);
```

```
String dentPlan = "Y";
double grossPay = 500.00;
if(dentPlan.equals("Y"))
   grossPay = grossPay - 23.50;
```

- A single-path (single-alternative) decision statement.

If-Else Statement

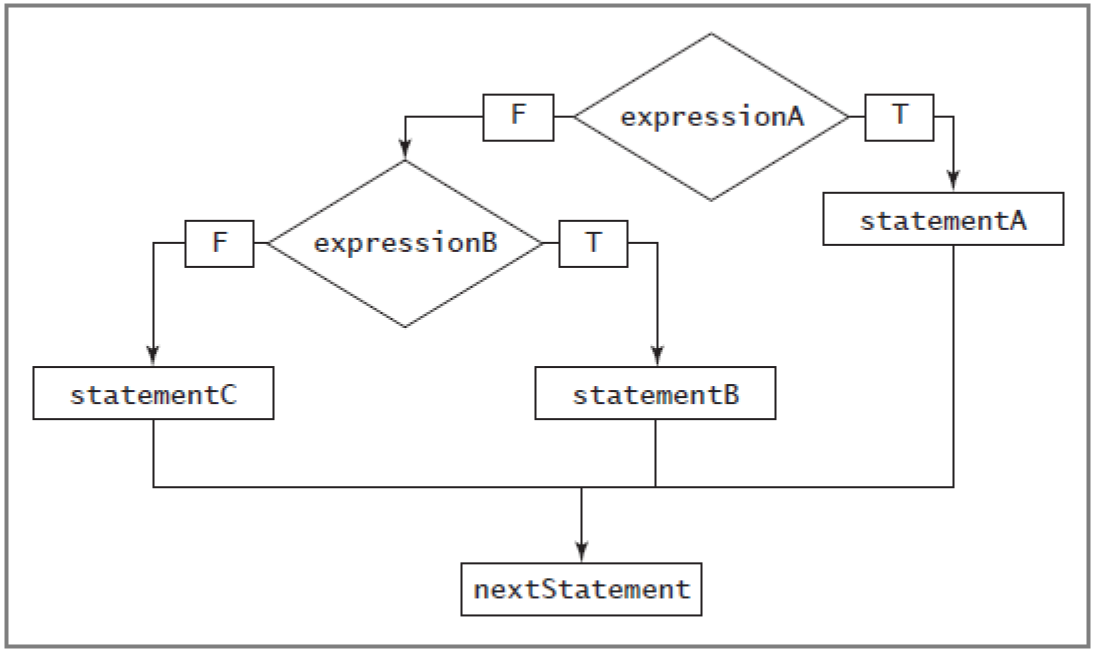
```
if(expression)
    statementA;
else
    statementB;
```

```
int hoursWorked = 45;
double rate = 15.00;
double grossPay;
String overtime = "Yes";
final int HOURS_IN_WEEK = 40;
final double OVERTIME_RATE = 1.5;
if(hoursWorked > HOURS_IN_WEEK)
{
   overtime = "Yes";
   grossPay = HOURS_IN_WEEK * rate +
      (hoursWorked - HOURS_IN_WEEK) *
      OVERTIME_RATE * rate;
}
else
{
   overtime = "No";
   grossPay = hoursWorked * rate;
}
System.out.println("Overtime: " + overtime);
System.out.println("Gross Pay: $" + grossPay);
```

- A dual-path (dual-alternative) decision statement

Nested If Statements

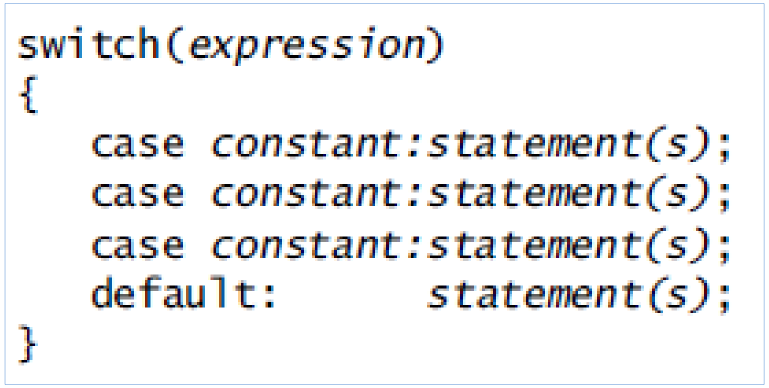
```
if(expressionA)
    statementA;
else if(expressionB)
    statementB;
else
    statementC;
```



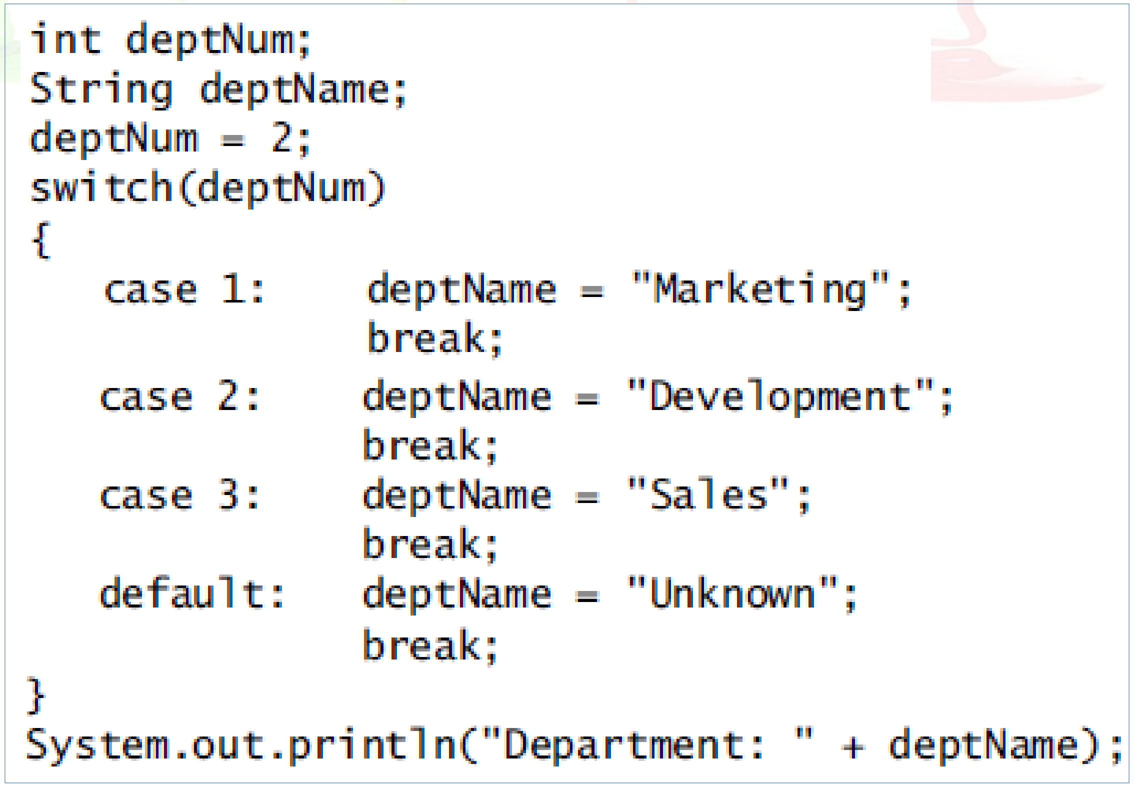
```
if(empDept <= 3)
   supervisorName = "Dillon";
else if(empDept <= 7)
   supervisorName = "Escher";
else
   supervisorName = "Fontana";
System.out.println("Supervisor: " + supervisorName);
```

- A multipath decision statement – more than two possible paths.

Switch Statement



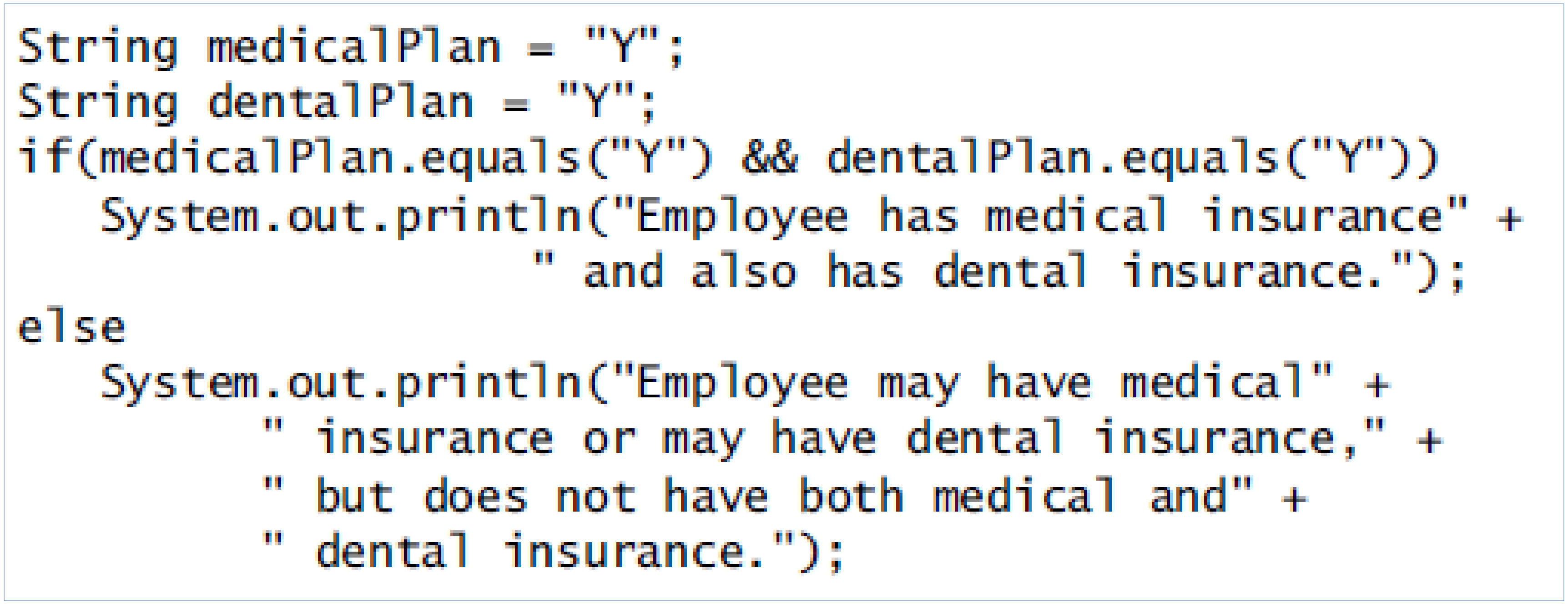
```
switch(expression)
{
   case constant:statement(s);
   case constant:statement(s);
   case constant:statement(s);
   default:      statement(s);
}
```



```
int deptNum;
String deptName;
deptNum = 2;
switch(deptNum)
{
   case 1:     deptName = "Marketing";
               break;
   case 2:     deptName = "Development";
               break;
   case 3:     deptName = "Sales";
               break;
   default:    deptName = "Unknown";
               break;
}
System.out.println("Department: " + deptName);
```

- Also a multipath decision statement.
  - Compares an expression with several integer constants.
  - Easier to read and to maintain than nested if statements.
  - If a break statement is omitted in a case, all the code up to the next break statement or a closing curly brace is executed. Typically not the way to use it

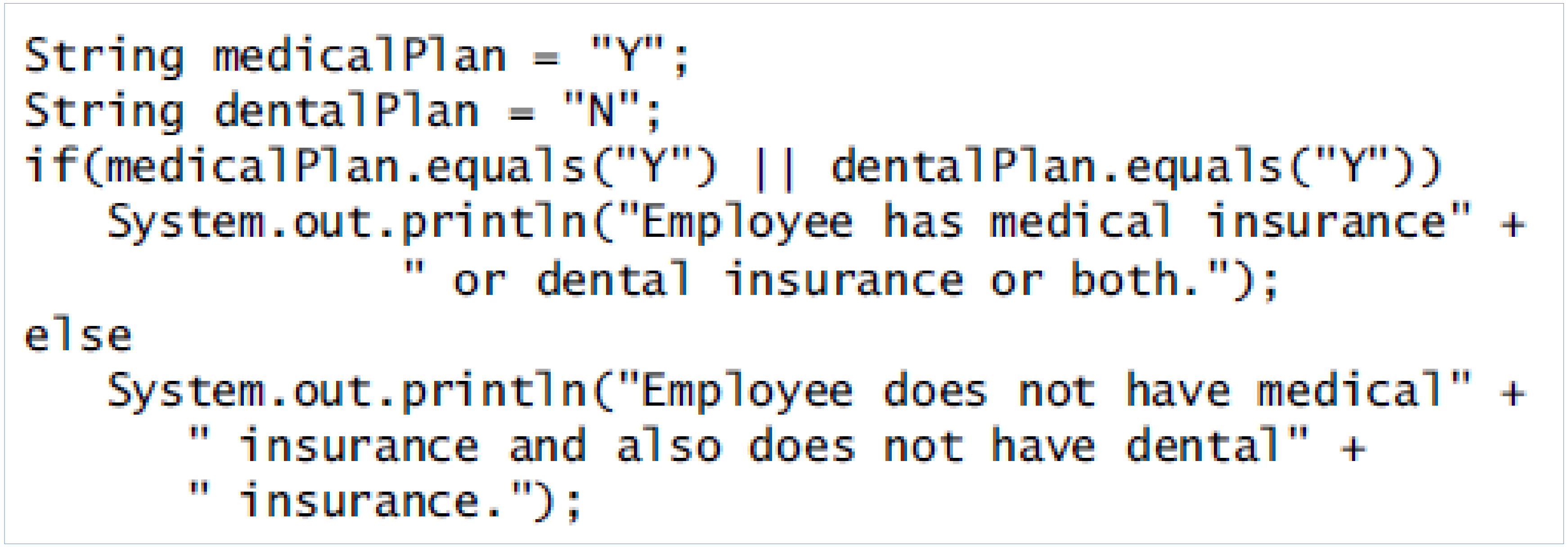
Multiple Comparisons



```
String medicalPlan = "Y";
String dentalPlan = "Y";
if(medicalPlan.equals("Y") && dentalPlan.equals("Y"))
   System.out.println("Employee has medical insurance" +
                    " and also has dental insurance.");
else
   System.out.println("Employee may have medical" +
         " insurance or may have dental insurance," +
         " but does not have both medical and" +
         " dental insurance.");
```

- Using AND logic.
  - All expressions must evaluate to true for the entire expression to be true.

Multiple Comparisons



```
String medicalPlan = "Y";
String dentalPlan = "N";
if(medicalPlan.equals("Y") || dentalPlan.equals("Y"))
   System.out.println("Employee has medical insurance" +
              " or dental insurance or both.");
else
   System.out.println("Employee does not have medical" +
      " insurance and also does not have dental" +
      " insurance.");
```

- Using OR logic.
  - Only one expression must evaluate to true for the entire expression to be true.

Thank You!