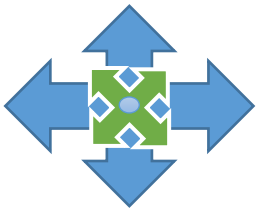
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>Selections<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

*Brought To You By*



***{My Life} Programming School***

***Learn Java By Reading First, Then Watch Someone Tackling Exercises Programmatically By Visiting The Link.***
***https://www.mylifeprogrammingschool.com***

## 3.1 Introduction

{Key Point} The program can decide which statements to execute based on a condition. For instance if the radius is negative, you don't want the program to compute the area. Therefore you need a way to let the programs know what to do in such cases. You can use the following selection statement to make sure you deal with non-negative values of "radius":

```
if (radius < 0) {
        System.out.println("Incorrect input");
}
else {
        area = radius * radius * 3.14159;
        System.out.println("Area is " + area);
}
```

Selection statements use conditions that are Boolean expressions. A *Boolean expression* is an expression that evaluates to a *Boolean value*: **true** or **false**. We now introduce Boolean types and relational operators.

## 3.2 boolean Data Type

{Key Point} The **boolean** data type declares a variable with the value either **true** or **false**. Java provides six *relational operators* (also known as *comparison operators*), shown below(radius is 5):

| Java Operator | Name | Example | Result |
|---|---|---|---|
| **<** | less than | **radius < 0** | **false** |
| **<=** | less than or equal to | **radius <= 0** | **false** |
| **>** | greater than | **radius > 0** | **true** |

| | | |
|---|---|---|
| >= | greater than or equal to | radius >= 0 true |
| == | equal to | radius == 0 false |
| != | not equal to | radius != 0 true |

```
boolean lightsOn = true; // Assign true to a variable called lightOn.
// The following will display true.
double radius = 1;
System.out.println(radius > 0);
```

**true** and **false** are literals, just like a number such as **10**. They are treated as reserved words and cannot be used as identifiers in the program.

Suppose you want to develop a program to let a first-grader practice addition. The program randomly generates two single-digit integers, **number1** and **number2**, and displays to the student a question such as "What is 1 + 7?," as shown in the sample run in Listing 3.1. After the student types the answer, the program displays a message to indicate whether it is true or false. There are several ways to generate random numbers. For now, generate the first integer using **System.currentTimeMillis() % 10** and the second using **System.currentTimeMillis() / 7 % 10**.

**LISTING 3.1** AdditionQuiz.java

```
1import java.util.Scanner;
2
3 public class AdditionQuiz {
4       public static void main(String[] args) {
5               int number1 = (int)(System.currentTimeMillis() % 10);
6               int number2 = (int)(System.currentTimeMillis() / 7 % 10);
7
8               // Create a Scanner
9               Scanner input = new Scanner(System.in);
10
11               System.out.print(
12              "What is " + number1 + " + " + number2 + "? ");
13
14              int number = input.nextInt();
15
16              System.out.println(
17              number1 + " + " + number2 + " = " + answer + " is " +
18              (number1 + number2 == answer));
19      }
20 }
```

What is 1 + 7? 8
1 + 7 = 8 is true

What is 4 + 8? 9
4 + 8 = 9 is false

Listing 3.1 gives the program. Lines 5–6 generate two numbers, **number1** and **number2**. Line 14 obtains an answer from the user. The answer is graded in line 18 using a Boolean expression **number1 + number2 == answer**.

### 3.3 if Statements

{Key Point} An **if** statement is a construct that enables a program to specify alternative paths of execution. The preceding program displays a message such as "6 + 2 = 7 is false." If you wish the message to be "6 + 2 = 7 is incorrect," you have to use a selection statement to make this minor change. Java has several types of selection statements: one-way **if** statements, two-way **if-else** statements, nested **if** statements, multi-way **if-else** statements, **switch** statements, and conditional expressions.

A one-way **if** statement executes an action if and only if the condition is **true**. The syntax for a one-way **if** statement is:

```
if (boolean-expression) {
        statement(s);
}
```

If the **boolean-expression** evaluates to **true**, the statements in the block are executed. As an example, see the following code:

```
if (radius >= 0) {
        area = radius * radius * PI;
        System.out.println("The area for the circle of radius " +
        radius + " is " + area);
}
```

The block braces can be omitted if they enclose a single statement. For example, the following statements are equivalent.

```
if (i > 0) {
        System.out.println("i is positive");
}

if (i > 0)
```

```
            System.out.println("i is positive");
```

Listing 3.2 gives a program that prompts the user to enter an integer. If the number is a multiple of **5**, the program displays **HiFive**. If the number is divisible by **2**, it displays **HiEven**.

**LISTING 3.2** SimpleIfDemo.java

```
1 import java.util.Scanner;
2
3 public class SimpleIfDemo {
4       public static void main(String[] args) {
5             Scanner input = new Scanner(System.in);
6             System.out.println("Enter an integer: ");
7             int number = input.nextInt();
8
9             if (number % 5 == 0)
10                  System.out.println("HiFive");
11
12            if (number % 2 == 0)
13                  System.out.println("HiEven");
14      }
15 }
```

```
Enter an integer: 4
HiEven

Enter an integer: 30
HiFive
HiEven
```

### 3.4 Two-Way if-else Statements

{Key Point} An **if-else** statement decides the execution path based on whether the condition is true or false. A one-way **if** statement performs an action if the specified condition is **true**. If the condition is **false**, nothing is done. But what if you want to take alternative actions when the condition is **false**? You can use a two-way **if-else** statement. Here is the syntax for a two-way **if-else** statement:

```
      if (boolean-expression) {
```

```
        statement(s)-for-the-true-case;
}
else {
        statement(s)-for-the-false-case;
}
```

If the **boolean-expression** evaluates to **true**, the statement(s) for the true case are executed; otherwise, the statement(s) for the false case are executed. For example, consider the following code:

```
if (radius >= 0) {
        area = radius * radius * PI;
        System.out.println("The area for the circle of radius " +
        radius + " is " + area);
}
else {
        System.out.println("Negative input");
}
```

If **radius >= 0** is **true**, **area** is computed and displayed; if it is **false**, the message **"Negative input"** is displayed. Here is another example below

```
if (number % 2 == 0)
        System.out.println(number + " is even.");
else
        System.out.println(number + " is odd.");
```

### 3.5 Nested if and Multi-Way if-else Statements

{Key Point} An **if** statement can be inside another **if** statement to form a nested **if** statement. The statement in an **if** or **if-else** statement can be any legal Java statement, including another **if** or **if-else** statement. The inner **if** statement is said to be *nested* inside the outer **if** statement. The inner **if** statement can contain another **if** statement; in fact, there is no limit to the depth of the nesting. For example, the following is a nested **if** statement:

```
if (i > k) {
        if (j > k)
                System.out.println("i and j are greater than k");
}
else
        System.out.println("i is less than or equal to k");
```

The **if (j > k)** statement is nested inside the **if (i > k)** statement. The nested **if** statement can be used to implement multiple alternatives. The statement given in below, for instance, prints a letter grade according to the score, with multiple alternatives.

```
if (score >= 90.0)
        System.out.print("A");
else
        if (score >= 80.0)
                System.out.print("B");
        else
                if (score >= 70.0)
                        System.out.print("C");
                else
                        if (score >= 60.0)
                                System.out.print("D");
                        else
                                System.out.print("F");
```

The above solution is equivalent to the following one, however the following one is better:

```
if (score >= 90.0)
        System.out.print("A");
else if (score >= 80.0)
        System.out.print("B");
else if (score >= 70.0)
        System.out.print("C");
else if (score >= 60.0)
        System.out.print("D");
else
        System.out.print("F");
```

The first condition **(score >= 90.0)** is tested. If it is **true**, the grade is **A**. If it is **false**, the second condition **(score >= 80.0)** is tested. If the second condition is **true**, the grade is **B**. If that condition is **false**, the third condition and the rest of the conditions (if necessary) are tested until a condition is met or all of the conditions prove to be

**false**. If all of the conditions are **false**, the grade is **F**. Note that a condition is tested only when all of the conditions that come before it are **false**.

### 3.6 Common Errors and Pitfalls

{Key Point} Forgetting necessary braces, ending an **if** statement in the wrong place, mistaking **==** for **=**, and dangling **else** clauses are common errors in selection statements. Duplicated statements in **if-else** statements and testing equality of double values are common pitfalls. The following errors are common among new programmers.

#### Common Error 1: Forgetting Necessary Braces

```
/* The following solution is wrong in a way that the area will get displayed
regardless of whether or not the condition holds.*/
if (radius >= 0)
        area = radius * radius * PI;
        System.out.println("The area "
        + " is " + area);

/* The following solution is correct since it only displays the area if the
condition is met.*/
if (radius >= 0) {
        area = radius * radius * PI;
        System.out.println("The area "
        + " is " + area);
}
```

#### Common Error 2: Wrong Semicolon at the if Line

Adding a semicolon at the end of an **if** line, as shown in (a) below, is a common mistake.

```
if (radius >= 0); // The semicolon shouldn't be here.
{
        area = radius * radius * PI;
        System.out.println("The area "
        + " is " + area);
}

if (radius >= 0) { }; // The curly braces shouldn't be here.
{
        area = radius * radius * PI;
        System.out.println("The area "
```

```
            + " is " + area);
    }
```

### Common Error 3: Redundant Testing of Boolean Values

To test whether a **boolean** variable is **true** or **false** in a test condition, it is redundant to use the equality testing operator like the code below:

```
if (even == true)
        System.out.println(
        "It is even.");

// The preferred way is the following one.


if (even)
        System.out.println(
        "It is even.");
```

### Common Error 4: Dangling else Ambiguity

The code below has two **if** clauses and one **else** clause. Which **if** clause is matched by the **else** clause? The indentation indicates that the **else** clause matches the first **if** clause.

```
int i = 1, j = 2, k = 3;
if (i > j)
        if (i > k)
                System.out.println("A");
        else
                System.out.println("B");
```

Which is equivalent to the following one:

```
int i = 1, j = 2, k = 3;
if (i > j)
        if (i > k)
                System.out.println("A");
else
        System.out.println("B");
```

To force the **else** clause to match the first **if** clause, you must add a pair of braces:

```
int i = 1, j = 2, k = 3;

if (i > j) {
        if (i > k)
                System.out.println("A");
}
else
        System.out.println("B");
```

This statement displays **B**.

### Common Error 5: Equality Test of Two Floating-Point Values

Floating-point numbers have a limited precision and calculations; involving floating-point numbers can introduce round-off errors. So, equality test of two floating-point values is not reliable. For example, you expect the following code to display **true**, but surprisingly it displays **false**.

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
System.out.println(x == 0.5);
```

Here, **x** is not exactly **0.5**, but is **0.5000000000000001**. You cannot reliably test equality of two floating-point values.

### Common Pitfall: Avoiding Duplicate Code in Different Cases

Often, new programmers write the duplicate code in different cases that should be combined in one place. For example, the highlighted code in the following statement is duplicated.

```
if (inState) {
        tuition = 5000;
        System.out.println("The tuition is " + tuition);
}
else {
        tuition = 15000;
        System.out.println("The tuition is " + tuition);
}
```

This is not an error, but it should be better written as follows:

```
if (inState) {
        tuition = 5000;
}
else {
        tuition = 15000;
}
System.out.println("The tuition is " + tuition);
```

### 3.7 Generating Random Numbers

{Key Point} You can use **Math.random()** to obtain a random double value between **0.0** and **1.0**, excluding **1.0**.

```
// 1. Generate a random number between 0 and 1.
double number1 = Math.random();

// 2. Generate a random single-digit integer
int number2 = (int)(Math.random() * 10);

// 3. Generate a random number 0-1000
short number3 = (short)(Math.random() * 1000);


// 5. Generate a random number between 5.5 and 20.5.
double number2 = 5.5 + Math.random() * 15.0;
```

### 3.8 Logical Operators

{Key Point} The logical operators **!**, **&&**, **||**, and **^** can be used to create a compound Boolean expression. Sometimes, whether a statement is executed is determined by a combination of several conditions. You can use logical operators to combine these conditions to form a compound Boolean expression. *Logical operators*, also known as *Boolean operators*, operate on Boolean values to create a new Boolean value.

| Operator | Name | Description |
|---|---|---|
| **!** | not logical | negation |
| **&&** | and logical | conjunction |
| **\|\|** | or | logical disjunction |
| **^** | exclusive or | logical exclusion |

Listing 3.3 gives a program that checks whether a number is divisible by **2** and **3**, by **2** or **3**, and by **2** or **3** but not both:

**LISTING 3.3** TestBooleanOperators.java

```
1 import java.util.Scanner;
2
3 public class TestBooleanOperators {
4       public static void main(String[] args) {
5               // Create a Scanner
6               Scanner input = new Scanner(System.in);
7
8               // Receive an input
9               System.out.print("Enter an integer: ");
10              int number = input.nextInt();
11
12              if (number % 2 == 0 && number % 3 == 0)
13              System.out.println(number + " is divisible by 2 and 3.");
14
15              if (number % 2 == 0 || number % 3 == 0)
16              System.out.println(number + " is divisible by 2 or 3.");
17
18              if (number % 2 == 0 ^ number % 3 == 0)
19              System.out.println(number +
20              " is divisible by 2 or 3, but not both.");
21      }
22 }
```

```
Enter an integer: 4
4 is divisible by 2 or 3.
4 is divisible by 2 or 3, but not both.

Enter an integer: 18
18 is divisible by 2 and 3.
18 is divisible by 2 or 3.
```

### 3.9 switch Statements

{Key Point} A **switch** statement executes statements based on the value of a variable or an expression. Suppose you want to compute tax payments based on one of the four possible status of a person namely(single filers, married jointly or qualifying widow(er), married filing separately, head of household) you can use a switch statements to accomplish the task.

```
switch (status) {
        case 0: compute tax for single filers;
        break;
```

```
        case 1: compute tax for married jointly or qualifying widow(er);
        break;
        case 2: compute tax for married filing separately;
        break;
        case 3: compute tax for head of household;
        break;
        default: System.out.println("Error: invalid status");
        System.exit(1);
}
```

This statement checks to see whether the status matches the value **0**, **1**, **2**, or **3**, in that order. If matched, the corresponding tax is computed; if not matched, a message is displayed.

Here is the full syntax for the **switch** statement:

```
switch (switch-expression) {
        case value1: statement(s)1;
        break;
        case value2: statement(s)2;
        break;
        ...
        case valueN: statement(s)N;
        break;
        default: statement(s)-for-default;
}
```

■ The **switch-expression** must yield a value of **char**, **byte**, **short**, **int**, or **String** type and must always be enclosed in parentheses. (The **char** and **String** types will be introduced in the next chapter.)

■ The **value1**, . . ., and **valueN** must have the same data type as the value of the **switchexpression**. Note that **value1**, . . ., and **valueN** are constant expressions, meaning that they cannot contain variables, such as **1 + x**.

■ When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached.

■ The **default** case, which is optional, can be used to perform actions when none of the specified cases matches the **switch-expression**.

■ The keyword **break** is optional. The **break** statement immediately ends the **switch** statement.

When both operands of a division are integers, the result of the division is the quotient and the fractional part is truncated. For example, **5 / 2** yields **2**, not **2.5**, and **–5 / 2** yields **-2**, not **–2.5**.

The **%** operator, known as *remainder* or *modulo* operator, yields the remainder after division. The operand on the left is the dividend and the operand on the right is the divisor. Therefore, **7 % 3** yields **1**, **3 % 7** yields **3**, **12 % 4** yields **0**, **26 % 8** yields **2**, and **20 % 13** yields **7**.

### 3.10 Conditional Expressions

{Key Point} A conditional expression evaluates an expression based on a condition. You might want to assign a value to a variable that is restricted by certain conditions. For example, the following statement assigns **1** to **y** if **x** is greater than **0**, and **-1** to **y** if **x** is less than or equal to **0**.

```
if (x > 0)
        y = 1;
else
        y = -1;
```

Alternatively, as in the following example, you can use a conditional expression to achieve the same result.

```
y = (x > 0) ? 1 : -1;
```

Conditional expressions are in a completely different style, with no explicit **if** in the statement. The syntax is:

```
boolean-expression ? expression1 : expression2;
```

```
System.out.println((num % 2 == 0) ? "num is even" : "num is odd");
```

### 3.15 Operator Precedence and Associativity

{Key Point} Operator precedence and associativity determine the order in which operators are evaluated. Suppose that you have this expression:

```
3 + 4 * 4 > 5 * (4 + 3) – 1 && (4 - 3 > 5)
```

What is its value? What is the execution order of the operators? The expression within parentheses is evaluated first. (Parentheses can be nested, in which case the expression within the inner parentheses is executed first.) When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule**.**

The precedence rule defines precedence for operators, as shown in the Table, which contains the operators you have learned so far. Operators are listed in decreasing order of precedence from top to bottom. The logical operators have lower precedence than the relational operators and the relational operators have lower precedence than the arithmetic operators.

1. **var**++ and **var—** (Postfix)
2. +, **–** (Unary plus and minus), **++var** and **—var** (Prefix)
3. (type) (Casting)
4. **!**(Not)
5. ***, /, %** (Multiplication, division, and remainder)
6. +, **–** (Binary addition and subtraction)
7. <, <=, >, >= (Relational)
8. ==, **!=** (Equality)
9. ^ (Exclusive OR)
10. **&&** (AND)
11. || (OR)
12. =, +=, **–=**, ***=**, /=, **%=** (Assignment operator)

If operators with the same precedence are next to each other, their *associativity* determines the order of evaluation. All binary operators except assignment operators are *left associative*.

For example, since **+** and **–** are of the same precedence and are left associative, the expression

a - b + c – d is equivalent to ((a - b) + c) – d
a = b += c = 5 is equivalent to a = (b += (c = 5))

Suppose **a**, **b**, and **c** are **1** before the assignment; after the whole expression is evaluated, **a** becomes **6**, **b** becomes **6**, and **c** becomes **5**. Note that left associativity for the assignment operator would not make sense.

==========End Of Chapter Three==========

Now You Can Checkout Chapter 3 Exercises And Their Corresponding Possible Solutions On https://www.mylifeprogrammingschool.com/java