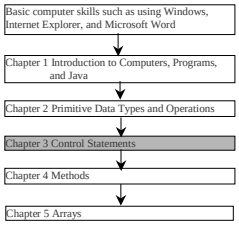
Chapter 3 Control Statements

Prerequisites for Part I



Objectives

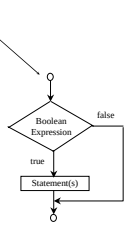
- To understand the flow of control in selection and loop statements (§3.2-3.7).
- To use Boolean expressions to control selection statements and loop statements (§3.2-3.7).
- To implement selection control using if and nested if statements (§3.2).
- To implement selection control using switch statements (§3.2).
- To write expressions using the conditional operator (§3.2).
- To use while, do-while, and for loop statements to control the repetition of statements (§3.4).
- To write nested loops (§3.4).
- To know the similarities and differences of three types of loops (§3.5).
- To implement program control with break and continue (§3.6).

Selection Statements

- `if` Statements
- `switch` Statements
- Conditional Operators

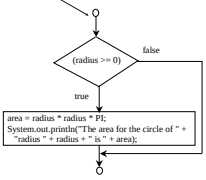
Simple `if` Statements

```
if (booleanExpression) {
  statement(s);
}
```



(A)

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



(B)

Note

Outer parentheses required

```
if ((i > 0) && (i < 10)) {
  System.out.println("i is an " +
    + "integer between 0 and 10");
}
```

(a)

Equivalent

Braces can be omitted if the block contains a single statement

```
if ((i > 0) && (i < 10))
  System.out.println("i is an " +
    + "integer between 0 and 10");
```

(b)

Caution

Adding a semicolon at the end of an if clause is a common mistake.

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

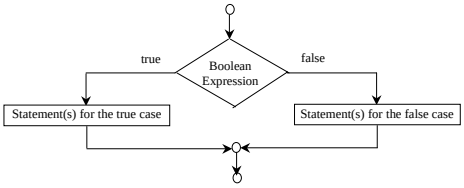
Wrong

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.

The `if...else` Statement

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



`if...else` Example

```
if (radius >= 0) {
  area = radius * radius * 3.14159;

  System.out.println("The area for the “
    + “circle of radius " + radius +
    " is " + area);
}
else {
  System.out.println("Negative input");
}
```


Multiple Alternative if Statements

```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalent

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```


Note

The else clause matches the most recent if clause in the same block.

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

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

(a)

Equivalent

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

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

(b)

Note, cont.

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

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

This statement prints B.

TIP

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

```
boolean even
  = number % 2 == 0;
```

(b)

CAUTION

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

(a)

Equivalent

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

(b)

Example 3.1 Computing Taxes

The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2002 are shown in Table 3.1.

Tax rate	Single filers	Married filing jointly or qualifying widow/widower	Married filing separately	Head of household
10%	Up to $6,000	Up to $12,000	Up to $6,000	Up to $10,000
15%	$6,001 - $27,950	$12,001 - $46,700	$6,001 - $23,350	$10,001 - $37,450
27%	$27,951 - $67,700	$46,701 - $112,850	$23,351 - $56,425	$37,451 - $96,700
30%	$67,701 - $141,250	$112,851 - $171,950	$56,426 - $85,975	$96,701 - $156,600
35%	$141,251- $307,050	$171,951 - $307,050	$85,976 - $153,525	$156,601 - $307,050
38.6%	$307,051 or more	$307,051 or more	$153,526 or more	$307,051 or more

Example 3.1 Computing Taxes, cont.

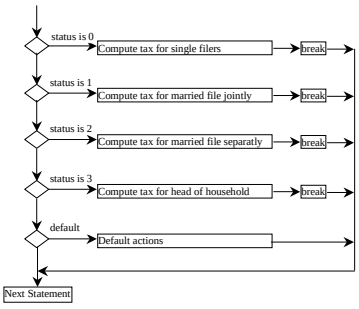
```
if (status == 0) {
  // Compute tax for single filers
}
else if (status == 1) {
  // Compute tax for married file jointly
}
else if (status == 2) {
  // Compute tax for married file separately
}
else if (status == 3) {
  // Compute tax for head of household
}
else {
  // Display wrong status
}
```

Compute TaxWithSelectionStatement | Run

`switch` Statements

```
switch (status) {
  case 0:  compute taxes for single filers;
           break;
  case 1:  compute taxes for married file jointly;
           break;
  case 2:  compute taxes for married file separately;
           break;
  case 3:  compute taxes for head of household;
           break;
  default: System.out.println("Errors: invalid status");
           System.exit(0);
}
```


`switch` Statement Flow Chart



`switch` Statement Rules

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.

```
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;
}
```

`switch` Statement Rules

The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the break statement is not present, the next case statement will be executed.

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

The case statements are executed in sequential order, but the order of the cases (including the default case) does not matter. However, it is good programming style to follow the logical sequence of the cases and place the default case at the end.

Conditional Operator

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

is equivalent to

```
y = (x > 0) ? 1 : -1;
(booleanExpression) ? expression1 : expression2
```

Ternary operator
Binary operator
Unary operator

Conditional Operator

```
if (num % 2 == 0)
  System.out.println(num + “is even”);
else
  System.out.println(num + “is odd”);


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


Conditional Operator, cont.

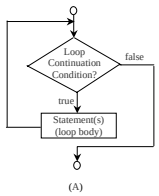
```
(booleanExp) ? exp1 : exp2
```


Repetitions

- `while` Loops
- `do-while` Loops
- `for` Loops
- `break` and `continue`

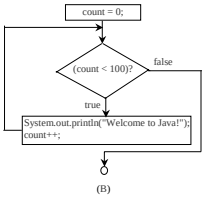
`while` Loop Flow Chart

```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```



(A)

```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```



(B)

Example 3.2: Using while Loops

Problem: Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

TestWhile

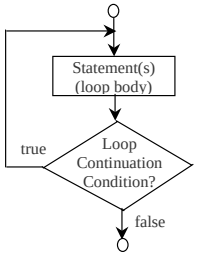
Caution

Don't use floating-point values for equality checking in a loop control. Since floating-point values are approximations, using them could result in imprecise counter values and inaccurate results. This example uses int value for data. If a floating-point type value is used for data, (data != 0) may be true even though data is 0.

```
// data should be zero
double data = Math.pow(Math.sqrt(2), 2) - 2;

if (data == 0)
  System.out.println("data is zero");
else
  System.out.println("data is not zero");
```

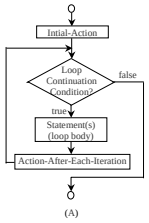

`do-while` Loop




```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

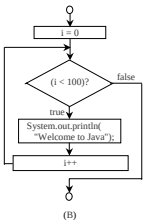

`for` Loops

```
for (initial-action; loop-
    continuation-condition;
    action-after-each-iteration) {
  // loop body;
  Statement(s);
}
```



(A)

```
int i;
for (i = 0; i < 100; i++) {
  System.out.println(
    "Welcome to Java!");
}
```



(B)

Note

The initial-action in a for loop can be a list of zero or more comma-separated expressions. The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. Therefore, the following two for loops are correct. They are rarely used in practice, however.

```
for (int i = 1; i < 100; System.out.println(i++));

for (int i = 0, j = 0; (i + j < 10); i++, j++) {
  // Do something
}
```


Note

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (A), which is an infinite loop, is correct. Nevertheless, I recommend that you use the equivalent loop in (B) to avoid confusion:

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

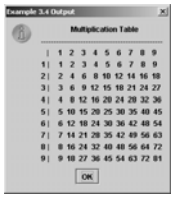
Example 3.3 Using for Loops

Problem: Write a program that sums a series that starts with 0.01 and ends with 1.0. The numbers in the series will increment by 0.01, as follows: 0.01 + 0.02 + 0.03 and so on.

TestSum | Run

Example 3.4 Displaying the Multiplication Table

Problem: Write a program that uses nested for loops to print a multiplication table.



TestMultiplicationTable

Which Loop to Use?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (A) in the following figure can always be converted into the following for loop in (B):

```
while (loop-continuation-condition) {
  // Loop body
}
```
(A)

Equivalent

```
for ( ; loop-continuation-condition; )
  // Loop body
}
```
(B)

A for loop in (A) in the following figure can generally be converted into the following while loop in (B) except in certain special cases (see Review Question 3.19 for one of them):

```
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
}
```
(A)

Equivalent

```
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```
(B)

Recommendations

I recommend that you use the one that is most intuitive and comfortable for you. In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times. A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0. A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

Caution

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

```
for (int i=0; i<10; i++);
{
  System.out.println("i is " + i);
}
```
Logic Error

Caution, cont.

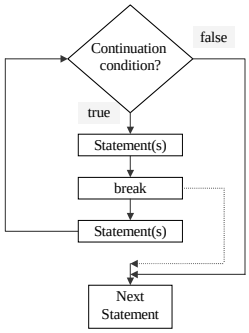
Similarly, the following loop is also wrong:

```
int i=0;
while (i < 10);
{
  System.out.println("i is " + i);
  i++;
}
```
Logic Error

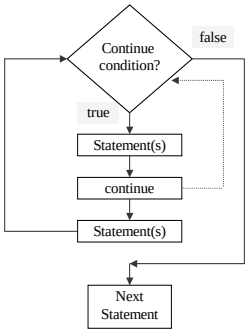
In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
  System.out.println("i is " + i);
  i++;
} while (i<10);
```
Correct

Using the Keywords break and continue



The `continue` Keyword



Using `break` and `continue`

Examples for using the `break` and `continue` keywords:

- Example 3.5: TestBreak.java

TestBreak Run

- Example 3.6: TestContinue.java

TestContinue Run

Example 3.7 Finding the Greatest Common Divisor

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.

Solution: Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8. So, how do you find the greatest common divisor? Let the two input integers be n1 and n2. You know number 1 is a common divisor, but it may not be the greatest commons divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

GreatestCommonDivisor

Example 3.8 Finding the Sales Amount

Problem: You have just started a sales job in a department store. Your pay consists of a base salary and a commission. The base salary is $5,000. The scheme shown below is used to determine the commission rate.

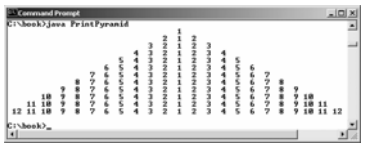
Sales Amount	Commission Rate
$0.01–$5,000	8 percent
$5,000.01–$10,000	10 percent
$10,000.01 and above	12 percent

Your goal is to earn $30,000 in a year. Write a program that will find out the minimum amount of sales you have to generate in order to make $30,000.

FindSalesAmount

Example 3.9 Displaying a Pyramid of Numbers

Problem: Write a program that prompts the user to enter an integer from 1 to 15 and displays a pyramid. For example, if the input integer is 12, the output is shown below.



PrintPyramid

Example 3.10
Displaying Prime Numbers

Problem: Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is *prime* if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

Solution: The problem can be broken into the following tasks:

- For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
- Determine whether a given number is prime.
- Count the prime numbers.
- Print each prime number, and print 10 numbers per line.

PrimeNumber