COMP20220
Programming II (Conversion)

Michael O'Mahony

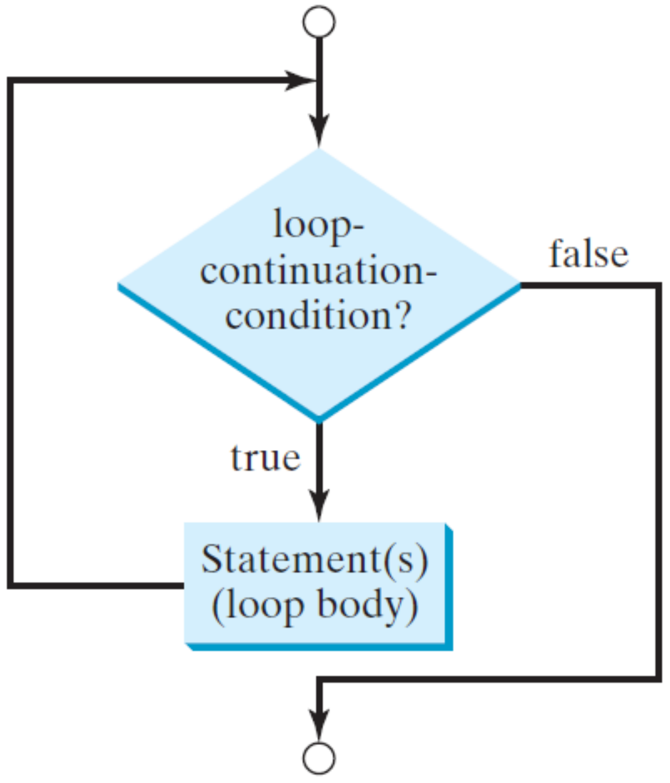
Chapter 5 Loops

Objectives

- To write programs for executing statements repeatedly using a **while** loop (§5.2).
- To control a loop with a sentinel value (§5.2.4).
- To write loops using **do-while** statements (§5.3).
- To write loops using **for** statements (§5.4).
- To discover the similarities and differences of three types of loop statements (§5.5).
- To write nested loops (§5.6).
- To implement program control with **break** and **continue** (§5.9).

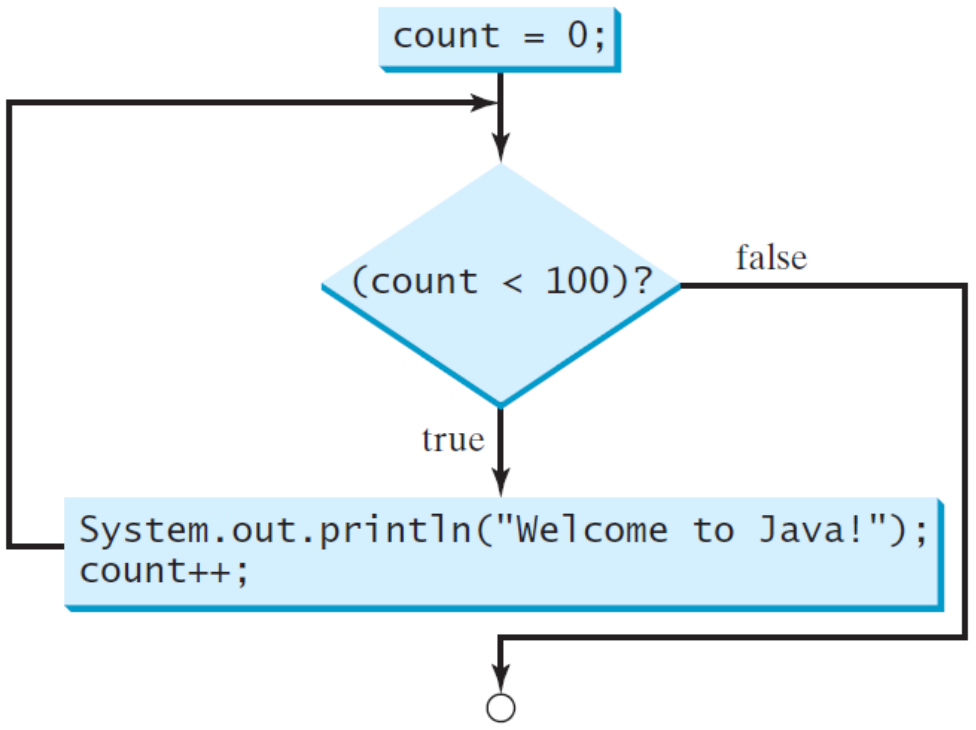
`while` Loop Flow Chart

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



`while` Loop Flow Chart

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



animation

Trace while Loop

Initialize count

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

animation

Trace while Loop, cont.

(count < 2) is true

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

animation

Trace while Loop, cont.

Print Welcome to Java

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Increase count by 1
count is now 1

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) is still true since count is 1

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Print Welcome to Java

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Increase count by 1
count is now 2

animation

Trace while Loop, cont.

(count < 2) is false since count is 2 now

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

animation

Trace while Loop

The loop exits. Execute the next statement after the loop.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Ending a Loop with a Sentinel Value

Often the number of times a loop is executed is not predetermined.

You can use an input value to signal the end of the loop. Such a value is known as a *sentinel value*.

Example – write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

SentinelValue

Caution

Do not use floating-point values for equality checking in a loop control.

Since floating-point values are approximations, using them for equality checking can result in problems…

Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1:

```
double sum = 0;
double num = 1;
while (num != 0) { // No guarantee num will ever be 0
  sum += num;
  num -= 0.1;
}
System.out.println(sum);
```

Caution

Do not use floating-point values for equality checking in a loop control.

Since floating-point values are approximations, using them can result in imprecise counter values and inaccurate results.

Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1:

```
double sum = 0;
double num = 1;
while (num != 0) { // No guarantee num will ever be 0
  sum += num;
  num -= 0.1;
}
System.out.println(sum);
```

Use: `while(num > 0)`

Problem: Guessing Numbers

Write a program that randomly generates an integer between 0 and 100, inclusive.

The program prompts the user to enter a number continuously until the number matches the randomly generated number.

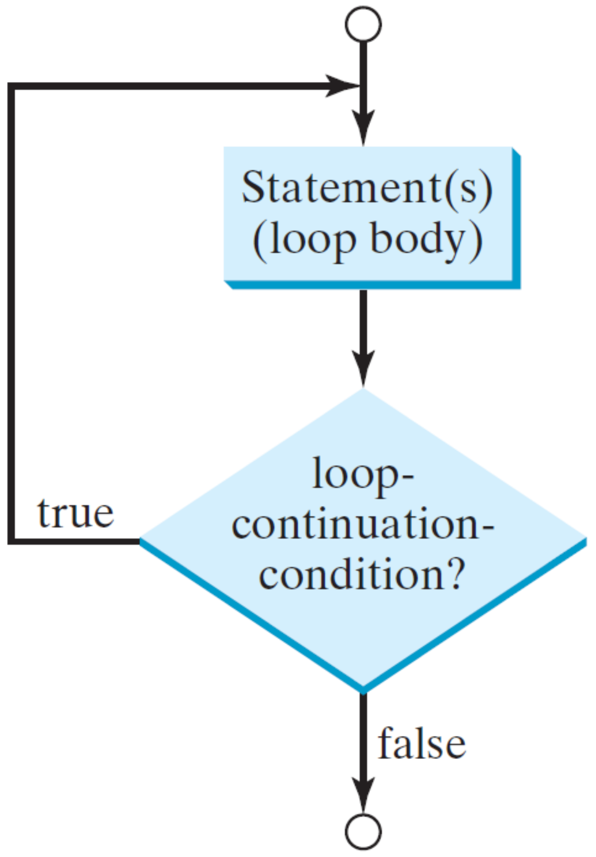
For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.

GuessNumber

`do-while` Loop

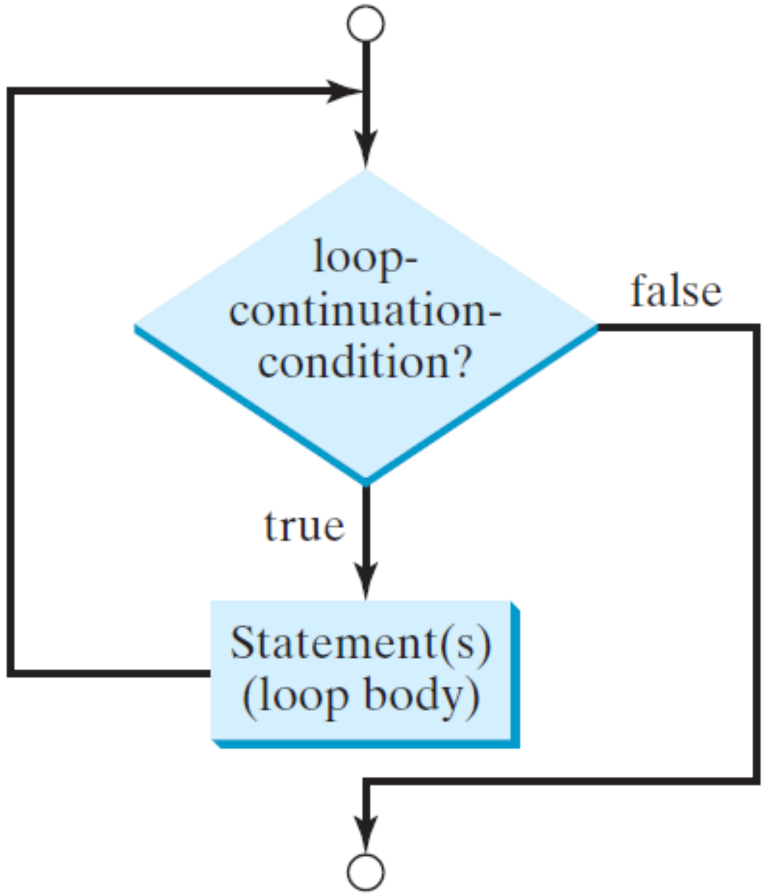
A do-while loop is the same as a while loop except that it executes the loop body first and then checks the <u>loop-continuation-condition</u> (aka <u>test</u>).

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

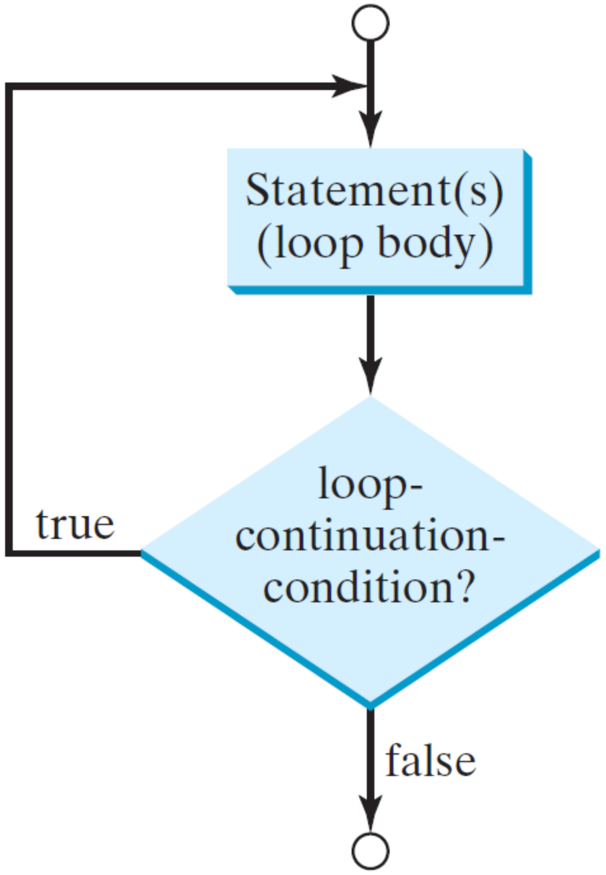


`while` vs. `do-while` Loops Flow Charts

`while` loop



`do-while` loop



Problem: Repeat Addition Until Correct

Write a program that prompts the user to add two single digits. Using a loop, let the user repeatedly enter an answer until it is correct.

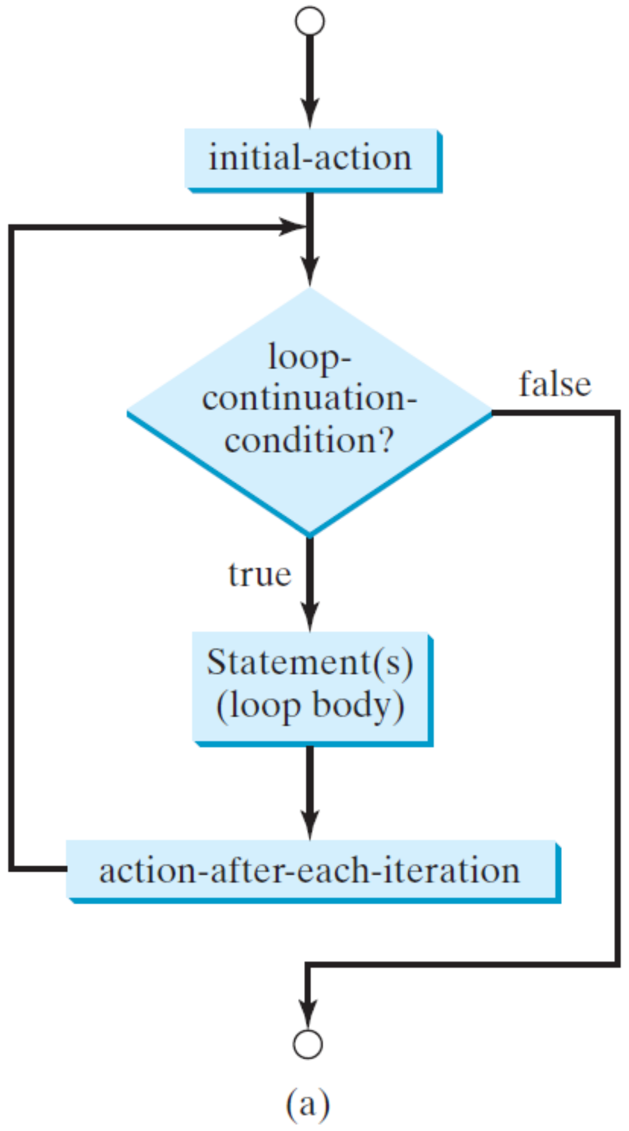
Compare solutions using `while` and `do-while` loops.

RepeatAdditionQuiz

RepeatAdditionQuiz2

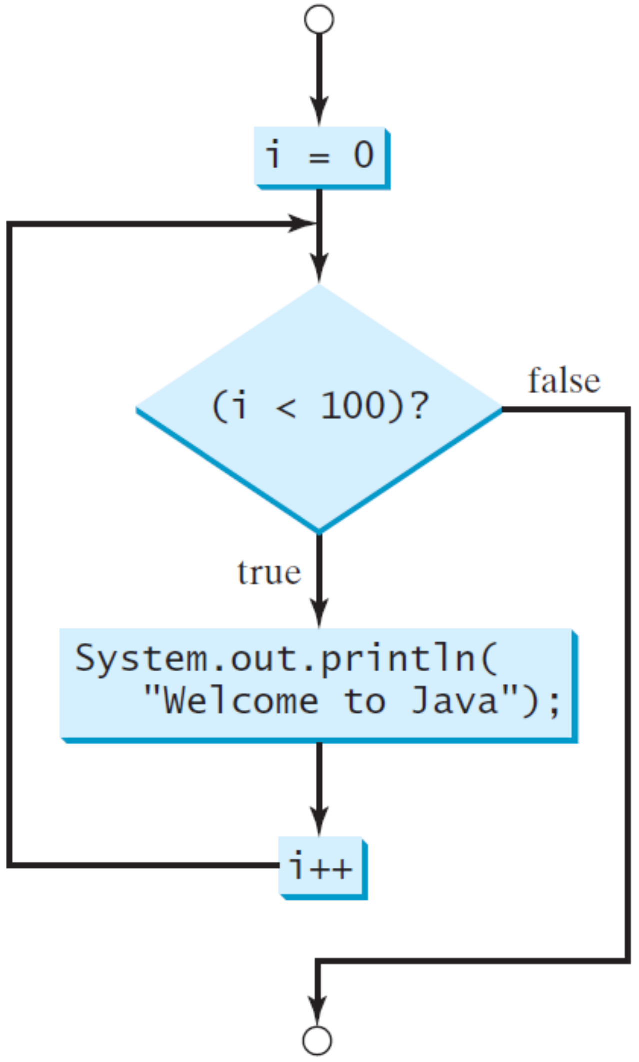
`for` Loops

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



`for` Loops

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



(b)

animation

Trace for Loop

Declare i

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

Execute initializer
i is now 0

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

(i < 2) is true
since i is 0

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println( "Welcome to Java!");
}
```

animation

Trace for Loop, cont.

Print Welcome to Java

```
int i;
for (i = 0; i < 2; i++) {
 System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

Execute adjustment statement
i is now 1

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

(i < 2) is still true
since i is 1

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

Print Welcome to Java

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

Execute adjustment statement
i is now 2

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

(i < 2) is false
since i is 2

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Exit the loop. Execute the next statement after the loop

Note

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

The <u>initial-action</u> in a `for` loop can be a list of zero or more comma-separated expressions.

The <u>action-after-each-iteration</u> in a `for` loop can be a list of zero or more comma-separated statements.

The following two `for` loops are correct:

```
for (int i = 1; i < 10; System.out.println(i++));

// prints 1, 2, … 9


for (int i = 0, j = 0; i + j < 10; i++, j++)

  System.out.println(i + " " + j);

// prints 0 0, 1 1, 2 2, 3 3, 4 4
```

Note

If the loop-continuation-condition (aka test) in a `for` loop is omitted, it is implicitly true.

The loop in (a), which is an infinite loop, is correct.

It is better to use the equivalent loop in (b) to avoid confusion.

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

(a)

Equivalent

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

(b)

Caution

Adding a semicolon at the end of the `for` clause before the loop body is a common mistake:

Logic Error

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

Caution

Adding a semicolon at the end of the `for` clause before the loop body is a common mistake:

Logic Error

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

// displays: i is 10
```

Caution, cont.

Similarly, the following `while` loop is also incorrect:

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

Logic error, infinite loop, program will not execute subsequent statements

Caution, cont.

Similarly, the following `while` loop is also incorrect:

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

Logic error, infinite loop, program will not execute subsequent statements

In the case of the `do-while` loop, a semicolon is needed to end the loop:

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

Correct

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) can always be converted into the `for` loop in (b):

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

(a)

Equivalent

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

(b)

A `for` loop in (a) can generally be converted into the following `while` loop in (b), except in certain special cases (more later):

```
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

Use the loop that is most intuitive... Generally:

- `for` loops – when the number of repetitions is known. For example, when you need to iterate over a string, iterate over an array…
- `while` loops – when the number of repetitions is not known. For example, reading numbers from the console until the input is 0.
- `do-while` loops – use instead of `while` loops if the loop body has to be executed before testing the continuation condition.

Nested Loops

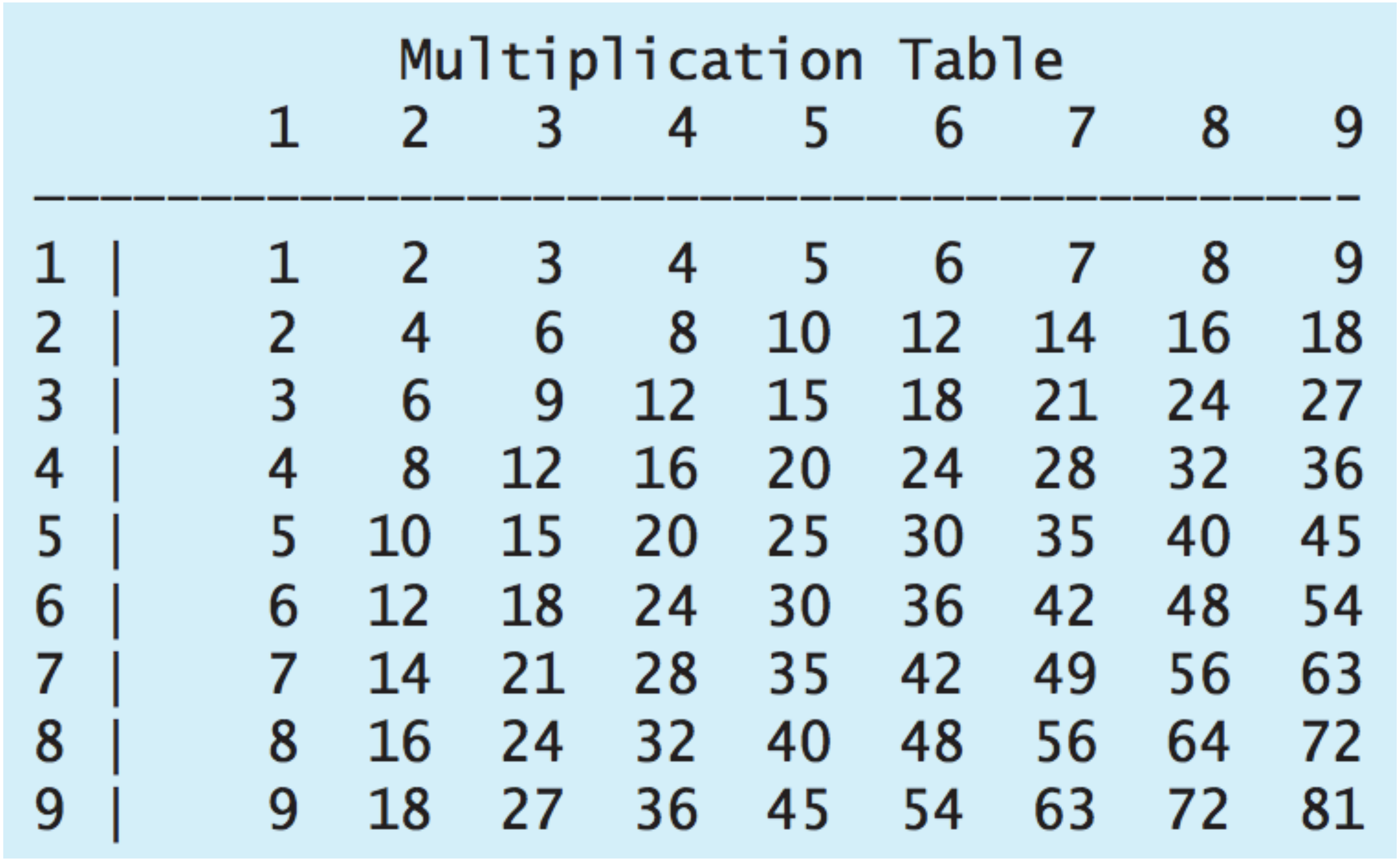
Nested loops consist of an outer loop and one or more inner loops.

Each time the outer loop is repeated, the inner loops are reentered, and started anew.

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

MultiplicationTable

Multiplication Table – Output



```
           Multiplication Table
       1   2   3   4   5   6   7   8   9
----------------------------------------
1 |    1   2   3   4   5   6   7   8   9
2 |    2   4   6   8  10  12  14  16  18
3 |    3   6   9  12  15  18  21  24  27
4 |    4   8  12  16  20  24  28  32  36
5 |    5  10  15  20  25  30  35  40  45
6 |    6  12  18  24  30  36  42  48  54
7 |    7  14  21  28  35  42  49  56  63
8 |    8  16  24  32  40  48  56  64  72
9 |    9  18  27  36  45  54  63  72  81
```

`break` and `continue`

Using the `break` and `continue` keywords in a loop:

- `break` – immediately terminates the loop.
- `continue` – ends the current iteration of the loop and program control goes to the end of the loop body.

for vs. while loops – continue

```
int sum = 0;
for (int i = 0; i < 4; i++) {
    if (i % 3 == 0)
        continue;
    sum += i;
}

System.out.println(sum);
```

`continue` – ends the current iteration of the loop and program control goes to the end of the loop body.

`for` vs. `while` loops – `continue`

```
int sum = 0;
for (int i = 0; i < 4; i++) {
    if (i % 3 == 0)
        continue;
    sum += i;
}

System.out.println(sum);
```

// prints: 3

`continue` – ends the current iteration of the loop and program control goes to the end of the loop body.

`for` vs. `while` loops – `continue`

```
int sum = 0;
for (int i = 0; i < 4; i++) {
    if (i % 3 == 0)
        continue;
    sum += i;
}

System.out.println(sum);
```

// prints: 3

```
int i = 0, sum = 0;
while (i < 4) {
    if (i % 3 == 0)
        continue;
    sum += i;
    i++;
}

System.out.println(sum);
```

// prints: ??

`continue` – ends the current iteration of the loop and program control goes to the end of the loop body.

`for` vs. `while` loops – `continue`

`while` (and `do-while`) loops:

- The loop-continuation-condition (aka test) is evaluated immediately after the `continue` statement.

`for` loop:

- The action-after-each-iteration is performed, then the loop-continuation-condition (aka test) is evaluated, immediately after the `continue` statement.

`continue` – ends the current iteration of the loop and program control goes to the end of the loop body.

`for` vs. `while` loops – `continue`

```
int i = 0, sum = 0;
while (i < 4) {
    if (i % 3 == 0)
        continue;
    sum += i;
    i++;
}

System.out.println(sum);
```

// prints: ??

```
int i = 0, sum = 0;
while (i < 4) {
    if (i % 3 == 0) {
        i++;
        continue;
    }
    sum += i;
    i++;
}

System.out.println(sum);
```

// prints: 3

`continue` – ends the current iteration of the loop and program control goes to the end of the loop body.

Next Topics…

Chapter 6

- Methods, parameters, writing and invoking methods, returning a value from a method, variable scope
- Developing reusable code that is modular, easy to read, easy to debug, and easy to maintain