Lecture 4
Chapter 5 Loops

COMP217
Java Programming
Spring 2021

Objectives

- Three basic program structures: sequence, selection and repetition
- Two types of conditions
  - Conditions using logical expressions
  - Conditions using arithmetic expressions
- This chapter focuses on repetition structures using logical conditions
  - `while` loops
  - `for` loops
  - `do…while` loops

Motivations

Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
System.out.println("Welcome to Java!");
```

So, how do you solve this problem?

Opening Problem

Problem: Print a string (e.g., "Welcome to Java!") a hundred times. How do you solve this problem?

100 times

```
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");


…

…

…
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
```

Introducing while Loops

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

```
while (condition) {
        action        // loop body
}
```

- The *condition* is called a *pretest* condition
  - pretest: “test before action”
- loop body: the code between the braces
- 1 iteration: a single execution of the loop body

Flowchart for a `while` Loop

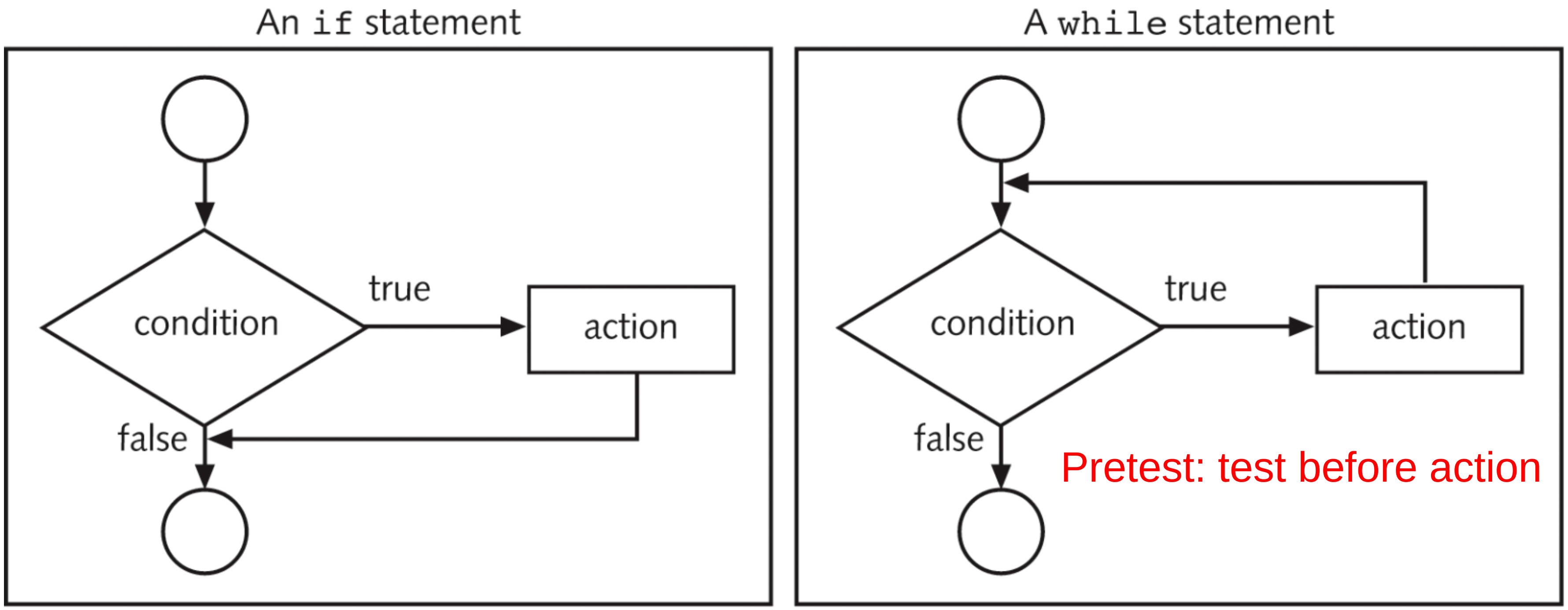


Figure 4.3 Flowchart symbols for the `if` statement (left) and the `while` statement (right)

LoopExample1.java

```
public class LoopExample1 {
    public static void main(String[] args) {
      int j=0;
      while (j < 5) {
            System.out.println(j + " ");
            j++;
      }
   }
}

/* result:
0 1 2 3 4
*/  // [Q] Why not 0 1 2 3 4 5 ?
```

LoopExample2.java

```java
public class LoopExample2 {
  public static void main(String[] args) {
    int n, i=1;
    Scanner scan = new Scanner(System.in);
    System.out.print("Multiplier?  ");
    n = scan.nextInt();
    while (i <= 9) {
      System.out.println(n + "*" + i + "=" + n*i);
      i++;
    }
  }
}
```

```
$ Java LoopExample2
Multiplier? 8
8*1=8
8*2=16
8*3=24
…
```

GCD (Greatest Common Divisor)

- Euclid algorithm
  - Input: two integers x and y

  (1) For x ≥ y

  (2) if x *mod* y is zero, y is the gcd

  (3) otherwise, the gcd of x and y is the gcd of y and x *mod* y

```java
import java.util.Scanner;

public class Gcd {
  public static void main(String[] args) {
    int x, y, r;
    Scanner scan = new Scanner(System.in);
    System.out.print("Enter two integers: ");
    x = scan.nextInt();
    y = scan.nextInt();

    if ( x < y ) {
      /* swap x and y to satisfy x>=y */
      r = x; x = y; y = r;
    }
    while (y != 0) {
      r = x % y;
      x = y;
      y = r;
    }
    System.out.println(
        "The greatest common divisor is "
        + x);
  }
}

/*
$ javac Gcd.java
$ java Gcd
Enter two integers: 240 36
The greatest common divisor is 12
*/
```

Caution

- Don't use floating-point values for equality checking in a loop control.
  - Floating-point values are approximations for real values
    - Ex.) 1.7 may be stored as 1.699999999999

Not recommended	Good

```
double item = 1; double sum = 0;
while (item != 0) {
  // No guarantee item will be 0
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```
	```
	int counter = 10;
	double step = 0.1; double sum = 0;
	while (counter != 0) {
	  // Equality test on integers
	  sum += step * (double) counter;
	  counter -= 1;
	}
	System.out.println(sum);
	```

`do`...`while` Loop

```
do{
        action
} while (condition);
```

- The loop body must execute <u>at least once</u>
- *posttest*: After the first execution of the loop body, the condition is tested
  - In the case of while loop, the loop body may never executes, because if the condition is false in the beginning, the loop ends

The `do...while` Flowchart

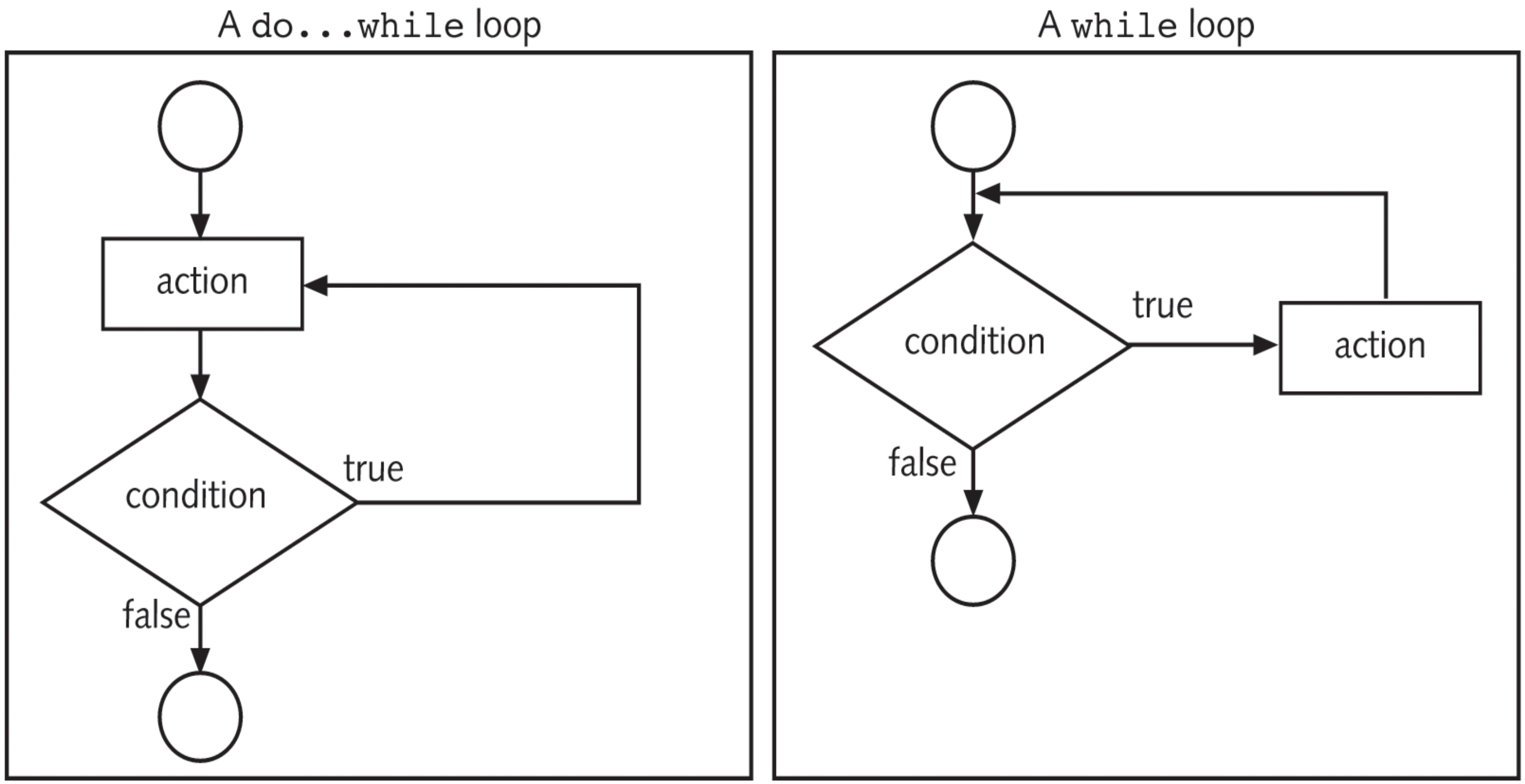


Figure 4.16 Flowcharts for a `do...while` loop (left) and a `while` loop (right)

LoopExample3.java

```java
import java.util.Scanner;

public class LoopExample3 {
  public static void main(String[] args) {
    int i;
    Scanner sc = new Scanner(System.in);
    System.out.print("Which number?  ");
    i = sc.nextInt();
    do {
      System.out.println("i = " + i);
      i++;
    }
    while (i < 9);
  }
}
```

```
/*
$ javac LoopExample3.java
$ java LoopExample3
Which number? 3
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
$ java LoopExample3
Which number? 10
i = 10
$ java LoopExample3
Which number? 20
i = 20
*/
```

LetterGame.java

```
import java.util.Scanner;

public class LetterGame {
  public static void main(String[] args) {
    int answer = 7;
    int guess;
    int tries = 0;
    Scanner scan = new Scanner(System.in);

    // ask at least once
    do {
      System.out.print("Your guess? ");
      guess = scan.nextInt();
      tries++;

      if ( guess > answer )
        System.out.println("Larger, try again.");
      else if ( guess < answer )
        System.out.println("Smaller, try again.");
    }
    while ( guess != answer );

    System.out.println("Matched in " + tries + " try(ies).");
  }
}
```

```
/*
$ javac LetterGame.java
$ java LetterGame
Your guess? 5
Smaller, try again.
Your guess? 10
Larger, try again.
Your guess? 7
Matched in 3 try(ies).
*/
```

Checkpoints

1. What is the output of the following code?

```
int n = 10;
while (n > 0) {
      System.out.println(n);
      n = n - 3;
}
```

2. Change the above code using a do-while loop

`for` Loop

- The `while` loop is the most general repetition structure
- The `for` loop is logically equivalent to the counter-controlled `while` loop

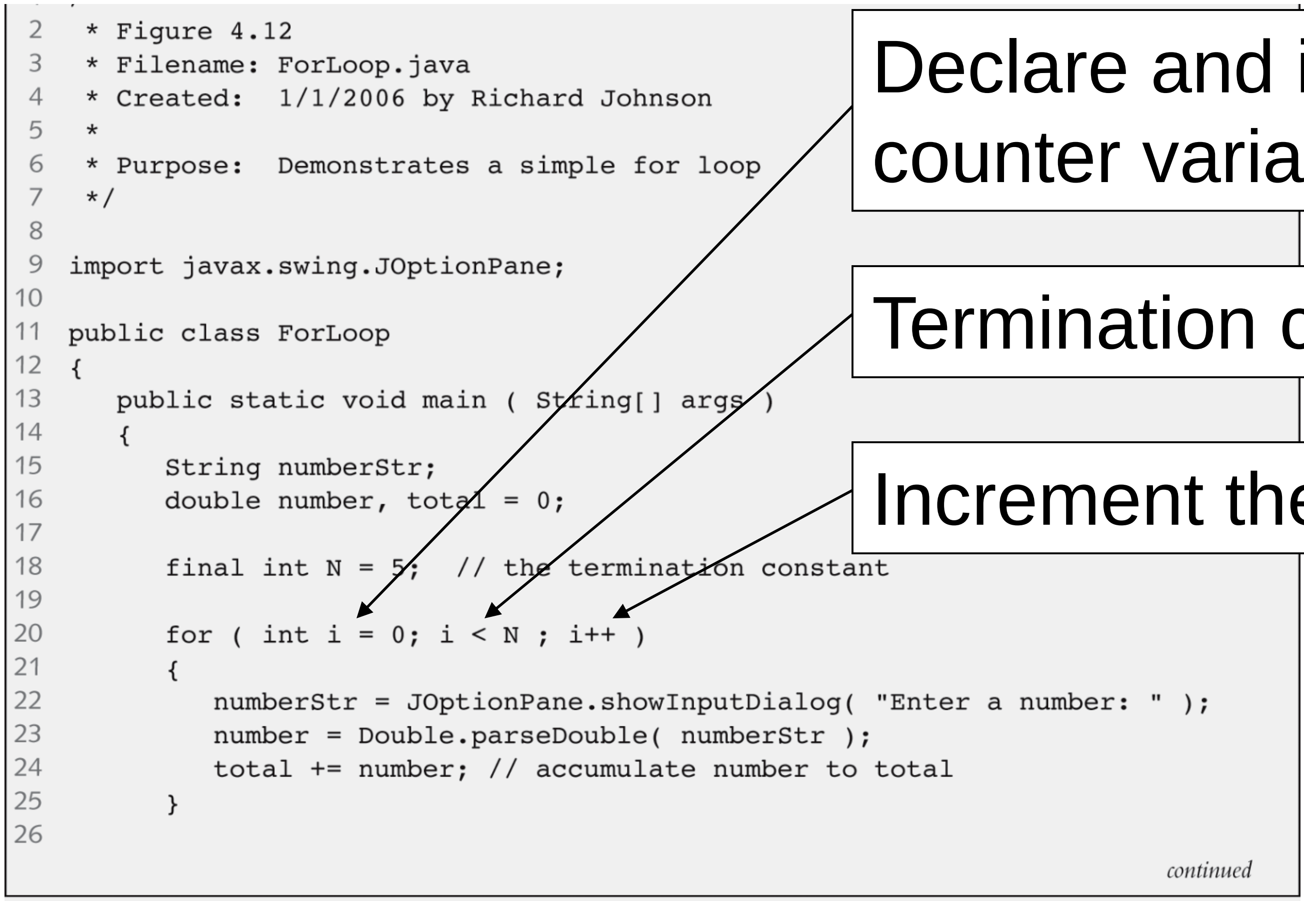
Example 4.1

```
int i = 1;
while ( i <= 10 )
{
    System.out.println( i );
    i++;
}
```

Example 4.2

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

The `for` Statement’s Syntax



```
 * Figure 4.12
 * Filename: ForLoop.java
 * Created:  1/1/2006 by Richard Johnson
 *
 * Purpose:  Demonstrates a simple for loop
 */

import javax.swing.JOptionPane;

public class ForLoop
{
   public static void main ( String[] args )
   {
      String numberStr;
      double number, total = 0;

      final int N = 5;   // the termination constant

      for ( int i = 0; i < N ; i++ )
      {
         numberStr = JOptionPane.showInputDialog( "Enter a number: " );
         number = Double.parseDouble( numberStr );
         total += number; // accumulate number to total
      }

```

continued

Figure 4.12 Program code for ForLoop.java

Declare and initialize the counter variable

Termination condition

Increment the counter

Sum.java

```java
import java.util.Scanner;
public class Sum {
  public static void main(String[] args) {
    int sum = 0, n;
    Scanner sc = new Scanner(System.in);

    System.out.print("n?  ");
    n = sc.nextInt();
    for (int i=1; i<=n; i++)
      sum += i;

    System.out.println("Sum from 1 to " + n + " = " + sum);
  }
}

/*
mico:week5$ javac Sum.java
mico:week5$ java Sum
n? 10
Sum from 1 to 10 = 55
*/
```

Factorial.java

```java
/* Class Factorial computues
 n! = n(n-1)(n-2)...1 */

import java.util.Scanner;

public class Factorial {
  public static void main(String[] args) {
    long fac;   // long: factorial is very large
    long pre_fac;       // to check overflow
    int i, n;
    Scanner sc = new Scanner(System.in);

    System.out.print("n? ");
    n = sc.nextInt();

    // start from fac = 0! = 1
    for (i=1, fac=1L; i<=n; i++) {
      pre_fac = fac;
      fac *= i;

      // check if overflowed
      if ( pre_fac != fac / i ) {
        System.out.println("Overflowed at " + i + "! = " + fac);
        fac = pre_fac;  // roll back to the previous, unoverflowed
        break;
      }
    }

    // [Q] Why (i-1)?
    System.out.println((i-1) + "! = " + fac);
  }
}
```

```
/*
mico:week5$ javac Factorial.java
mico:week5$ java Factorial
n? 10
10! = 3628800
mico:week5$ java Factorial
n? 20
20! = 2432902008176640000
mico:week5$ java Factorial
n? 30
Overflowed at 21! = -4249290049419214848
20! = 2432902008176640000
*/
```

Note

Multiple initial-actions and **multiple** action-after-each-iterations in a for loop and are allowed with comma (,) separator.

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

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

If the loop-continuation-condition in a for loop is omitted, it is implicitly **true**.

```
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, as shown below:

Logic Error

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

Caution, cont.

Similarly, the following loop is also wrong:

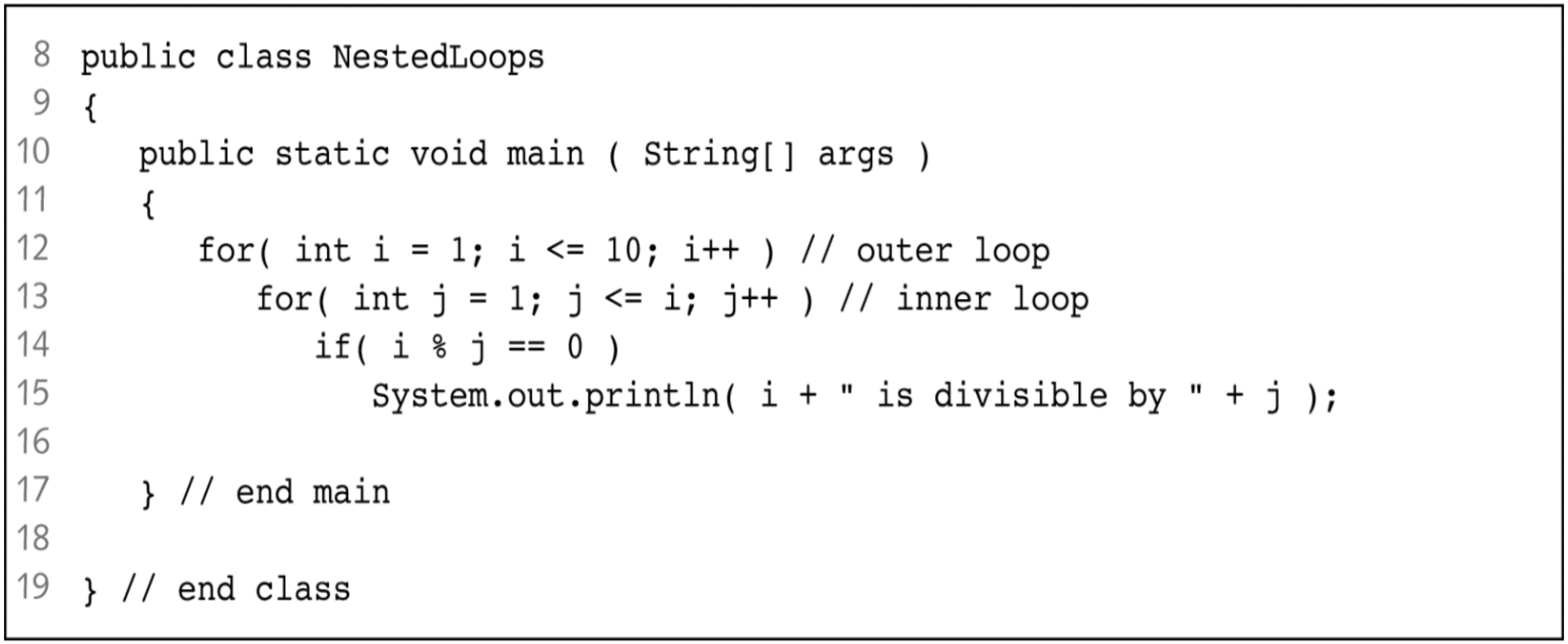
```
int i=0;
while (i < 10);        Logic Error
{
  System.out.println("i is " + i);
  i++;
}
```

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
```

Nested Loops

- A nested loop is a loop within a loop
  - Any type and any number of loops can be nested



```java
public class NestedLoops
{
   public static void main ( String[] args )
   {
      for( int i = 1; i <= 10; i++ ) // outer loop
         for( int j = 1; j <= i; j++ ) // inner loop
            if( i % j == 0 )
               System.out.println( i + " is divisible by " + j );

   } // end main

} // end class
```

Figure 4.26 Program code for NestedLoops.java (continued)

NestedLoop2.java

```java
import java.util.Scanner;

public class NestedLoop2 {
  public static void main(String[] args) {
    int n;
    Scanner sc = new Scanner(System.in);

    System.out.print("How many lines? ");
    n = sc.nextInt();

    // can delare variable inside for loop
    for (int y=1; y<=n; y++) {
      for (int x=1; x<=y; x++) {
        System.out.print("*");
      }
      System.out.println("");   // change line
    }
  }
}
```

```
/*
mico:week5$ javac NestedLoop2.java
mico:week5$ java NestedLoop2
How many lines? 5
*
**
***
****
*****
mico:week5$ java NestedLoop2
How many lines? 10
*
**
***
****
*****
******
*******
********
*********
**********
*/
```

The `break` Statement in Loops

- Recall that `break` was used to exit early from a `switch` structure
  - The `break` statement in a `switch` statement ensured that the `switch` exited after a particular case was executed
- A `break` statement in a loop causes that particular loop to terminate
- A `break` statement can be used in any kind of loop (for, while, do…while)

break

```
public class TestBreak {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      sum += number;
      if (sum >= 100)
        break;
    }

    System.out.println("The number is " + number);
    System.out.println("The sum is " + sum);
  }
}
```

BreakTest.java

```java
import java.util.Scanner;

public class BreakTest {
  public static void main(String[] args) {
    int total = 0, count = 0;    // initialize when declared
    Scanner sc = new Scanner(System.in);

    // This example shows exiting a loop not by counting
    while ( true ) {
      int score;          // can declare inside
      System.out.print("Your score? (negative number when done) ");
      score = sc.nextInt();
      if ( score < 0 )
        break;  // (**) get out of the loop
      total += score;
      count++;
    }

    // (**) break jumps here
    // variable count is to compute average
    System.out.printf("Average score is %.2f\n",
        (double)total/(double)count);
  }
}
```

```
/*
mico:week5$ javac BreakTest.java
mico:week5$ java BreakTest
Your score? (negative number when done) 3
Your score? (negative number when done) 4
Your score? (negative number when done) 5
Your score? (negative number when done) 0
Your score? (negative number when done) -1
Average score is 3.00
*/
```

continue

- The `continue` statement causes the current loop iteration to be skipped

```
public class TestContinue {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      if (number == 10 || number == 11)
        continue;
      sum += number;
    }

    System.out.println("The sum is " + sum);
  }
}
```

ContinueTest.java

```
public class ContinueTest {
  public static void main(String[] args) {
    String s = "no news is good news";
    int n = 0;

    for (int i=0; i<s.length(); i++) {
      // count number of 'n' appearances
      if (s.charAt(i) != 'n')
        continue;

      // count
      n++;
    }
    System.out.println("Number of n's appearances = " + n);
  }
}

/*
mico:week5$ javac ContinueTest.java
mico:week5$ java ContinueTest
Number of n's appearances = 3
*/
```

A Caveat About `break` and `continue`

- Some programmers prefer to avoid `break` and `continue` because they make the program harder to understand
- Some programmers feel `break` and `continue` are useful in some situations
- Recommendation
  - Try not to use `break` and `continue`,
  - Write a loop to avoid them

Checkpoints

1. What is the output of the following program?

```
int n = 12;
while (n > 0) {
    n = n - 2;
    if( n == 6 ) break;
    System.out.println(n);
}
```

2. What is the output if we replace break with continue?