2

Control Statements

Muhammed S. Anwar

OBJECTIVES

In this chapter you will learn:

- To use basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` and `if…else` selection statements to choose among alternative actions.
- To use the `while` repetition statement to execute statements in a program repeatedly.
- To use counter-controlled repetition and sentinel-controlled repetition.
- To use the assignment, increment and decrement operators.
- The primitive data types.

Outline

Outline

4.2 Algorithms

- **Algorithms**
  - **The actions to execute**
  - **The order in which these actions execute**
- **Program control**
  - **Specifies the order in which actions execute in a program**

4.3 Pseudocode

- **Pseudocode**
  - **An informal language similar to English**
  - **Helps programmers develop algorithms**
  - **Does not run on computers**
  - **Should contain input, output and calculation actions**
  - **Should not contain variable declarations**

4.4 Control Structures

- **Sequence structure**
  - **"built-in" to Java**
- **Selection structure**
  - **`if`, `if…else` and `switch`**
- **Repetition structure**
  - **`while`, `do…while` and `for`**

4.4 Control Structures (Cont.)

- **UML activity diagram (www.uml.org)**
  - **Models the workflow (or activity) of a part of a software system**
  - **Action-state symbols (rectangles with their sides replaced with outward-curving arcs)**
    - **represent action expressions specifying actions to perform**
  - **Diamonds**
    - **Decision symbols (explained in Section 4.5)**
    - **Merge symbols (explained in Section 4.7)**

4.4 Control Structures (Cont.)

- **Small circles**
  - **Solid circle represents the activity's initial state**
  - **Solid circle surrounded by a hollow circle represents the activity's final state**
- **Transition arrows**
  - **Indicate the order in which actions are performed**
- **Notes (rectangles with the upper-right corners folded over)**
  - **Explain the purposes of symbols (like comments in Java)**
  - **Are connected to the symbols they describe by dotted lines**

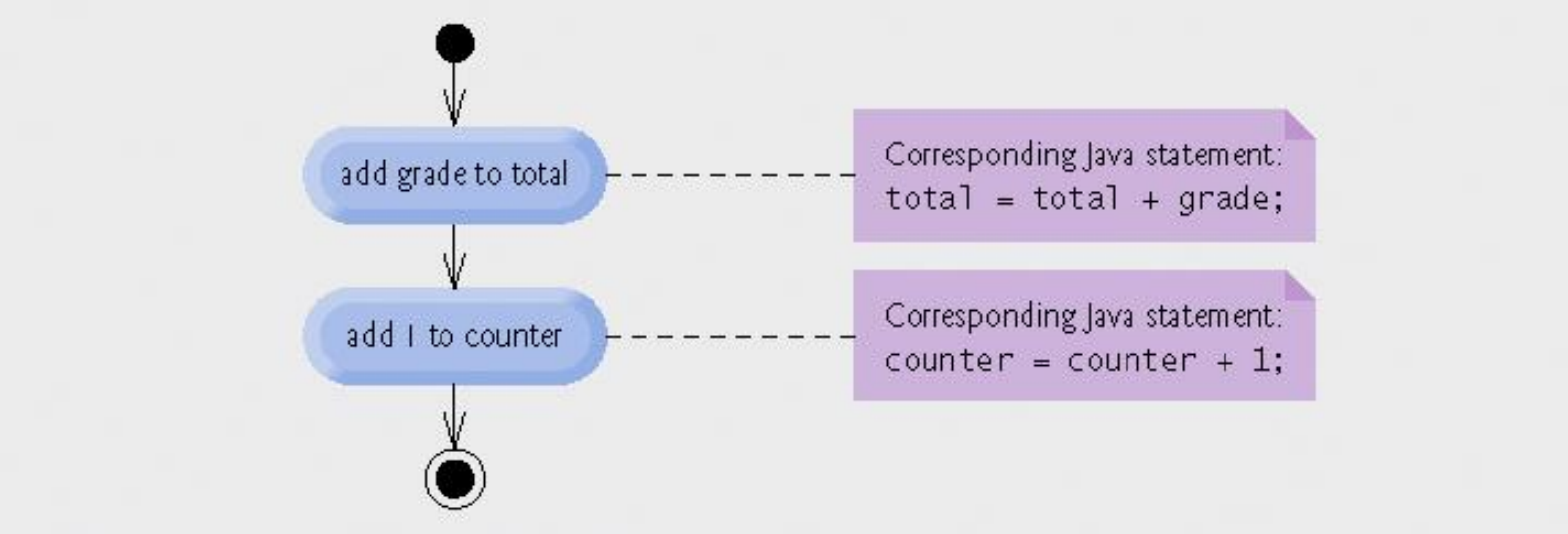


Fig. 4.1 | Sequence structure activity diagram.

4.4 Control Structures (Cont.)

- **Selection Statements**
  - **`if` statement**
    - **Single-selection statement**
  - **`if`...`else` statement**
    - **Double-selection statement**
  - **`switch` statement**
    - **Multiple-selection statement**

4.4 Control Structures (Cont.)

- **Repetition statements**
  - **Also known as looping statements**
  - **Repeatedly performs an action while its loop-continuation condition remains true**
  - **`while` statement**
    - **Performs the actions in its body zero or more times**
  - **`do`...`while` statement**
    - **Performs the actions in its body one or more times**
  - **`for` statement**
    - **Performs the actions in its body zero or more times**

4.4 Control Structures (Cont.)

- **Java has three kinds of control structures**
  - **Sequence statement,**
  - **Selection statements (three types) and**
  - **Repetition statements (three types)**
  - **All programs are composed of these control statements**
    - **Control-statement stacking**
      - **All control statements are single-entry/single-exit**
    - **Control-statement nesting**

4.5 `if` Single-Selection Statement

- **`if` statements**
  - **Execute an action if the specified condition is `true`**
  - **Can be represented by a decision symbol (diamond) in a UML activity diagram**
    - **Transition arrows out of a decision symbol have guard conditions**
      - **Workflow follows the transition arrow whose guard condition is true**

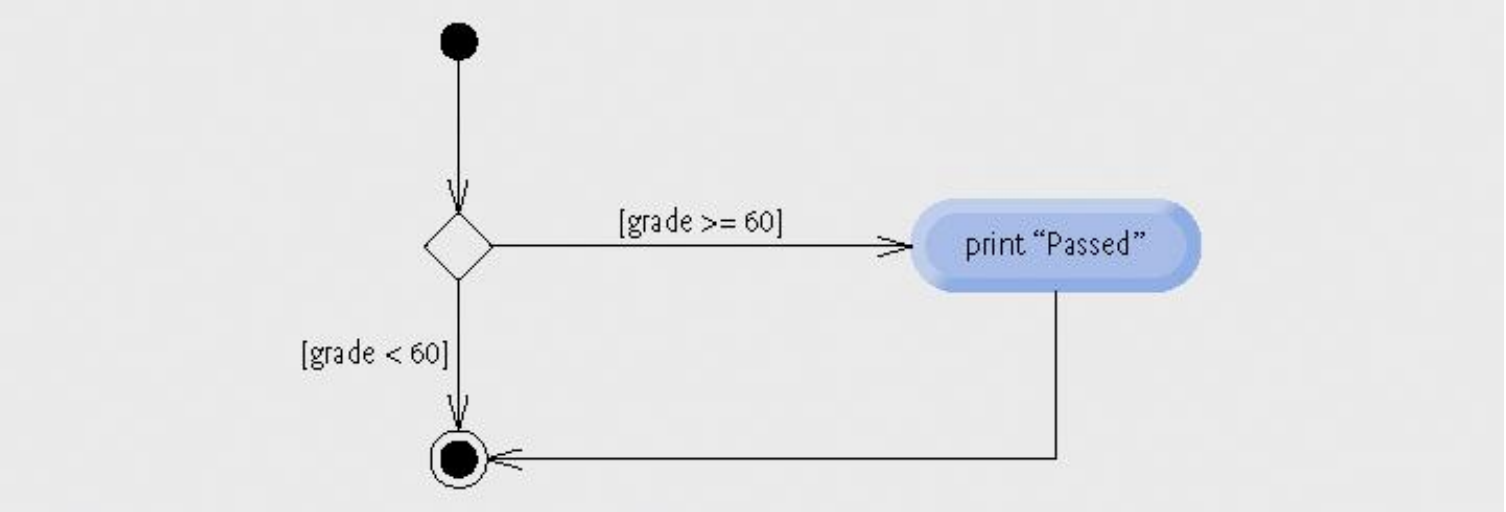


Fig. 4.2 | `if` single-selection statement UML activity diagram.

4.6 `if…else` Double-Selection Statement

- **`if…else` statement**
  - **Executes one action if the specified condition is `true` or a different action if the specified condition is `false`**
- **Conditional Operator (`? :`)**
  - **Java's only ternary operator (takes three operands)**
  - **`? :` and its three operands form a conditional expression**
    - **Entire conditional expression evaluates to the second operand if the first operand is `true`**
    - **Entire conditional expression evaluates to the third operand if the first operand is `false`**

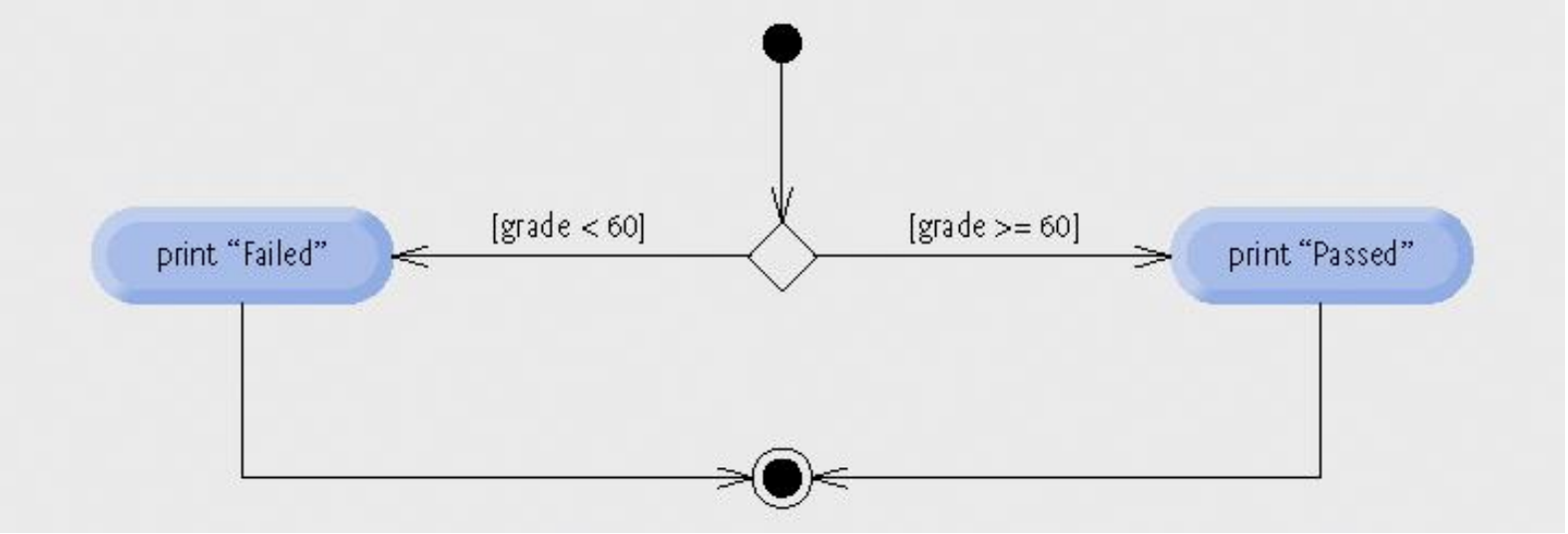


Fig. 4.3 | `if`…else double-selection statement UML activity diagram.

4.6 `if`…`else` Double-Selection Statement (Cont.)

- **Nested `if`…`else` statements**
  - **`if`…`else` statements can be put inside other `if`…`else` statements**
- **Blocks**
  - **Braces `{ }` associate statements into blocks**
  - **Blocks can replace individual statements as an `if` body**

4.7 `while` Repetition Statement

- **`while` statement**
  - **Repeats an action while its loop-continuation condition remains true**
  - **Uses a merge symbol in its UML activity diagram**
    - **Merges two or more workflows**
    - **Represented by a diamond (like decision symbols) but has:**
      - **Multiple incoming transition arrows,**
      - **Only one outgoing transition arrow and**
      - **No guard conditions on any transition arrows**

Common Programming Error 4.3

Not providing, in the body of a `while` statement, an action that eventually causes the condition in the `while` to become false normally results in a logic error called an *infinite loop*, in which the loop never terminates.

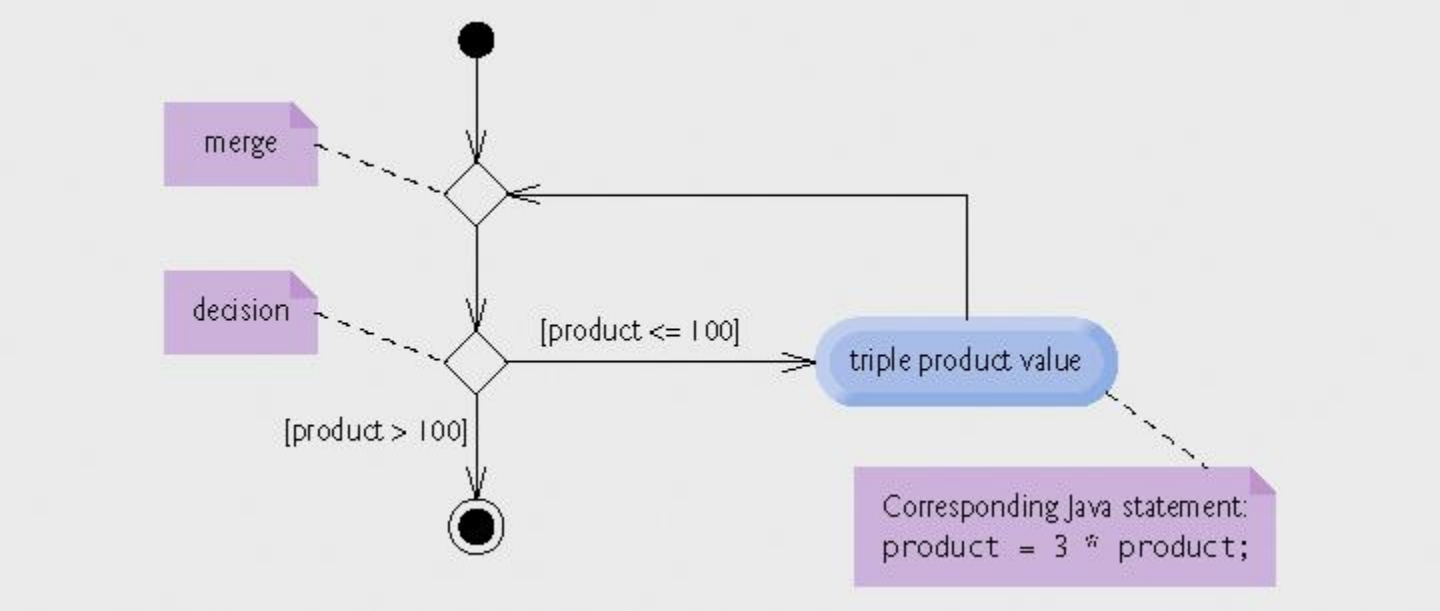


Fig. 4.4 | `while` repetition statement UML activity diagram.

4.8 Formulating Algorithms: Counter-Controlled Repetition

- **Counter-controlled repetition**
  - **Use a counter variable to count the number of times a loop is iterated**
- **Integer division**
  - **The fractional part of an integer division calculation is truncated (thrown away)**

```
1     Set total to zero
2     Set grade counter to one
3
4     While grade counter is less than or equal to ten
5         Prompt the user to enter the next grade
6         Input the next grade
7         Add the grade into the total
8         Add one to the grade counter
9
10    Set the class average to the total divided by ten
11    Print the class average
```

Fig. 4.5 | Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

```java
import java.util.Scanner;

public class AverageCalculator
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner( System.in );

        int total; // sum of grades entered by user
        int gradeCounter; // number of the grade to be entered next
        int grade; // grade value entered by user
        int average; // average of grades

        total = 0; // initialize total
        gradeCounter = 1; // initialize loop counter

        while ( gradeCounter <= 5 ) // loop 10 times
        {
           System.out.print( "Enter grade: " ); // prompt
           grade = input.nextInt(); // read grade from user
           total = total + grade; // add grade to total
           gradeCounter = gradeCounter + 1; // increment counter by 1
        } // end while

        average = total / 5; // integer division yields integer result

        System.out.printf( "\nTotal of all 10 grades is %d\n", total );
        System.out.printf( "Class average is %d\n", average );
    }
}
```

```
Enter grade: 76
Enter grade: 86
Enter grade: 92
Enter grade: 56
Enter grade: 66

Total of all 10 grades is 376
Class average is 75
```

4.9 Formulating Algorithms: Sentinel-Controlled Repetition

- **Sentinel-controlled repetition**
  - **Also known as indefinite repetition**
  - **Use a sentinel value (also known as a signal, dummy or flag value)**
    - **A sentinel value cannot also be a valid input value**

```
1     Initialize total to zero
2     Initialize counter to zero
3
4     Prompt the user to enter the first grade
5     Input the first grade (possibly the sentinel)
6
7     While the user has not yet entered the sentinel
8         Add this grade into the running total
9         Add one to the grade counter
10        Prompt the user to enter the next grade
11        Input the next grade (possibly the sentinel)
12
13    If the counter is not equal to zero
14        Set the average to the total divided by the counter
15        Print the average
16    else
17        Print "No grades were entered"
```

Fig. 4.8 | Class-average problem pseudocode algorithm with sentinel-controlled repetition.

```java
Scanner input = new Scanner(System.in);

int total; // sum of grades
int gradeCounter; // number of grades entered
int grade; // grade value
double average; // number with decimal point for average

total = 0; // initialize total
gradeCounter = 0; // initialize loop counter

System.out.print("Enter grade or -1 to quit: ");
grade = input.nextInt();

// loop until sentinel value read from user
while (grade != -1) {
    total = total + grade; // add grade to total
    gradeCounter = gradeCounter + 1; // increment counter

    System.out.print("Enter grade or -1 to quit: ");
    grade = input.nextInt();
} // end while

if (gradeCounter != 0) {
    average = (double) total / gradeCounter;

    System.out.printf("\nTotal of the %d grades entered is %d\n", gradeCounter, total);
    System.out.printf("Class average is %.2f\n", average);
} // end if
else
    System.out.println("No grades were entered");
```

```
Enter grade or -1 to quit: 58
Enter grade or -1 to quit: 75
Enter grade or -1 to quit: 86
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: -1

Total of the 4 grades entered is 307
Class average is 76.75
```

```
Enter grade or -1 to quit: 55
Enter grade or -1 to quit: 65
Enter grade or -1 to quit: 75
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 195
Class average is 65.00
```

```
Enter grade or -1 to quit: -1
No grades were entered
```

4.10 Formulating Algorithms: Nested Control Statements

- **Control statements can be nested within one another**
  - **Place one control statement inside the body of the other**

4.11 Compound Assignment Operators

- **Compound assignment operators**
  - **An assignment statement of the form:**
    ***variable* = *variable operator expression*;**
    where *operator* is +, -, *, / or % can be written as:
    ***variable operator*= *expression*;**
  - **example: `c = c + 3;` can be written as `c += 3;`**
    - **This statement adds 3 to the value in variable `c` and stores the result in variable `c`**

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Fig. 4.14 | Arithmetic compound assignment operators.

4.12 Increment and Decrement Operators

- **Unary increment and decrement operators**
  - **Unary increment operator (++) adds one to its operand**
  - **Unary decrement operator (−−) subtracts one from its operand**
  - **Prefix increment (and decrement) operator**
    - **Changes the value of its operand, then uses the new value of the operand in the expression in which the operation appears**
  - **Postfix increment (and decrement) operator**
    - **Uses the current value of its operand in the expression in which the operation appears, then changes the value of the operand**

```
1  // Fig. 4.16: Increment.java
2  // Prefix increment and postfix increment operators.
3
4  public class Increment
5  {
6     public static void main( String args[] )
7     {
8        int c;
9
10       // demonstrate postfix increment operator
11       c = 5; // assign 5 to c
12       System.out.println( c );   // print 5
13       System.out.println( c++ ); // print 5 then postincrement
14       System.out.println( c );   // print 6
15
16       System.out.println(); // skip a line
17
18       // demonstrate prefix increment operator
19       c = 5; // assign 5 to c
20       System.out.println( c );   // print 5
21       System.out.println( ++c ); // preincrement then print 6
22       System.out.println( c );   // print 6
23
24    } // end main
25
26 } // end class Increment
```

Postincrementing the `c` variable

Preincrementing the `c` variable

```
5
5
6

5
6
6
```

4.13 Primitive Types

- **Java is a strongly typed language**
  - **All variables have a type**
- **Primitive types in Java are portable across all platforms that support Java**

5.1 Introduction

- **Continue structured-programming discussion**
  - **Introduce Java's remaining control structures**
    - `for`, `do...while`, `switch`

5.2 Essentials of Counter-Controlled Repetition

- **Counter-controlled repetition requires:**
  - **Control variable (loop counter)**
  - **Initial value of the control variable**
  - **Increment/decrement of control variable through each loop**
  - **Loop-continuation condition that tests for the final value of the control variable**

```java
// Fig. 5.1: WhileCounter.java
// Counter-controlled repetition with the while repetition statement.

public class WhileCounter
{
   public static void main( String args[] )
   {
      int counter = 1; // declare and initialize control variable

      while ( counter <= 10 ) // loop-continuation condition
      {
         System.out.printf( "%d  ", counter );
         ++counter; // increment control variable by 1
      } // end while

      System.out.println(); // output a newline
   } // end main
} // end class WhileCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

5.3 for Repetition Statement

- **Handles counter-controlled-repetition details**

```
1  // Fig. 5.2: ForCounter.java
2  // Counter-controlled repetition with the for repetition statement.
3
4  public class ForCounter
5  {
6     public static void main( String args[] )
7     {
8        // for statement header includes initialization,
9        // loop-continuation condition and increment
10       for ( int counter = 1; counter <= 10; counter++ )
11          System.out.printf( "%d  ", counter );
12
13       System.out.println(); // output a newline
14    } // end main
15 } // end class ForCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

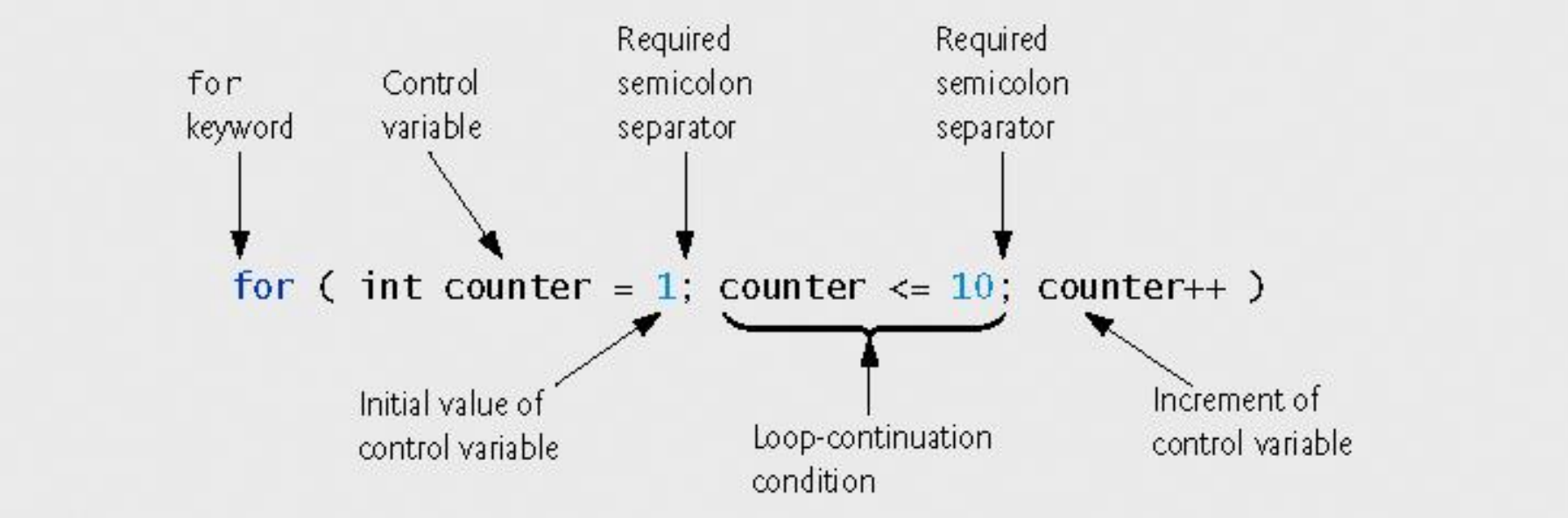


Fig. 5.3 | `for` **statement header components.**

5.3 `for` Repetition Statement (Cont.)

`for` (*initialization*; *loopContinuationCondition*; *increment*)
    statement;

can usually be rewritten as:

initialization;
`while` (*loopContinuationCondition*)
{
    statement;
    increment;
}

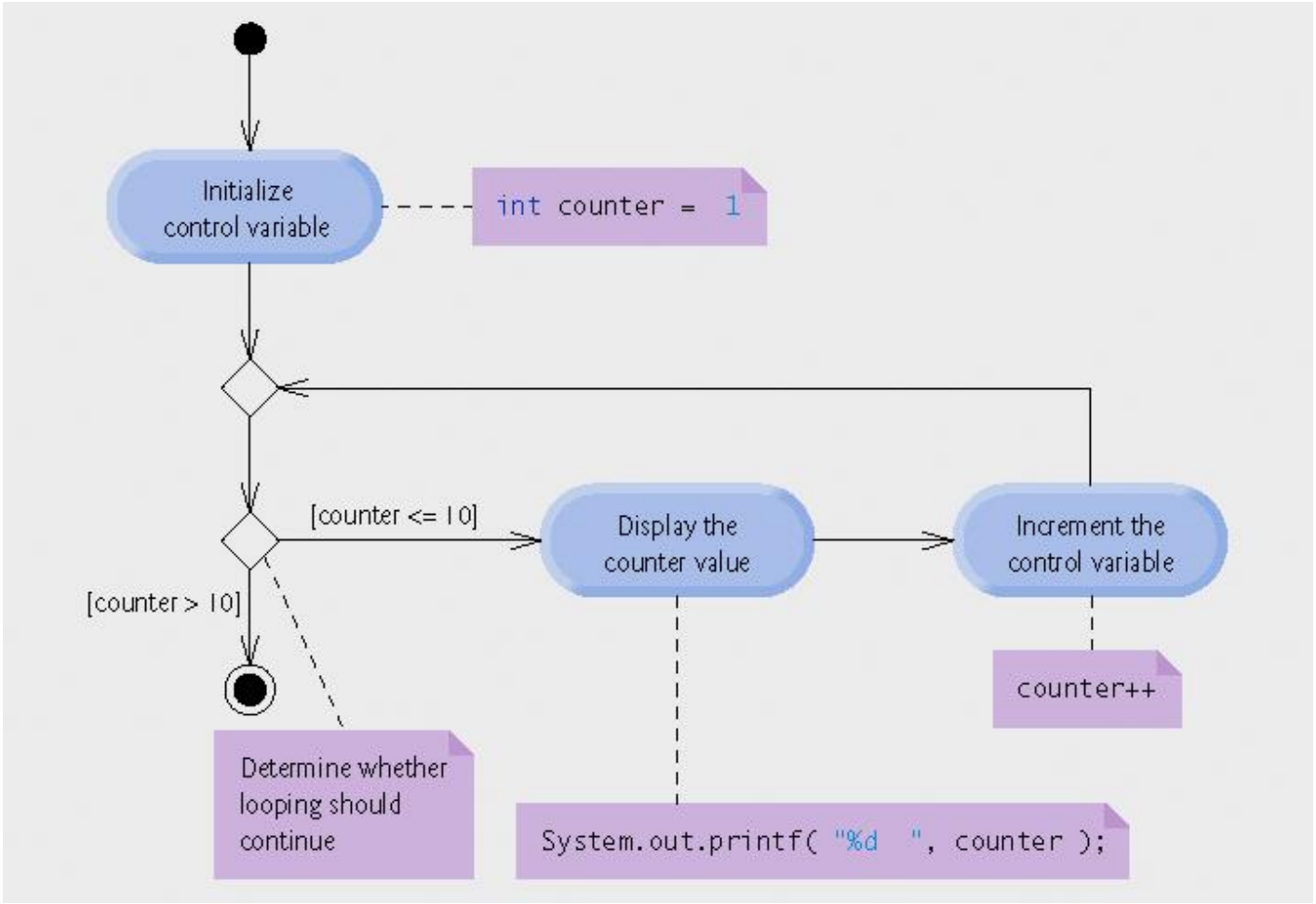


Fig. 5.4 | UML activity diagram for the for statement in Fig. 5.2.

5.4 Examples Using the `for` Statement

- **Varying control variable in `for` statement**
  - Vary control variable from 1 to 100 in increments of 1
    - `for ( int i = 1; i <= 100; i++ )`
  - Vary control variable from 100 to 1 in increments of –1
    - `for ( int i = 100; i >= 1; i-- )`
  - Vary control variable from 7 to 77 in increments of 7
    - `for ( int i = 7; i <= 77; i += 7 )`
  - Vary control variable from 20 to 2 in decrements of 2
    - `for ( int i = 20; i >= 2; i -= 2 )`
  - Vary control variable over the sequence: 2, 5, 8, 11, 14, 17, 20
    - `for ( int i = 2; i <= 20; i += 3 )`
  - Vary control variable over the sequence: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0
    - `for ( int i = 99; i >= 0; i -= 11 )`

5.5 do…while Repetition Statement

- **do…while statement**
  - **Similar to while statement**
  - **Tests loop-continuation after performing body of loop**
    - **i.e., loop body always executes at least once**

```
1  // Fig. 5.7: DoWhileTest.java
2  // do...while repetition statement.
3
4  public class DoWhileTest
5  {
6     public static void main( String args[] )
7     {
8        int counter = 1; // initialize counter
9
10       do
11       {
12          System.out.printf( "%d  ", counter );
13          ++counter;
14       } while ( counter <= 10 ); // end do...while
15
16       System.out.println(); // outputs a newline
17    } // end main
18 } // end class DoWhileTest
```

```
1  2  3  4  5  6  7  8  9  10
```

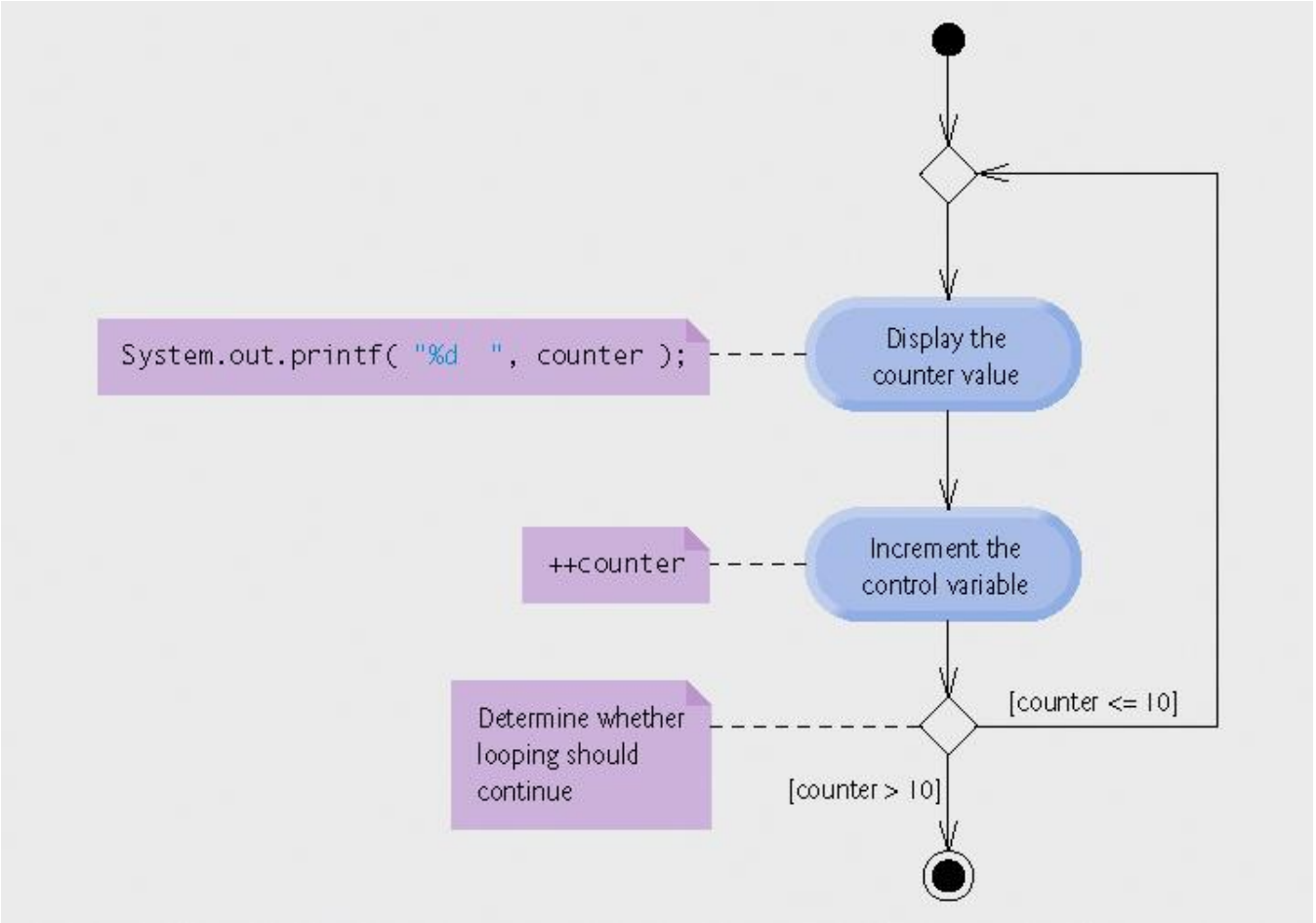


Fig. 5.8 | `do...while` repetition statement UML activity diagram.

5.6 `switch` Multiple-Selection Statement

- **`switch` statement**
  - **Used for multiple selections**

```java
import java.util.Scanner;

public class AverageCalculator
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        int number;

        System.out.print("Enter a number: ");
        number = input.nextInt();

        if(number % 2 == 0)
        {
            System.out.print(number + " is even");
        }
        else
        {
            System.out.print(number + " is odd");
        }

    }//end main
}//end class
```

```java
import java.util.Scanner;

public class AverageCalculator
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        int number;

        System.out.print("Enter a number: ");
        number = input.nextInt();

        switch (number % 2)
        {
            case 0:
                System.out.print(number + " is even");
                break;

            default:
                System.out.print(number + " is odd");
        }

    }//end main
}//end class
```

```
Enter a number: 6
6 is even
```

5.7 break and continue Statements

- break/continue
  - Alter flow of control
- break statement
  - Causes immediate exit from control structure
    - Used in while, for, do…while or switch statements
- continue statement
  - Skips remaining statements in loop body
  - Proceeds to next iteration
    - Used in while, for or do…while statements

```java
// Fig. 5.12: BreakTest.java
// break statement exiting a for statement.
public class BreakTest
{
   public static void main( String args[] )
   {
      int count; // control variable also used after loop terminates

      for ( count = 1; count <= 10; count++ ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            break;         // terminate loop

         System.out.printf( "%d ", count );
      } // end for

      System.out.printf( "\nBroke out of loop at count = %d\n", count );
   } // end main
} // end class BreakTest
```

```
1 2 3 4
Broke out of loop at count = 5
```

```java
// Fig. 5.13: ContinueTest.java
// continue statement terminating an iteration of a for statement.
public class ContinueTest
{
   public static void main( String args[] )
   {
      for ( int count = 1; count <= 10; count++ ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            continue;      // skip remaining code in loop

         System.out.printf( "%d ", count );
      } // end for

      System.out.println( "\nUsed continue to skip printing 5" );
   } // end main
} // end class ContinueTest
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

5.8 Logical Operators

- **Logical operators**
  - **Allows for forming more complex conditions**
  - **Combines simple conditions**
- **Java logical operators**
  - **&& (conditional AND)**
  - **|| (conditional OR)**
  - **! (logical NOT)**

5.8 Logical Operators (Cont.)

- **Conditional AND (&&) Operator**
  - Consider the following `if` statement

```
if ( gender == FEMALE && age >= 65 )
   ++seniorFemales;
```

  - Combined condition is `true`
    - if and only if both simple conditions are `true`
  - Combined condition is `false`
    - if either or both of the simple conditions are `false`

expression1	expression2	expression1 && expression2
false	false	False
false	true	False
true	false	False
true	true	True

5.8 Logical Operators (Cont.)

- **Conditional OR (||) Operator**
  - **Consider the following `if` statement**

```
if ( ( semesterAverage >= 90 ) || ( finalExam >= 90 )
   System.out.println( “Student grade is A” );
```

  - **Combined condition is `true`**
    - **if either or both of the simple condition are `true`**
  - **Combined condition is `false`**
    - **if both of the simple conditions are `false`**

expression1	expression2	expression1 || expression2
false	false	false
false	true	true
true	false	true
true	true	true

5.8 Logical Operators (Cont.)

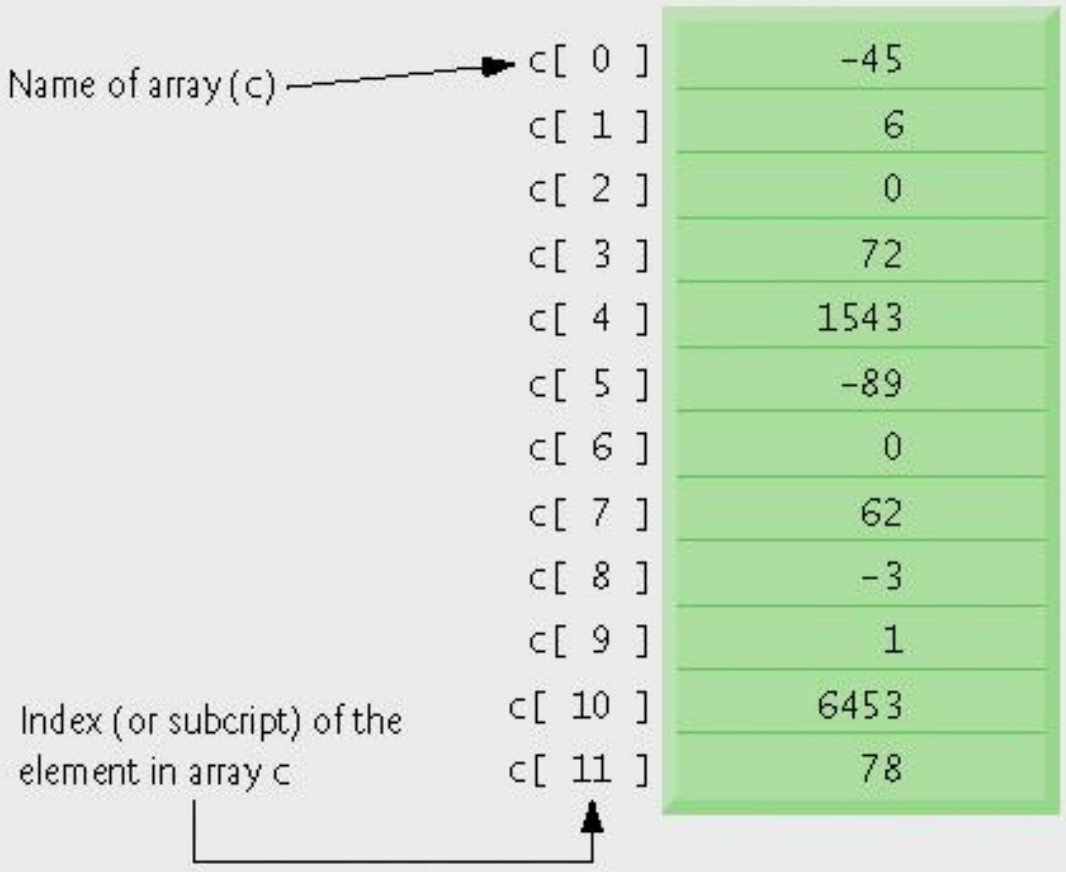
- **Logical NOT or Negatition**
  - **Consider the following `if` statement**

```
if ( !(semesterAverage >= 60) )
   System.out.println( “Student is failed” );
```

expression	!expression
false	true
true	false

7.1 Introduction

- **Arrays**
  - **Data structures**
  - **Related data items of same type**
  - **Remain same size once created**
    - **Fixed-length entries**


Name of array (c)
c[0]
c[1]
c[2]
c[3]
c[4]
c[5]
c[6]
c[7]
c[8]
c[9]
c[10]
c[11]
-45
6
0
72
1543
-89
0
62
-3
1
6453
78
Index (or subcript) of the element in array c

7.2 Arrays (Cont.)

- **Index**
  - **Also called subscript**
  - **Position number in square brackets**
  - **Must be positive integer or integer expression**
  - **First element has index zero**

```
a = 5;
b = 6;
c[ a + b ] += 2;
```

- **Adds `2` to `c[ 11 ]`**

7.2 Arrays (Cont.)

- **Examine array `c`**
  - **`c` is the array *name***
  - **`c.length` accesses array `c`'s *length***
  - **`c` has 12 *elements* (`c[0]`, `c[1]`, … `c[11]`)**
    - **The *value* of `c[0]` is `-45`**

7.3 Declaring and Creating Arrays

- **Declaring and Creating arrays**
  - **Arrays are objects that occupy memory**
  - **Created dynamically with keyword new**

```
int c[] = new int[ 12 ];
```

    - **Equivalent to**

```
int c[];  // declare array variable
c = new int[ 12 ]; // create array
```

  - **We can create arrays of objects too**

```
String b[] = new String[ 100 ];
```

7.4 Examples Using Arrays

- **Declaring arrays**
- **Creating arrays**
- **Initializing arrays**
- **Manipulating array elements**

7.4 Examples Using Arrays

- **Creating and initializing an array**
  - **Declare array**
  - **Create array**
  - **Initialize array elements**

```java
// Fig. 7.2: InitArray.java
// Creating an array.

public class InitArray
{
   public static void main( String args[] )
   {
      int array[]; // declare array named array

      array = new int[ 10 ]; // create the space for array

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // end class InitArray
```

```
Index   Value
    0       0
    1       0
    2       0
    3       0
    4       0
    5       0
    6       0
    7       0
    8       0
    9       0
```

7.4 Examples Using Arrays (Cont.)

- **Using an array initializer**
  - Use *initializer list*
    - Items enclosed in braces ({})
    - Items in list separated by commas

      ```
      int n[] = { 10, 20, 30, 40, 50 };
      ```

      - Creates a five-element array
      - Index values of 0, 1, 2, 3, 4
  - Do not need keyword new

```java
// Fig. 7.3: InitArray.java
// Initializing the elements of an array with an array initializer.

public class InitArray
{
   public static void main( String args[] )
   {
      // initializer list specifies the value for each element
      int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // end class InitArray
```

```
Index   Value
    0      32
    1      27
    2      64
    3      18
    4      95
    5      14
    6      90
    7      70
    8      60
    9      37
```

7.4 Examples Using Arrays (Cont.)

- **Calculating a value to store in each array element**
  - **Initialize elements of 10-element array to even integers**

```java
// Fig. 7.4: InitArray.java
// Calculating values to be placed into elements of an array.

public class InitArray
{
   public static void main( String args[] )
   {
      final int ARRAY_LENGTH = 10; // declare constant
      int array[] = new int[ ARRAY_LENGTH ]; // create array

      // calculate value for each array element
      for ( int counter = 0; counter < array.length; counter++ )
         array[ counter ] = 2 + 2 * counter;

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // end class InitArray
```

```
Index   Value
    0       2
    1       4
    2       6
    3       8
    4      10
    5      12
    6      14
    7      16
    8      18
    9      20
```

7.4 Examples Using Arrays (Cont.)

- **Summing the elements of an array**
  - **Array elements can represent a series of values**
    - **We can sum these values**

```java
// Fig. 7.5: SumArray.java
// Computing the sum of the elements of an array.

public class SumArray
{
   public static void main( String args[] )
   {
      int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;

      // add each element's value to total
      for ( int counter = 0; counter < array.length; counter++ )
         total += array[ counter ];

      System.out.printf( "Total of array elements: %d\n",   total );
   } // end main
} // end class SumArray
```

```
Total of array elements: 849
```

7.6 Enhanced `for` Statement

- **Enhanced `for` statement**
  - **Iterates through elements of an array or a collection without using a counter**
  - **Syntax**

```
for ( parameter : arrayName )
   statement
```

```java
// Fig. 7.12: EnhancedForTest.java
// Using enhanced for statement to total integers in an array.

public class EnhancedForTest
{
   public static void main( String args[] )
   {
      int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;

      // add each element's value to total
      for ( int number : array )
         total += number;

      System.out.printf( "Total of array elements: %d\n", total );
   } // end main
} // end class EnhancedForTest
```

```
Total of array elements: 849
```