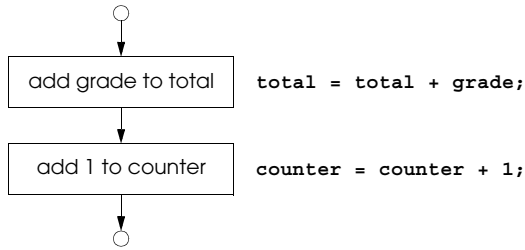

Fig. 4.1 Flowcharting Java's sequence structure.

Java Keywords

`abstract`	`boolean`	`break`	`byte`	`case`
`catch`	`char`	`class`	`continue`	`default`
`do`	`double`	`else`	`extends`	`false`
`final`	`finally`	`float`	`for`	`if`
`implements`	`import`	`instanceof`	`int`	`interface`
`long`	`native`	`new`	`null`	`package`
`private`	`protected`	`public`	`return`	`short`
`static`	`super`	`switch`	`synchronized`	`this`
`throw`	`throws`	`transient`	`true`	`try`
`void`	`volatile`	`while`		

Keywords that are reserved but not used by Java

`const`	`goto`

Fig. 4.2 Java keywords.

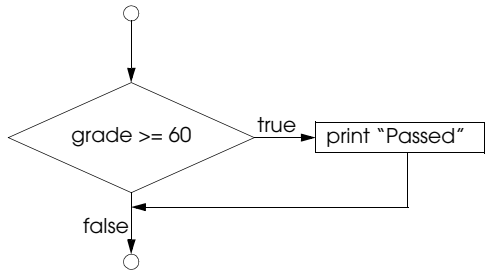

Fig. 4.3 Flowcharting the single-selection `if` structure.

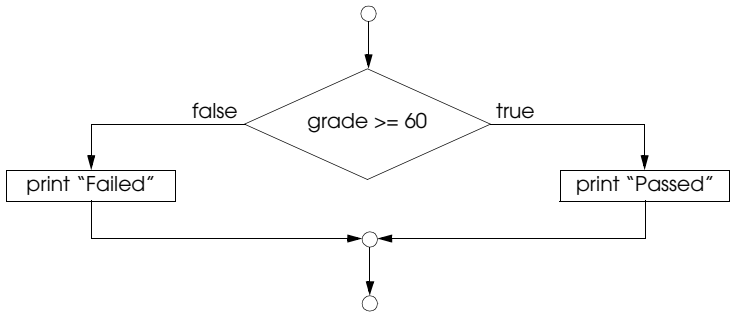

Fig. 4.4 Flowcharting the double-selection **`if/else`** structure.

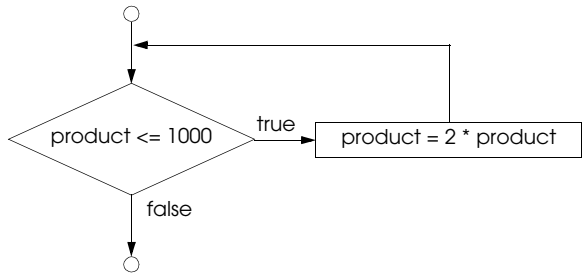

Fig. 4.5 Flowcharting the **`while`** repetition structure.

```
Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
    Input the next grade
    Add the grade into the total
    Add one to the grade counter

Set the class average to the total divided by ten
Print the class average
```

Fig. 4.6 Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

```
1  // Fig. 4.7: Average1.java
2  // Class average program with counter-controlled repetition
3  import javax.swing.JOptionPane;
4
5  public class Average1 {
6     public static void main( String args[] )
7     {
8        int total,           // sum of grades
9            gradeCounter,    // number of grades entered
10           gradeValue,      // grade value
11           average;         // average of all grades
12       String grade;        // grade typed by user
13
14       // Initialization Phase
15       total = 0;           // clear total
16       gradeCounter = 1;    // prepare to loop
17
18       // Processing Phase
19       while ( gradeCounter <= 10 ) {  // loop 10 times
20
21          // prompt for input and read grade from user
22          grade = JOptionPane.showInputDialog(
23                      "Enter integer grade: " );
24
25          // convert grade from a String to an integer
26          gradeValue = Integer.parseInt( grade );
27
28          // add gradeValue to total
29          total = total + gradeValue;
30
31          // add 1 to gradeCounter
32          gradeCounter = gradeCounter + 1;
33       }
34
35       // Termination Phase
36       average = total / 10;  // perform integer division
37
38       // display average of exam grades
39       JOptionPane.showMessageDialog(
40          null, "Class average is " + average, "Class Average",
41          JOptionPane.INFORMATION_MESSAGE );
42
43        System.exit( 0 );      // terminate the program
44    }
45 }
```

Fig. 4.7 Class-average program with counter-controlled repetition (part 1 of 2).

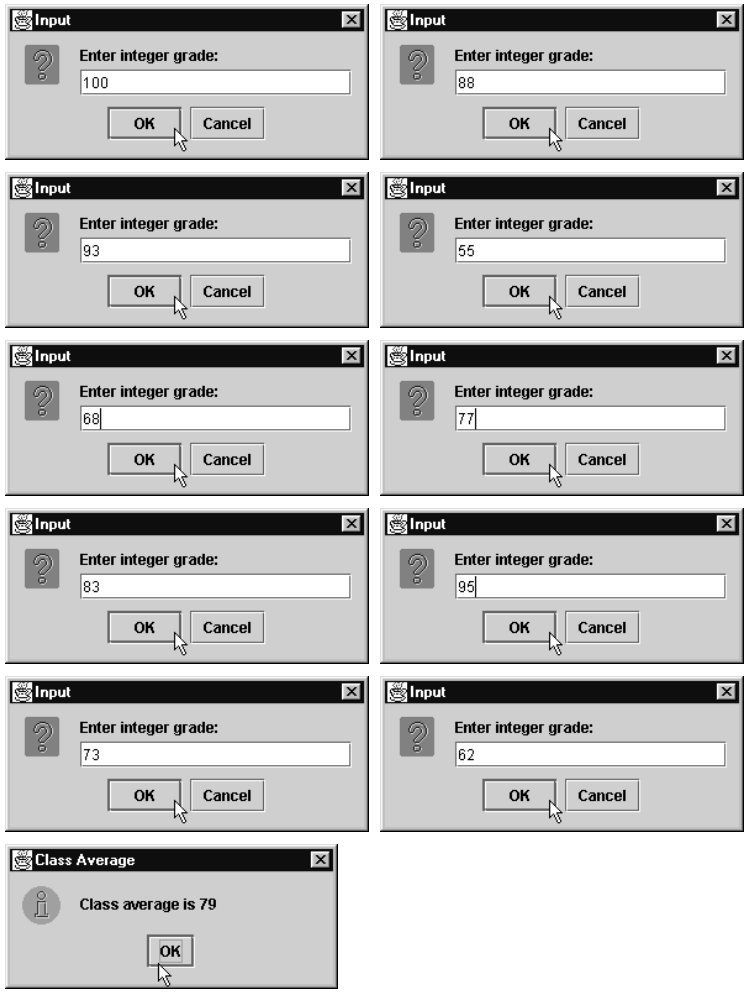

Fig. 4.7 Class-average program with counter-controlled repetition (part 2 of 2).

```
Initialize total to zero
Initialize counter to zero

Input the first grade (possibly the sentinel)
While the user has not as yet entered the sentinel
      Add this grade into the running total
      Add one to the grade counter
      Input the next grade (possibly the sentinel)

If the counter is not equal to zero
      Set the average to the total divided by the counter
      Print the average
else
      Print "No grades were entered"
```

Fig. 4.8 Pseudocode algorithm that uses sentinel-controlled repetition to solve the class-average problem.

```
1  // Fig. 4.9: Average2.java
2  // Class average program with sentinel-controlled repetition
3  import javax.swing.JOptionPane;
4  import java.text.DecimalFormat;
5
6  public class Average2 {
7     public static void main( String args[] )
8     {
9        int gradeCounter,  // number of grades entered
10           gradeValue,    // grade value
11           total;         // sum of grades
12       double average;    // average of all grades
13       String input;      // grade typed by user
14
15       // Initialization phase
16       total = 0;         // clear total
17       gradeCounter = 0;  // prepare to loop
18
19       // Processing phase
20       // prompt for input and read grade from user
21       input = JOptionPane.showInputDialog(
22                  "Enter Integer Grade, -1 to Quit:" );
23
24       // convert grade from a String to an integer
25       gradeValue = Integer.parseInt( input );
26
27       while ( gradeValue != -1 ) {
28          // add gradeValue to total
29          total = total + gradeValue;
30
31          // add 1 to gradeCounter
32          gradeCounter = gradeCounter + 1;
33
34          // prompt for input and read grade from user
35          input = JOptionPane.showInputDialog(
36                     "Enter Integer Grade, -1 to Quit:" );
37
38          // convert grade from a String to an integer
39          gradeValue = Integer.parseInt( input );
40       }
41
42       // Termination phase
43       DecimalFormat twoDigits = new DecimalFormat( "0.00" );
44
45       if ( gradeCounter != 0 ) {
46          average = (double) total / gradeCounter;
47
48          // display average of exam grades
49          JOptionPane.showMessageDialog( null,
50             "Class average is " + twoDigits.format( average ),
51             "Class Average",
52             JOptionPane.INFORMATION_MESSAGE );
53       }
```

Fig. 4.9 Class-average program with sentinel-controlled repetition (part 1 of 2).

```
54          else
55             JOptionPane.showMessageDialog( null,
56                "No grades were entered", "Class Average",
57                JOptionPane.INFORMATION_MESSAGE );
58
59          System.exit( 0 );     // terminate the program
60       }
61    }
```

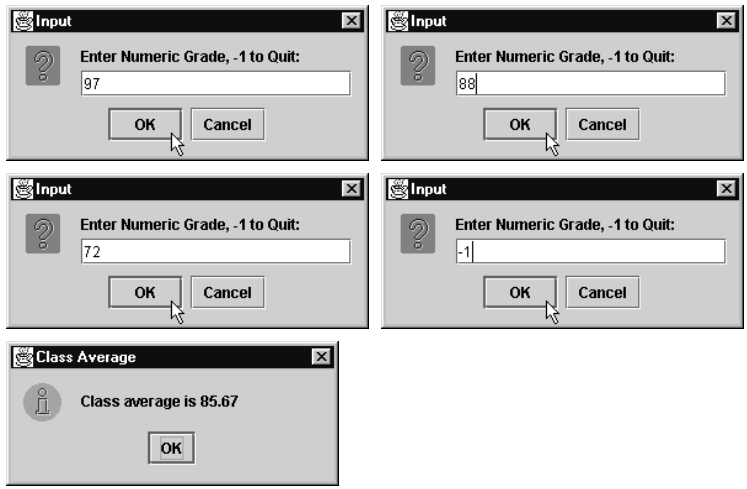

Fig. 4.9 Class-average program with sentinel-controlled repetition (part 2 of 2).

```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
    Input the next exam result

    If the student passed
        Add one to passes
    else
        Add one to failures

    Add one to student counter

Print the number of passes
Print the number of failures
If more than eight students passed
    Print "Raise tuition"
```

Fig. 4.10 Pseudocode for examination-results problem.

```
1  // Fig. 4.11: Analysis.java
2  // Analysis of examination results.
3  import javax.swing.JOptionPane;
4
5  public class Analysis {
6     public static void main( String args[] )
7     {
8        // initializing variables in declarations
9        int passes = 0,              // number of passes
10           failures = 0,            // number of failures
11           student = 1,             // student counter
12           result;                  // one exam result
13       String input,                // user-entered value
14              output;               // output string
15
16       // process 10 students; counter-controlled loop
17       while ( student <= 10 ) {
18          input = JOptionPane.showInputDialog(
19                      "Enter result (1=pass,2=fail)" );
20          result = Integer.parseInt( input );
21
22          if ( result == 1 )
23             passes = passes + 1;
24          else
25             failures = failures + 1;
26
27          student = student + 1;
28       }
29
30       // termination phase
31       output = "Passed: " + passes +
32                "\nFailed: " + failures;
```

Fig. 4.11 Java program for examination-results problem (part 1 of 2).

```
33
34          if( passes > 8 )
35             output = output + "\nRaise Tuition";
36
37          JOptionPane.showMessageDialog( null, output,
38             "Analysis of Examination Results",
39             JOptionPane.INFORMATION_MESSAGE );
40
41          System.exit( 0 );
42       }
43    }
```

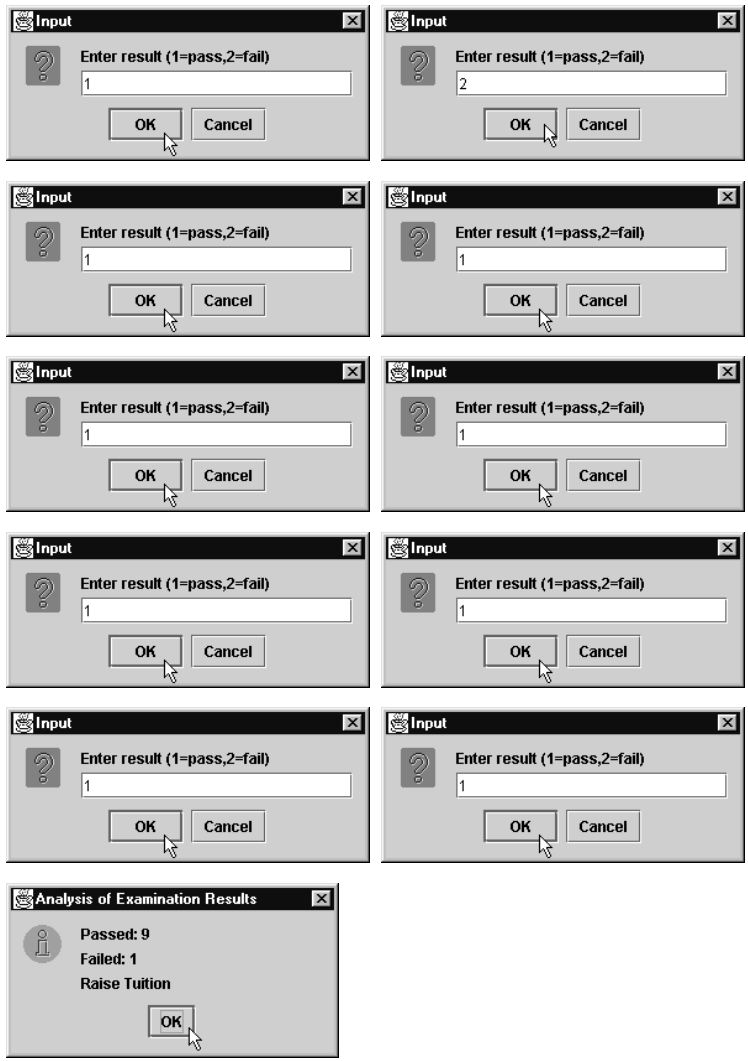

Fig. 4.11 Java program for examination-results problem (part 2 of 2).

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. 4.12 Arithmetic assignment operators.

Operator	Called	Sample expression	Explanation
`++`	preincrement	`++a`	Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides.
`++`	postincrement	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
`--`	predecrement	`--b`	Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides.
`--`	postdecrement	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

Fig. 4.13 The increment and decrement operators.

```
1  // Fig. 4.14: Increment.java
2  // Preincrementing and postincrementing
3
4  public class Increment {
5     public static void main( String args[] )
6     {
7        int c;
8
9        c = 5;
10       System.out.println( c );   // print 5
11       System.out.println( c++ ); // print 5 then postincrement
12       System.out.println( c );   // print 6
13
14       System.out.println();      // skip a line
15
16       c = 5;
17       System.out.println( c );   // print 5
18       System.out.println( ++c ); // preincrement then print 6
19       System.out.println( c );   // print 6
20    }
21 }
```

```
5
5
6

5
6
6
```

Fig. 4.14 The difference between preincrementing and postincrementing.

Operators	Associativity	Type
`()`	left to right	parentheses
`++ --`	right to left	unary postfix
`++ -- + - (`*type*`)`	right to left	unary
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Fig. 4.15 Precedence and associativity of the operators discussed so far.

Type	Size in bits	Values	Standard
`boolean`	8	`true` or `false`	
`char`	16	`'\u0000'` to `'\uFFFF'`	(ISO Unicode character set)
`byte`	8	–128 to +127	
`short`	16	–32,768 to +32,767	
`int`	32	–2,147,483,648 to +2,147,483,647	
`long`	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	
`float`	32	–3.40292347E+38 to +3.40292347E+38	(IEEE 754 floating point)
`double`	64	–1.79769313486231570E+308 to +1.79769313486231570E+308	(IEEE 754 floating point)

Fig. 4.16 The Java primitive data types.