Problem Solving Techniques

OBJECTIVES

In this chapter you will learn:

- To use basic problem-solving techniques.
- 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 i**ncrement** and **decrement** operators.
- To use the **for** and **do…while** repetition statements to execute statements in a program repeatedly.
- To understand multiple selection using the **switch** selection statement.
- To use the **break** and **continue** program control statements to alter the flow of control.

Control Structures

- Sequential execution
  - Statements are normally executed one after the other in the order in which they are written
  - Three control structures
    - The sequence structure,
    - The selection structure and
    - The repetition structure
- Selection Statements
  - `if` statement
    - Single-selection statement
  - `if…else` statement
    - Double-selection statement
  - `switch` statement
    - Multiple-selection statement

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

`if, if..else` Statement

- `if` statements
  - Execute an action if the specified condition is `true`
- `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`

The ? : operator in Java

The value of a variable often depends on whether a particular boolean expression is or is not true. For instance one common operation is setting the value of a variable to the maximum of two quantities.

In Java you might write

```
if (a > b)
  { max = a; }
else
  { max = b; }
```

Using the conditional operator you can rewrite the above example in a single line like this:

```
max = (a > b) ? a : b;
```

The condition, (a > b), is tested. If it is true the first value, a, is returned. If it is false, the second value, b, is returned. Whichever value is returned is dependent on the conditional test, a > b. The condition can be any expression which returns a boolean value.

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

- Nested `if…else` statements
  - `if…else` statements can be put inside other `if…else` statements
- Dangling-`else` problem
  - `else`s are always associated with the immediately preceding `if` unless otherwise specified by braces `{ }`
- Blocks
  - Braces `{ }` associate statements into blocks
  - Blocks can replace individual statements as an `if` body

`while` Repetition Statement

- `while` statement
  - Repeats an action while its loop-continuation condition remains true
  - 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.
- Counter-controlled repetition
  - Use a counter variable to count the number of times a loop is iterated

```java
// Fig. 4.6: GradeBook.java
// GradeBook class that solves class-average problem using
// counter-controlled repetition.
import java.util.Scanner; // program uses class Scanner

public class GradeBook
{
   private String courseName; // name of course this GradeBook represents

   // constructor initializes courseName
   public GradeBook( String name )
   {
      courseName = name; // initializes course
   } // end constructor

   // method to set the course name
   public void setCourseName( String name )
   {
      courseName = name; // store the course name
   } // end method setCourseName

   // method to retrieve the course name
   public String getCourseName()
   {
      return courseName;
   } // end method getCourseName

```

Assign a value to instance variable **courseName**

Declare method **setCourseName**

Declare method **getCourseName**

```
28    // display a welcome message to the GradeBook user
29    public void displayMessage()
30    {
31       // getCourseName gets the name of the course
32       System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33          getCourseName() );
34    } // end method displayMessage
35
36    // determine class average based on 10 grades entered by user
37    public void determineClassAverage()
38    {
39       // create Scanner to obtain input from command window
40       Scanner input = new Scanner( System.in );
41
42       int total; // sum of grades entered by user
43       int gradeCounter; // number of the grade to be entered next
44       int grade; // grade value entered by user
45       int average; // average of grades
46
47       // initialization phase
48       total = 0; // initialize total
49       gradeCounter = 1; // initialize loop counter
50
```

Declare method **displayMessage**

Declare method **determineClassAverage**

Declare and initialize **Scanner** variable **input**

Declare local **int** variables **total**, **gradeCounter**, **grade** and **average**

```java
        // processing phase
        while ( gradeCounter <= 10 ) // loop 10 times
        {
           System.out.print( "Enter grade: " ); // prompt
           grade = input.nextInt(); // input next grade
           total = total + grade; // add grade to total
           gradeCounter = gradeCounter + 1; // increment counter by 1
        } // end while

        // termination phase
        average = total / 10; // integer division yields integer result

        // display total and average of grades
        System.out.printf( "\nTotal of all 10 grades is %d\n", total );
        System.out.printf( "Class average is %d\n", average );
    } // end method determineClassAverage

} // end class GradeBook
```

while loop iterates as long as **gradeCounter <= 10**

Increment the counter variable **gradeCounter**

Calculate average grade

Display results

```java
// Fig. 4.7: GradeBookTest.java
// Create GradeBook object and invoke its determineClassAverage method.

public class GradeBookTest
{
   public static void main( String [] args )
   {
      // create GradeBook object myGradeBook and
      // pass course name to constructor
      GradeBook myGradeBook = new GradeBook(
         "CS101 Introduction to Java Programming"

      myGradeBook.displayMessage(); // display welcome message
      myGradeBook.determineClassAverage(); // find average of 10 grades
   } // end main

} // end class GradeBookTest
```

Create a new **GradeBook** object

Pass the course's name to the **GradeBook** constructor as a **string**

Call **GradeBook**'s **determineClassAverage** method

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

There are two types of casting

- Widening **Casting** (automatically) : Implicit Casting
  - ❑**converting** a **smaller type** to a **larger type** size.
  - ❑byte -> short -> char -> int -> long -> float -> double.
- Narrowing **Casting** (manually) : Explicit Casting
  - ❑ **converting** a **larger type** to a **smaller** size **type**.
  - ❑double -> float -> long -> int -> char -> short -> byte.

Implicit Casting

```java
public class ImplicitCastingExample {
    public static void main(String args[]) {
        byte i = 40;
        // No casting needed for below conversion
        short j = i;
        int k = j;
        long l = k;
        float m = l;
        double n = m;
        System.out.println("byte value : "+i);
        System.out.println("short value : "+j);
        System.out.println("int value : "+k);
        System.out.println("long value : "+l);
        System.out.println("float value : "+m);
        System.out.println("double value : "+n);
    }
}
```

Output:

```
byte value : 40
short value : 40
int value : 40
long value : 40
float value : 40.0
double value : 40.0
```

Common Programming Error

- Assuming that integer division rounds (rather than truncates) can lead to incorrect results. For example, $7 \div 4$, which yields 1.75 in conventional arithmetic, truncates to 1 in integer arithmetic, rather than rounding to 2.

Explicit Casting

```java
public class ExplicitCastingExample {
    public static void main(String args[]) {
        double d = 30.0;
        // Explicit casting is needed for below conversion
        float f = (float) d;
        long l = (long) f;
        int i = (int) l;
        short s = (short) i;
        byte b = (byte) s;
        System.out.println("double value : "+d);
        System.out.println("float value : "+f);
        System.out.println("long value : "+l);
        System.out.println("int value : "+i);
        System.out.println("short value : "+s);
        System.out.println("byte value : "+b);
    }
}
```

Output:

```
double value : 30.0
float value : 30.0
long value : 30
int value : 30
short value : 30
byte value : 30
```

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

```java
// Fig. 4.9: GradeBook.java
// GradeBook class that solves class-average program using
// sentinel-controlled repetition.
import java.util.Scanner; // program uses class Scanner

public class GradeBook
{
   private String courseName; // name of course this GradeBook represents

   // constructor initializes courseName
   public GradeBook( String name )
   {
      courseName = name; // initializes courseName
   } // end constructor

   // method to set the course name
   public void setCourseName( String name )
   {
      courseName = name; // store the course name
   } // end method setCourseName

   // method to retrieve the course name
   public String getCourseName()
   {
      return courseName;
   } // end method getCourseName

```

```java
   // display a welcome message to the GradeBook user
   public void displayMessage()
   {
      // getCourseName gets the name of the course
      System.out.printf( "Welcome to the grade book for\n%s!\n\n",
         getCourseName() );
   } // end method displayMessage

   // determine the average of an arbitrary number of grades
   public void determineClassAverage()
   {
      // create Scanner to obtain input from command window
      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

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

      // processing phase
      // prompt for input and read grade from user
      System.out.print( "Enter grade or -1 to quit: " );
      grade = input.nextInt();

```

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

         // prompt for input and read next grade from user
         System.out.print( "Enter grade or -1 to quit: " );
         grade = input.nextInt();
      } // end while

      // termination phase
      // if user entered at least one grade...
      if ( gradeCounter != 0 )
      {
         // calculate average of all grades entered
         average = (double) total / gradeCounter;

         // display total and average (with two digits of precision)
         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 // no grades were entered, so output appropriate message
         System.out.println( "No grades were entered" );
   } // end method determineClassAverage

} // end class GradeBook
```

while loop iterates as long as **grade !=** the sentinel value, **-1**

unary cast operator

Calculate average grade using **(double)** to perform explicit conversion

Display average grade

Display **"No grades were entered"** message

```java
// Fig. 4.10: GradeBookTest.java
// Create GradeBook object and invoke its determineClassAverage method.

public class GradeBookTest
{
   public static void main( String [] args )
   {
      // create GradeBook object myGradeBook and
      // pass course name to constructor
      GradeBook myGradeBook = new GradeBook(
         "CS101 Introduction to Java Programming" );

      myGradeBook.displayMessage(); // display welcome message
      myGradeBook.determineClassAverage(); // find average of grades
   } // end main

} // end class GradeBookTest
```

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 257
Class average is 85.67
```

Formulating Algorithms: Nested Control Statements

```java
// Fig. 4.12: Analysis.java
// Analysis of examination results.
import java.util.Scanner; // class uses class Scanner

public class Analysis
{
   public void processExamResults
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      // initializing variables in declarations
      int passes = 0; // number of passes
      int failures = 0; // number of failures
      int studentCounter = 1; // student counter
      int result; // one exam result (obtains value from user)

      // process 10 students using counter-controlled loop
      while ( studentCounter <= 10 )
      {
         // prompt user for input and obtain value from user
         System.out.print( "Enter result (1 = pass, 2 = fail): " );
         result = input.nextInt();

```

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

```java
         // if...else nested in while
         if ( result == 1 )           // if result 1,
            passes = passes + 1;      // increment passes;
         else                         // else result is not 1, so
            failures = failures + 1; // increment failures

         // increment studentCounter so loop eventually terminates
         studentCounter = studentCounter + 1;
      } // end while

      // termination phase; prepare and display results
      System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );

      // determine whether more than 8 students passed
      if ( passes > 8 )
         System.out.println( "Raise Tuition" );
   } // end method processExamResults

} // end class Analysis
```

Determine whether this student passed or failed and increment the appropriate variable

Determine whether more than eight students passed the exam

```java
// Fig. 4.13: AnalysisTest.java
// Test program for class Analysis.

public class AnalysisTest
{
   public static void main( String args[] )
   {
      Analysis application = new Analysis(); // create Analysis object
      application.processExamResults(); // call method to process results
   } // end main

} // end class AnalysisTest
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Raise Tuition
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

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

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

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

```java
// Fig. 4.16: Increment.java
// Prefix increment and postfix increment operators.

public class Increment
{
   public static void main( String [] args )
   {
      int c;

      // demonstrate postfix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // print 5
      System.out.println( c++ ); // print 5 then postincrement
      System.out.println( c );   // print 6

      System.out.println(); // skip a line

      // demonstrate prefix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // print 5
      System.out.println( ++c ); // preincrement then print 6
      System.out.println( c );   // print 6

   } // end main

} // end class Increment
```

Postincrementing the `c` variable

Preincrementing the `c` variable

```
5
5
6

5
6
6
```

Operators	Associativity	Type
`++ --`	right to left	unary postfix
`++ -- + - ( type )`	right to left	unary prefix
`* / %`	left to right	Multiplicative
`+ -`	left to right	Additive
`< <= > >=`	left to right	Relational
`== !=`	left to right	Equality
`?:`	right to left	Conditional
`= += -= *= /= %=`	right to left	assignment

Precedence and associativity of the operators discussed so far.

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

while loop vs for loop

```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
```

```java
// Fig. 5.2: ForCounter.java
// Counter-controlled repetition with the for repetition statement.

public class ForCounter
{
   public static void main( String [] args )
   {
      // for statement header includes initialization,
      // loop-continuation condition and increment
      for ( int counter = 1; counter <= 10; counter++ )
         System.out.printf( "%d  ", counter );

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

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

`for` Repetition Statement (Cont.)

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

can usually be rewritten as:

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

- Using commas instead of the two required semicolons in a `for` header is a syntax error.

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 )`

Example using for loop and java.lang.Math class

- Compound interest application
- *A person invests $1,000 in a savings account yielding 5% interest. Assuming that all the interest is left on deposit, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula to determine the amounts:*

  $a = p\,(1 + r)^n$

  where

  p is the original amount invested (i.e., the principal)
  r is the annual interest rate (e.g., use 0.05 for 5%)
  n is the number of years
  a is the amount on deposit at the end of the nth year.

```java
// Fig. 5.6: Interest.java
// Compound-interest calculations with for.

public class Interest
{
   public static void main( String [] args )
   {
      double amount; // amount on deposit at end of each year
      double principal = 1000.0; // initial amount before interest
      double rate = 0.05; // interest rate

      // display headers
      System.out.printf( "%s%20s\n", "Year", "Amount on deposit" );

      // calculate amount on deposit for each of ten years
      for ( int year = 1; year <= 10; year++ )
      {
         // calculate new amount for specified year
         amount = principal * Math.pow( 1.0 + rate, year );

         // display the year and the amount
         System.out.printf( "%4d%,20.2f\n", year, amount );
      } // end for
   } // end main
} // end class Interest
```

Second string is right justified and displayed with a field width of 20

Output:

```
Year    Amount on deposit
   1            1,050.00
   2            1,102.50
   3            1,157.63
   4            1,215.51
   5            1,276.28
   6            1,340.10
   7            1,407.10
   8            1,477.46
   9            1,551.33
  10            1,628.89
```

Examples Using the `for` Statement (Cont.)

- Formatting output
  - Field width
  - Comma (`,`) formatting flag to output numbers with grouping separators
- `static` method
  - *ClassName.methodName*(*arguments*)

`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

```java
// Fig. 5.7: DoWhileTest.java
// do...while repetition statement.

public class DoWhileTest
{
   public static void main( String [] args )
   {
      int counter = 1; // initialize counter

      do
      {
         System.out.printf( "%d  ", counter );
         ++counter;
      } while ( counter <= 10 ); // end do...while

      System.out.println(); // outputs a newline
   } // end main
} // end class DoWhileTest
```

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

`switch` Multiple-Selection Statement

- `switch` statement
  - Used for multiple selections

```java
// Fig. 5.9: GradeBook.java
// GradeBook class uses switch statement to count A, B, C, D and F grades.
import java.util.Scanner; // program uses class Scanner

public class GradeBook
{
   private String courseName; // name of course this GradeBook represents
   private int total; // sum of grades
   private int gradeCounter; // number of grades entered
   private int aCount; // count of A grades
   private int bCount; // count of B grades
   private int cCount; // count of C grades
   private int dCount; // count of D grades
   private int fCount; // count of F grades

   // constructor initializes courseName;
   // int instance variables are initialized to 0 by default
   public GradeBook( String name )
   {
      courseName = name; // initializes courseName
   } // end constructor

   // method to set the course name
   public void setCourseName( String name )
   {
      courseName = name; // store the course name
   } // end method setCourseName
```

```java
   // method to retrieve the course name
   public String getCourseName()
   {
      return courseName;
   } // end method getCourseName

   // display a welcome message to the GradeBook user
   public void displayMessage()
   {
      // getCourseName gets the name of the course
      System.out.printf( "Welcome to the grade book for\n%s!\n\n",
         getCourseName() );
   } // end method displayMessage

   // input arbitrary number of grades from user
   public void inputGrades()
   {
      Scanner input = new Scanner( System.in );

      int grade; // grade entered by user

      System.out.printf( "%s\n%s\n   %s\n   %s\n",
         "Enter the integer grades in the range 0-100.",
         "Type the end-of-file indicator to terminate input:",
         "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
         "On Windows type <ctrl> z then press Enter" );

```

End of file indicator:
<ctrl> z

```java
      // loop until user enters the end-of-file indicator
      while ( input.hasNext() )
      {
         grade = input.nextInt(); // read grade
         total += grade; // add grade to total
         ++gradeCounter; // increment number of grades

         // call method to increment appropriate counter
         incrementLetterGradeCounter( grade );
      } // end while
   } // end method inputGrades

   // add 1 to appropriate counter for specified grade
   public void incrementLetterGradeCounter( int numericGrade )
   {
      // determine which grade was entered
      switch ( grade / 10 )
      {
         case 9:  // grade was between 90
         case 10: // and 100
            ++aCount; // increment aCount
            break; // necessary to exit switch

         case 8: // grade was between 80 and 89
            ++bCount; // increment bCount
            break; // exit switch

```

As long as the end of file indicator has not been typed, method hasNext will return true.

```
83          case 7: // grade was between 70 and 79
84             ++cCount; // increment cCount
85             break; // exit switch
86
87          case 6: // grade was between 60 and 69
88             ++dCount; // increment dCount
89             break; // exit switch
90
91          default: // grade was less than 60
92             ++fCount; // increment fCount
93             break; // optional; will exit switch anyway
94       } // end switch
95    } // end method incrementLetterGradeCounter
96
97    // display a report based on the grades entered by user
98    public void displayGradeReport()
99    {
100      System.out.println( "\nGrade Report:" );
101
102      // if user entered at least one grade...
103      if ( gradeCounter != 0 )
104      {
105         // calculate average of all grades entered
106         double average = (double) total / gradeCounter;
107
```

```
        // output summary of results
        System.out.printf( "Total of the %d grades entered is %d\n",
           gradeCounter, total );
        System.out.printf( "Class average is %.2f\n", average );
        System.out.printf( "%s\n%s%d\n%s%d\n%s%d\n%s%d\n%s%d\n",
           "Number of students who received each grade:",
           "A: ", aCount,   // display number of A grades
           "B: ", bCount,   // display number of B grades
           "C: ", cCount,   // display number of C grades
           "D: ", dCount,   // display number of D grades
           "F: ", fCount ); // display number of F grades
     } // end if
     else // no grades were entered, so output appropriate message
        System.out.println( "No grades were entered" );
  } // end method displayGradeReport
} // end class GradeBook
```

```java
// Fig. 5.10: GradeBookTest.java
// Create GradeBook object, input grades and display grade report.

public class GradeBookTest
{
   public static void main( String[] args )
   {
      // create GradeBook object myGradeBook and
      // pass course name to constructor
      GradeBook myGradeBook = new GradeBook(
         "CS101 Introduction to Java Programming" );

      myGradeBook.displayMessage(); // display welcome message
      myGradeBook.inputGrades(); // read grades from user
      myGradeBook.displayGradeReport(); // display report based on grades
   } // end main
} // end class GradeBookTest
```

Output

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter the integer grades in the range 0-100.
Type the end-of-file indicator to terminate input:
   On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
   On Windows type <ctrl> z then press Enter
99
92
45
57
63
71
76
85
90
100
^Z

Grade Report:
Total of the 10 grades entered is 778
Class average is 77.80
Number of students who received each grade:
A: 4
B: 1
C: 2
D: 1
F: 2
```

`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

break vs continue statement

```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
```

Loop 10 times

Exit **for** statement (break) when count equals 5

```
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
```

Logical Operators

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

```
x & y     // bitwise AND, 0101 & 0011 = 0001
x | y     // bitwise OR,  0101 | 0011 = 0111

x && y    // true if both x and y are true
x || y    // true if either x or y are true
```

- Boolean Logical Exclusive OR (^)
  - One of its operands is `true` and the other is `false`
    - Evaluates to `true`
  - Both operands are `true` or both are `false`
    - Evaluates to `false`