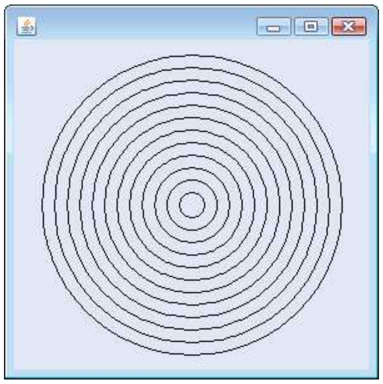

Fig. 5.29 | Drawing concentric circles.

5.2 Modify Exercise 5.16 from the end-of-chapter exercises to read input using dialogs and to display the bar chart using rectangles of varying lengths.

5.12 Wrap-Up

In this chapter, we completed our introduction to control statements, which enable you to control the flow of execution in methods. Chapter 4 discussed `if`, `if…else` and `while`. This chapter demonstrated `for`, `do…while` and `switch`. We showed that any algorithm can be developed using combinations of the sequence structure, the three types of selection statements—`if`, `if…else` and `switch`—and the three types of repetition statements—`while`, `do…while` and `for`. In this chapter and Chapter 4, we discussed how you can combine these building blocks to utilize proven program-construction and problem-solving techniques. You used the `break` statement to exit a `switch` statement and to immediately terminate a loop, and used a `continue` statement to terminate a loop's current iteration and proceed with the loop's next iteration. This chapter also introduced Java's logical operators, which enable you to use more complex conditional expressions in control statements. In Chapter 6, we examine methods in greater depth.

Summary

Section 5.2 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition (p. 153) requires a control variable, the initial value of the control variable, the increment by which the control variable is modified each time through the loop (also known as each iteration of the loop) and the loop-continuation condition that determines whether looping should continue.
- You can declare a variable and initialize it in the same statement.

Section 5.3 `for` Repetition Statement

- The `while` statement can be used to implement any counter-controlled loop.
- The `for` statement (p. 155) specifies all the details of counter-controlled repetition in its header
- When the `for` statement begins executing, its control variable is declared and initialized. If the loop-continuation condition is initially true, the body executes. After executing the loop's body, the increment expression executes. Then the loop-continuation test is performed again to determine whether the program should continue with the next iteration of the loop.

- The general format of the `for` statement is

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

  where the *initialization* expression names the loop's control variable and provides its initial value, *loopContinuationCondition* determines whether the loop should continue executing and *increment* modifies the control variable's value, so that the loop-continuation condition eventually becomes false. The two semicolons in the `for` header are required.
- Most `for` statements can be represented with equivalent `while` statements as follows:

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

- Typically, `for` statements are used for counter-controlled repetition and `while` statements for sentinel-controlled repetition.
- If the *initialization* expression in the `for` header declares the control variable, the control variable can be used only in that `for` statement—it will not exist outside the `for` statement.
- The expressions in a `for` header are optional. If the *loopContinuationCondition* is omitted, Java assumes that it's always true, thus creating an infinite loop. You might omit the *initialization* expression if the control variable is initialized before the loop. You might omit the *increment* expression if the increment is calculated with statements in the loop's body or if no increment is needed.
- The increment expression in a `for` acts as if it's a standalone statement at the end of the `for`'s body.
- A `for` statement can count downward by using a negative increment—i.e., a decrement (p. 158).
- If the loop-continuation condition is initially `false`, the `for` statement's body does not execute.

Section 5.4 Examples Using the `for` *Statement*

- Java treats floating-point constants like `1000.0` and `0.05` as type `double`. Similarly, Java treats whole-number constants like `7` and `-22` as type `int`.
- The format specifier `%4s` outputs a `String` in a field width (p. 161) of 4—that is, `printf` displays the value with at least 4 character positions. If the value to be output is less than 4 character positions wide, the value is right justified (p. 161) in the field by default. If the value is greater than 4 character positions wide, the field width expands to accommodate the appropriate number of characters. To left justify (p. 161) the value, use a negative integer to specify the field width.
- `Math.pow(`x`, `y`)` (p. 162) calculates the value of x raised to the y^{th} power. The method receives two `double` arguments and returns a `double` value.
- The comma (`,`) formatting flag (p. 162) in a format specifier indicates that a floating-point value should be output with a grouping separator (p. 162). The actual separator used is specific to the user's locale (i.e., country). In the United States, the number will have commas separating every three digits and a decimal point separating the fractional part of the number, as in 1,234.45.
- The `.` in a format specifier indicates that the integer to its right is the number's precision.

Section 5.5 `do…while` *Repetition Statement*

- The `do…while` statement (p. 163) is similar to the `while` statement. In the `while`, the program tests the loop-continuation condition at the beginning of the loop, before executing its body; if the condition is false, the body never executes. The `do…while` statement tests the loop-continuation condition *after* executing the loop's body; therefore, the body always executes at least once.

Section 5.6 `switch` Multiple-Selection Statement

- The `switch` statement (p. 165) performs different actions based on the possible values of a constant integral expression (a constant value of type `byte`, `short`, `int` or `char`, but not `long`), or a `String`.
- The end-of-file indicator is a system-dependent keystroke combination that terminates user input. On UNIX/Linux/Mac OS X systems, end-of-file is entered by typing the sequence *<Ctrl> d* on a line by itself. This notation means to simultaneously press both the *Ctrl* key and the *d* key. On Windows systems, enter end-of-file by typing *<Ctrl> z*.
- `Scanner` method `hasNext` (p. 168) determines whether there's more data to input. This method returns the `boolean` value `true` if there's more data; otherwise, it returns `false`. As long as the end-of-file indicator has not been typed, method `hasNext` will return `true`.
- The `switch` statement consists of a block that contains a sequence of `case` labels (p. 168) and an optional `default` case (p. 168).
- In a `switch`, the program evaluates the controlling expression and compares its value with each `case` label. If a match occurs, the program executes the statements for that `case`.
- Listing cases consecutively with no statements between them enables the cases to perform the same set of statements.
- Every value you wish to test in a `switch` must be listed in a separate `case` label.
- Each `case` can have multiple statements, and these need not be placed in braces.
- A `case`'s statements typically end with a `break` statement (p. 168) that terminates the `switch`'s execution.
- Without `break` statements, each time a match occurs in the `switch`, the statements for that case and subsequent cases execute until a `break` statement or the end of the `switch` is encountered.
- If no match occurs between the controlling expression's value and a `case` label, the optional `default` case executes. If no match occurs and the `switch` does not contain a `default` case, program control simply continues with the first statement after the `switch`.

Section 5.7 Class `AutoPolicy` Case Study: `String`s in `switch` Statements

- `String`s can be used in a `switch` statement's controlling expression and `case` labels.

Section 5.8 `break` and `continue` Statements

- The `break` statement, when executed in a `while`, `for`, `do…while` or `switch`, causes immediate exit from that statement.
- The `continue` statement (p. 174), when executed in a `while`, `for` or `do…while`, skips the loop's remaining body statements and proceeds with its next iteration. In `while` and `do…while` statements, the program evaluates the loop-continuation test immediately. In a `for` statement, the increment expression executes, then the program evaluates the loop-continuation test.

Section 5.9 Logical Operators

- Simple conditions are expressed in terms of the relational operators `>`, `<`, `>=` and `<=` and the equality operators `==` and `!=`, and each expression tests only one condition.
- Logical operators (p. 176) enable you to form more complex conditions by combining simple conditions. The logical operators are `&&` (conditional AND), `||` (conditional OR), `&` (boolean logical AND), `|` (boolean logical inclusive OR), `^` (boolean logical exclusive OR) and `!` (logical NOT).
- To ensure that two conditions are *both* true, use the `&&` (conditional AND) operator. If either or both of the simple conditions are false, the entire expression is false.
- To ensure that either *or* both of two conditions are true, use the `||` (conditional OR) operator, which evaluates to true if either or both of its simple conditions are true.

- A condition using `&&` or `||` operators (p. 176) uses short-circuit evaluation (p. 178)—they're evaluated only until it's known whether the condition is true or false.
- The `&` and `|` operators (p. 178) work identically to the `&&` and `||` operators but always evaluate both operands.
- A simple condition containing the boolean logical exclusive OR (`^`; p. 179) operator is `true` *if and only if one of its operands is* `true` *and the other is* `false`. If both operands are `true` or both are `false`, the entire condition is `false`. This operator is also guaranteed to evaluate both of its operands.
- The unary `!` (logical NOT; p. 179) operator "reverses" the value of a condition.

Self-Review Exercises

5.1 Fill in the blanks in each of the following statements:

a) Typically, ________ statements are used for counter-controlled repetition and ________ statements for sentinel-controlled repetition.

b) The `do…while` statement tests the loop-continuation condition ________ executing the loop's body; therefore, the body always executes at least once.

c) The ________ statement selects among multiple actions based on the possible values of an integer variable or expression, or a `String`.

d) The ________ statement, when executed in a repetition statement, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.

e) The ________ operator can be used to ensure that two conditions are *both* true before choosing a certain path of execution.

f) If the loop-continuation condition in a `for` header is initially ________, the program does not execute the `for` statement's body.

g) Methods that perform common tasks and do not require objects are called ________ methods.

5.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The `default` case is required in the `switch` selection statement.

b) The `break` statement is required in the last case of a `switch` selection statement.

c) The expression `((x > y) && (a < b))` is true if either `x > y` is true or `a < b` is true.

d) An expression containing the `||` operator is true if either or both of its operands are true.

e) The comma (`,`) formatting flag in a format specifier (e.g., `%,20.2f`) indicates that a value should be output with a thousands separator.

f) To test for a range of values in a `switch` statement, use a hyphen (–) between the start and end values of the range in a `case` label.

g) Listing cases consecutively with no statements between them enables the cases to perform the same set of statements.

5.3 Write a Java statement or a set of Java statements to accomplish each of the following tasks:

a) Sum the odd integers between 1 and 99, using a `for` statement. Assume that the integer variables `sum` and `count` have been declared.

b) Calculate the value of `2.5` raised to the power of `3`, using the `pow` method.

c) Print the integers from 1 to 20, using a `while` loop and the counter variable `i`. Assume that the variable `i` has been declared, but not initialized. Print only five integers per line. [*Hint:* Use the calculation `i % 5`. When the value of this expression is 0, print a newline character; otherwise, print a tab character. Assume that this code is an application. Use the `System.out.println()` method to output the newline character, and use the `System.out.print('\t')` method to output the tab character.]

d) Repeat part (c), using a `for` statement.

5.4 Find the error in each of the following code segments, and explain how to correct it:

a)
```
i = 1;
while (i <= 10);
   ++i;
}
```
b)
```
for (k = 0.1; k != 1.0; k += 0.1)
   System.out.println(k);
```
c)
```
switch (n)
{
   case 1:
      System.out.println("The number is 1");
   case 2:
      System.out.println("The number is 2");
      break;
   default:
      System.out.println("The number is not 1 or 2");
      break;
}
```
d) The following code should print the values 1 to 10:
```
n = 1;
while (n < 10)
   System.out.println(n++);
```

Answers to Self-Review Exercises

5.1 a) `for`, `while`. b) after. c) `switch`. d) `continue`. e) `&&` (conditional AND). f) `false`. g) `static`.

5.2 a) False. The `default` case is optional. If no default action is needed, then there's no need for a `default` case. b) False. The `break` statement is used to exit the `switch` statement. The `break` statement is not required for the last case in a `switch` statement. c) False. *Both* of the relational expressions must be true for the entire expression to be true when using the `&&` operator. d) True. e) True. f) False. The `switch` statement does not provide a mechanism for testing ranges of values, so every value that must be tested should be listed in a separate `case` label. g) True.

5.3 a)
```
sum = 0;
for (count = 1; count <= 99; count += 2)
   sum += count;
```
b)
```
double result = Math.pow(2.5, 3);
```
c)
```
i = 1;

while (i <= 20)
{
   System.out.print(i);

   if (i % 5 == 0)
      System.out.println();
   else
      System.out.print('\t');

   ++i;
}
```

d)
```
for (i = 1; i <= 20; i++)
{
   System.out.print(i);

   if (i % 5 == 0)
      System.out.println();
   else
      System.out.print('\t');
}
```

5.4 a) Error: The semicolon after the `while` header causes an infinite loop, and there's a missing left brace.
Correction: Replace the semicolon by a `{`, or remove both the `;` and the `}`.

b) Error: Using a floating-point number to control a `for` statement may not work, because floating-point numbers are represented only approximately by most computers.
Correction: Use an integer, and perform the proper calculation in order to get the values you desire:

```
for (k = 1; k != 10; k++)
   System.out.println((double) k / 10);
```

c) Error: The missing code is the `break` statement in the statements for the first `case`.
Correction: Add a `break` statement at the end of the statements for the first `case`. This omission is not necessarily an error if you want the statement of `case 2:` to execute every time the `case 1:` statement executes.

d) Error: An improper relational operator is used in the `while`'s continuation condition.
Correction: Use `<=` rather than `<`, or change `10` to `11`.

Exercises

5.5 Describe the four basic elements of counter-controlled repetition.

5.6 Compare and contrast the `while` and `for` repetition statements.

5.7 Discuss a situation in which it would be more appropriate to use a `do…while` statement than a `while` statement. Explain why.

5.8 Compare and contrast the `break` and `continue` statements.

5.9 Find and correct the error(s) in each of the following segments of code:

a)
```
For (i = 100, i >= 1, i++)
   System.out.println(i);
```

b) The following code should print whether integer `value` is odd or even:

```
switch (value % 2)
{
   case 0:
      System.out.println("Even integer");

   case 1:
      System.out.println("Odd integer");
}
```

c) The following code should output the odd integers from 19 to 1:

```
for (i = 19; i >= 1; i += 2)
   System.out.println(i);
```

d) The following code should output the even integers from 2 to 100:

```
counter = 2;

do
{
   System.out.println(counter);
   counter += 2;
} While (counter < 100);
```

5.10 What does the following program do?

```java
// Exercise 5.10: Printing.java
public class Printing
{
   public static void main(String[] args)
   {
      for (int i = 1; i <= 10; i++)
      {
         for (int j = 1; j <= 5; j++)
            System.out.print('@');

         System.out.println();
      }
   }
} // end class Printing
```

5.11 *(Find the Smallest Value)* Write an application that finds the smallest of several integers. Assume that the first value read specifies the number of values to input from the user.

5.12 *(Calculating the Product of Odd Integers)* Write an application that calculates the product of the odd integers from 1 to 15.

5.13 *(Factorials) Factorials* are used frequently in probability problems. The factorial of a positive integer *n* (written *n*! and pronounced "*n* factorial") is equal to the product of the positive integers from 1 to *n*. Write an application that calculates the factorials of 1 through 20. Use type `long`. Display the results in tabular format. What difficulty might prevent you from calculating the factorial of 100?

5.14 *(Modified Compound-Interest Program)* Modify the compound-interest application of Fig. 5.6 to repeat its steps for interest rates of 5%, 6%, 7%, 8%, 9% and 10%. Use a `for` loop to vary the interest rate.

5.15 *(Triangle Printing Program)* Write an application that displays the following patterns separately, one below the other. Use `for` loops to generate the patterns. All asterisks (*) should be printed by a single statement of the form `System.out.print('*');` which causes the asterisks to print side by side. A statement of the form `System.out.println();` can be used to move to the next line. A statement of the form `System.out.print(' ');` can be used to display a space for the last two patterns. There should be no other output statements in the program. [*Hint:* The last two patterns require that each line begin with an appropriate number of blank spaces.]

```
(a)            (b)            (c)            (d)
*              **********     **********              *
**             *********       *********             **
***            ********         ********            ***
****           *******           *******           ****
*****          ******             ******          *****
******         *****               *****         ******
*******        ****                 ****        *******
********       ***                   ***       ********
*********      **                     **      *********
**********     *                       *     **********
```

5.16 *(Bar Chart Printing Program)* One interesting application of computers is to display graphs and bar charts. Write an application that reads five numbers between 1 and 30. For each number that's read, your program should display the same number of adjacent asterisks. For example, if your program reads the number 7, it should display *******. Display the bars of asterisks *after* you read all five numbers.

5.17 *(Calculating Sales)* An online retailer sells five products whose retail prices are as follows: Product 1, $2.98; product 2, $4.50; product 3, $9.98; product 4, $4.49 and product 5, $6.87. Write an application that reads a series of pairs of numbers as follows:

a) product number
b) quantity sold

Your program should use a `switch` statement to determine the retail price for each product. It should calculate and display the total retail value of all products sold. Use a sentinel-controlled loop to determine when the program should stop looping and display the final results.

5.18 *(Modified Compound-Interest Program)* Modify the application in Fig. 5.6 to use only integers to calculate the compound interest. [*Hint:* Treat all monetary amounts as integral numbers of pennies. Then break the result into its dollars and cents portions by using the division and remainder operations, respectively. Insert a period between the dollars and the cents portions.]

5.19 Assume that `i = 1`, `j = 2`, `k = 3` and `m = 2`. What does each of the following statements print?

a) `System.out.println(i == 1);`
b) `System.out.println(j == 3);`
c) `System.out.println((i >= 1) && (j < 4));`
d) `System.out.println((m <= 99) & (k < m));`
e) `System.out.println((j >= i) || (k == m));`
f) `System.out.println((k + m < j) | (3 - j >= k));`
g) `System.out.println(!(k > m));`

5.20 *(Calculating the Value of π)* Calculate the value of π from the infinite series

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

Print a table that shows the value of π approximated by computing the first 200,000 terms of this series. How many terms do you have to use before you first get a value that begins with 3.14159?

5.21 *(Pythagorean Triples)* A right triangle can have sides whose lengths are all integers. The set of three integer values for the lengths of the sides of a right triangle is called a Pythagorean triple. The lengths of the three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Write an application that displays a table of the Pythagorean triples for `side1`, `side2` and the `hypotenuse`, all no larger than 500. Use a triple-nested `for` loop that tries all possibilities. This method is an example of "brute-force" computing. You'll learn in more advanced computer science courses that for many interesting problems there's no known algorithmic approach other than using sheer brute force.

5.22 *(Modified Triangle Printing Program)* Modify Exercise 5.15 to combine your code from the four separate triangles of asterisks such that all four patterns print side by side. [*Hint:* Make clever use of nested `for` loops.]

5.23 *(De Morgan's Laws)* In this chapter, we discussed the logical operators `&&`, `&`, `||`, `|`, `^` and `!`. De Morgan's laws can sometimes make it more convenient for us to express a logical expression. These laws state that the expression `!(`*condition1* `&&` *condition2*`)` is logically equivalent to the expression `(!`*condition1* `||` `!`*condition2*`)`. Also, the expression `!(`*condition1* `||` *condition2*`)` is logically equivalent to the expression `(!`*condition1* `&&` `!`*condition2*`)`. Use De Morgan's laws to write equivalent expressions for each of the following, then write an application to show that both the original expression and the new expression in each case produce the same value:

a) `!(x < 5) && !(y >= 7)`
b) `!(a == b) || !(g != 5)`
c) `!((x <= 8) && (y > 4))`
d) `!((i > 4) || (j <= 6))`

5.24 *(Diamond Printing Program)* Write an application that prints the following diamond shape. You may use output statements that print a single asterisk (*), a single space or a single newline character. Maximize your use of repetition (with nested `for` statements), and minimize the number of output statements.

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

5.25 *(Modified Diamond Printing Program)* Modify the application you wrote in Exercise 5.24 to read an odd number in the range 1 to 19 to specify the number of rows in the diamond. Your program should then display a diamond of the appropriate size.

5.26 A criticism of the `break` statement and the `continue` statement is that each is unstructured. Actually, these statements can always be replaced by structured statements, although doing so can be awkward. Describe in general how you'd remove any `break` statement from a loop in a program and replace it with some structured equivalent. [*Hint:* The `break` statement exits a loop from the body of the loop. The other way to exit is by failing the loop-continuation test. Consider using in the loop-continuation test a second test that indicates "early exit because of a 'break' condition."] Use the technique you develop here to remove the `break` statement from the application in Fig. 5.13.

5.27 What does the following program segment do?

```
for (i = 1; i <= 5; i++)
{
   for (j = 1; j <= 3; j++)
   {
      for (k = 1; k <= 4; k++)
         System.out.print('*');

      System.out.println();
   } // end inner for

   System.out.println();
} // end outer for
```

5.28 Describe in general how you'd remove any `continue` statement from a loop in a program and replace it with some structured equivalent. Use the technique you develop here to remove the `continue` statement from the program in Fig. 5.14.

5.29 *("The Twelve Days of Christmas" Song)* Write an application that uses repetition and `switch` statements to print the song "The Twelve Days of Christmas." One `switch` statement should be used to print the day ("first," "second," and so on). A separate `switch` statement should be used to print the remainder of each verse. Visit the website `en.wikipedia.org/wiki/The_Twelve_Days_of_Christmas_(song)` for the lyrics of the song.

5.30 *(Modified `AutoPolicy` Class)* Modify class `AutoPolicy` in Fig. 5.11 to validate the two-letter state codes for the northeast states. The codes are: CT for Connecticut, MA for Massachusetts, ME for Maine, NH for New Hampshire, NJ for New Jersey, NY for New York, PA for Pennsylvania and VT for Vermont. In `AutoPolicy` method `setState`, use the logical OR (`||`) operator (Section 5.9) to create a compound condition in an `if...else` statement that compares the method's argument with each two-letter code. If the code is incorrect, the `else` part of the `if...else` statement should display an error message. In later chapters, you'll learn how to use exception handling to indicate that a method received an invalid value.

Making a Difference

5.31 *(Global Warming Facts Quiz)* The controversial issue of global warming has been widely publicized by the film "An Inconvenient Truth," featuring former Vice President Al Gore. Mr. Gore and a U.N. network of scientists, the Intergovernmental Panel on Climate Change, shared the 2007 Nobel Peace Prize in recognition of "their efforts to build up and disseminate greater knowledge about man-made climate change." Research *both* sides of the global warming issue online (you might want to search for phrases like "global warming skeptics"). Create a five-question multiple-choice quiz on global warming, each question having four possible answers (numbered 1–4). Be objective and try to fairly represent both sides of the issue. Next, write an application that administers the quiz, calculates the number of correct answers (zero through five) and returns a message to the user. If the user correctly answers five questions, print "Excellent"; if four, print "Very good"; if three or fewer, print "Time to brush up on your knowledge of global warming," and include a list of some of the websites where you found your facts.

5.32 *(Tax Plan Alternatives; The "FairTax")* There are many proposals to make taxation fairer. Check out the FairTax initiative in the United States at `www.fairtax.org`. Research how the proposed FairTax works. One suggestion is to eliminate income taxes and most other taxes in favor of a 23% consumption tax on all products and services that you buy. Some FairTax opponents question the 23% figure and say that because of the way the tax is calculated, it would be more accurate to say the rate is 30%—check this carefully. Write a program that prompts the user to enter expenses in various expense categories they have (e.g., housing, food, clothing, transportation, education, health care, vacations), then prints the estimated FairTax that person would pay.

5.33 *(Facebook User Base Growth)* According to CNNMoney.com, Facebook hit one billion users in October 2012. Using the compound-growth technique you learned in Fig. 5.6 and assuming its user base grows at a rate of 4% per month, how many months will it take for Facebook to grow its user base to 1.5 billion users? How many months will it take for Facebook to grow its user base to two billion users?