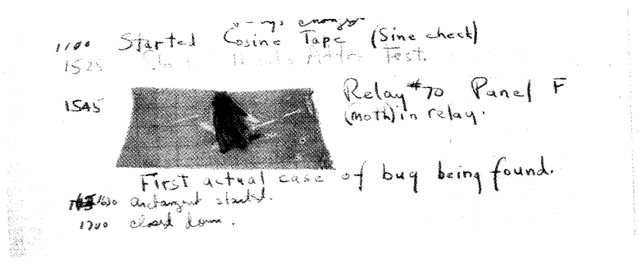
Random Fact 5.1

The First Bug

According to legend, the first bug was one found in 1947 in the Mark II, a huge electro-mechanical computer at Harvard University. It really was caused by a bug—a moth was trapped in a relay switch. Actually, from the note that the operator left in the log book next to the moth (see the figure), it appears as if the term "bug" had already been in active use at the time.



The First Bug

The pioneering computer scientist Maurice Wilkes wrote: "Somehow, at the Moore School and afterwards, one had always assumed there would be no particular difficulty in getting programs right. I can remember the exact instant in time at which it dawned on me that a great part of my future life would be spent finding mistakes in my own programs."

Summary of Learning Objectives

Explain the flow of execution in a loop.

- A `while` statement executes a block of code repeatedly. A condition controls for how long the loop is executed.
- An off-by-one error is a common error when programming loops. Think through simple test cases to avoid this type of error.

Use `for` loops to implement counting loops.

- You use a `for` loop when a variable runs from a starting to an ending value with a constant increment or decrement.
- Make a choice between symmetric and asymmetric loop bounds.
- Count the number of iterations to check that your `for` loop is correct.

Implement loops that process a data set until a sentinel value is encountered.

- Sometimes, the termination condition of a loop can only be evaluated in the middle of a loop. You can introduce a Boolean variable to control such a loop.

Use nested loops to implement multiple levels of iterations.

- When the body of a loop contains another loop, the loops are nested. A typical use of nested loops is printing a table with rows and columns.

Apply loops to the implementation of simulations that involve random values.

- In a simulation, you repeatedly generate random numbers and use them to simulate an activity.

Use a debugger to locate errors in a running program.

- A debugger is a program that you can use to execute another program and analyze its run-time behavior.
- You can make effective use of a debugger by mastering just three concepts: breakpoints, single-stepping, and inspecting variables.
- When a debugger executes a program, the execution is suspended when-ever a breakpoint is reached.
- The single-step command executes the program one line at a time.
- A debugger can be used only to analyze the presence of bugs, not to show that a program is bug-free.
- Use the divide-and-conquer technique to locate the point of failure of a program.
- During debugging, compare the actual contents of variables against the values you know they should have.

Classes, Objects, and Methods Introduced in this Chapter

```
java.util.Random
   nextDouble
   nextInt
```

Media Resources

www.wiley.com/go/global/horstmann

- ***Worked Example*** Credit Card Processing
- ***Worked Example*** Manipulating the Pixels in an Image
- ***Worked Example*** A Sample Debugging Session
- Lab Exercises
- ***Animation*** Tracing a Loop
- ***Animation*** The `for` Loop
- Practice Quiz
- Code Completion Exercises

Review Exercises

★★ **R5.1** Which loop statements does Java support? Give simple rules when to use each loop type.

★★ **R5.2** What does the following code print?

```
for (int i = 0; i < 10; i++)
{
```

```
      for (int j = 0; j < 10; j++)
         System.out.print(i * j % 10);
      System.out.println();
   }
```

★★ **R5.3** How many iterations do the following loops carry out? Assume that `i` is an integer variable that is not changed in the loop body.

a. `for (i = 1; i <= 10; i++) . . .`
b. `for (i = 0; i < 10; i++) . . .`
c. `for (i = 10; i > 0; i--) . . .`
d. `for (i = -10; i <= 10; i++) . . .`
e. `for (i = 10; i >= 0; i++) . . .`
f. `for (i = -10; i <= 10; i = i + 2) . . .`
g. `for (i = -10; i <= 10; i = i + 3) . . .`

★ **R5.4** Rewrite the following `for` loop into a `while` loop.

```
int s = 0;
for (int i = 1; i <= 10; i++) s = s + i;
```

★★ **R5.5** Rewrite the following `do` loop into a `while` loop.

```
int n = 1;
double x = 0;
double s;
do
{
   s = 1.0 / (n * n);
   x = x + s;
   n++;
}
while (s > 0.01);
```

★ **R5.6** What is an infinite loop? On your computer, how can you terminate a program that executes an infinite loop?

★★★ **R5.7** Give three strategies for implementing the following "loop and a half":

```
Loop
   Read name of bridge.
   If not OK, exit loop.
   Read length of bridge in feet.
   If not OK, exit loop.
   Convert length to meters.
   Print bridge data.
```

Use a Boolean variable, a `break` statement, and a method with multiple `return` statements. Which of these three approaches do you find clearest?

★ **R5.8** Implement a loop that prompts a user to enter a number between 1 and 10, giving three tries to get it right.

★ **R5.9** Sometimes students write programs with instructions such as "Enter data, 0 to quit" and that exit the data entry loop when the user enters the number 0. Explain why that is usually a poor idea.

★ **R5.10** How would you use a random number generator to simulate the drawing of a playing card?

★ **R5.11** What is an "off-by-one error"? Give an example from your own programming experience.

★★ **R5.12** Give an example of a `for` loop in which symmetric bounds are more natural. Give an example of a `for` loop in which asymmetric bounds are more natural.

★ **R5.13** What are nested loops? Give an example where a nested loop is typically used.

★T **R5.14** Explain the differences between these debugger operations:

- Stepping into a method
- Stepping over a method

★★T **R5.15** Explain in detail how to inspect the string stored in a `String` object in your debugger.

★★T **R5.16** Explain in detail how to inspect the information stored in a `Rectangle` object in your debugger.

★★T **R5.17** Explain in detail how to use your debugger to inspect the balance stored in a `BankAccount` object.

★★T **R5.18** Explain the divide-and-conquer strategy to get close to a bug in a debugger.

Programming Exercises

★ **P5.1** Complete the program in How To 5.1 on page 199. Your program should read twelve temperature values and print the month with the highest temperature.

★★★ **P5.2** *Credit Card Number Check.* The last digit of a credit card number is the *check digit*, which protects against transcription errors such as an error in a single digit or switching two digits. The following method is used to verify actual credit card numbers but, for simplicity, we will describe it for numbers with 8 digits instead of 16:

- Starting from the rightmost digit, form the sum of every other digit. For example, if the credit card number is 4358 9795, then you form the sum 5 + 7 + 8 + 3 = 23.
- Double each of the digits that were not included in the preceding step. Add all digits of the resulting numbers. For example, with the number given above, doubling the digits, starting with the next-to-last one, yields 18 18 10 8. Adding all digits in these values yields 1 + 8 + 1 + 8 + 1 + 0 + 8 = 27.
- Add the sums of the two preceding steps. If the last digit of the result is 0, the number is valid. In our case, 23 + 27 = 50, so the number is valid.

Write a program that implements this algorithm. The user should supply an 8-digit number, and you should print out whether the number is valid or not. If it is not valid, you should print out the value of the check digit that would make the number valid.

★ **P5.3** *Currency conversion.* Write a program `CurrencyConverter` that asks the user to enter today's price of one dollar in euro. Then the program reads U.S. dollar values and converts each to euro values. Stop when the user enters `Q`.

P5.4 *Projectile flight.* Suppose a cannonball is propelled vertically into the air with a starting velocity v_0. Any calculus book will tell us that the position of the ball after t seconds is $s(t) = -0.5 \cdot g \cdot t^2 + v_0 \cdot t$, where $g = 9.81$ m/sec^2 is the gravitational force of the earth. No calculus book ever mentions why someone would want to carry out such an obviously dangerous experiment, so we will do it in the safety of the computer.

In fact, we will confirm the theorem from calculus by a simulation. In our simulation, we will consider how the ball moves in very short time intervals Δt. In a short time interval the velocity v is nearly constant, and we can compute the distance the ball moves as $\Delta s = v \cdot \Delta t$. In our program, we will simply set

```
double deltaT = 0.01;
```

and update the position by

```
s = s + v * deltaT;
```

The velocity changes constantly—in fact, it is reduced by the gravitational force of the earth. In a short time interval, v decreases by $g \cdot \Delta t$, and we must keep the velocity updated as

```
v = v - g * deltaT;
```

In the next iteration the new velocity is used to update the distance.

Now run the simulation until the cannonball falls back to the earth. Get the initial velocity as an input (100 m/sec is a good value). Update the position and velocity 100 times per second, but only print out the position every full second. Also print out the values from the exact formula $s(t) = -0.5 \cdot g \cdot t^2 + v_0 \cdot t$ for comparison. Use a class `Cannonball`.

What is the benefit of this kind of simulation when an exact formula is available? Well, the formula from the calculus book is *not* exact. Actually, the gravitational force diminishes the farther the cannonball is away from the surface of the earth. This complicates the algebra sufficiently that it is not possible to give an exact formula for the actual motion, but the computer simulation can simply be extended to apply a variable gravitational force. For cannonballs, the calculus-book formula is actually good enough, but computers are necessary to compute accurate trajectories for higher-flying objects such as ballistic missiles.

★ **P5.5** Write a program that prints the powers of ten

```
1.0
10.0
100.0
1000.0
10000.0
100000.0
1000000.0
1.0E7
1.0E8
1.0E9
1.0E10
1.0E11
```

Implement a class

```
public class PowerGenerator
{
   /**
      Constructs a power generator.
      @param aFactor the number that will be multiplied by itself
   */
   public PowerGenerator(double aFactor) { . . . }

   /**
      Computes the next power.
   */
   public double nextPower() { . . . }
   . . .
}
```

Then supply a test class `PowerGeneratorRunner` that calls `System.out.println(myGenerator.nextPower())` twelve times.

★★ **P5.6** The *Fibonacci sequence* is defined by the following rule. The first two values in the sequence are 1 and 1. Every subsequent value is the sum of the two values preceding it. For example, the third value is 1 + 1 = 2, the fourth value is 1 + 2 = 3, and the fifth is 2 + 3 = 5. If f_n denotes the nth value in the Fibonacci sequence, then

$$f_1 = 1$$
$$f_2 = 1$$
$$f_n = f_{n-1} + f_{n-2} \quad \text{if } n > 2$$

Write a program that prompts the user for n and prints the first n values in the Fibonacci sequence. Use a class `FibonacciGenerator` with a method `nextNumber`.

Hint: There is no need to store all values for f_n. You only need the last two values to compute the next one in the series:

```
fold1 = 1;
fold2 = 1;
fnew = fold1 + fold2;
```

After that, discard `fold2`, which is no longer needed, and set `fold2` to `fold1` and `fold1` to `fnew`.

Your generator class will be tested with this runner program:

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

      System.out.println("Enter n:");
      int n = in.nextInt();

      FibonacciGenerator fg = new FibonacciGenerator();

      for (int i = 1; i <= n; i++)
         System.out.println(fg.nextNumber());
   }
}
```

P5.7 *Mean and standard deviation.* Write a program that reads a set of floating-point data values from the input. When the user indicates the end of input, print out the count of the values, the average, and the standard deviation. The average of a data set $x_1, \ldots, x_n$ is

$$\overline{x} = \frac{\sum x_i}{n}$$

where $\sum x_i = x_1 + \cdots + x_n$ is the sum of the input values. The standard deviation is

$$s = \sqrt{\frac{\sum (x_i - \overline{x})^2}{n - 1}}$$

However, that formula is not suitable for our task. By the time you have computed the mean, the individual x_i are long gone. Until you know how to save these values, use the numerically less stable formula

$$s = \sqrt{\frac{\sum x_i^2 - \frac{1}{n}\left(\sum x_i\right)^2}{n - 1}}$$

You can compute this quantity by keeping track of the count, the sum, and the sum of squares in the `DataSet` class as you process the input values.

★ **P5.8** *Factoring of integers.* Write a program that asks the user for an integer and then prints out all its factors in increasing order. For example, when the user enters 150, the program should print

```
2
3
5
5
```

Use a class `FactorGenerator` with a constructor `FactorGenerator(int numberToFactor)` and methods `nextFactor` and `hasMoreFactors`. Supply a class `FactorPrinter` whose `main` method reads a user input, constructs a `FactorGenerator` object, and prints the factors.

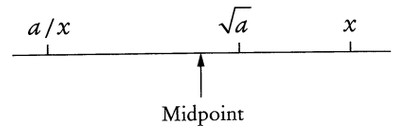
★ **P5.9** *Prime numbers.* Write a program that prompts the user for an integer and then prints out all prime numbers up to that integer. For example, when the user enters 20, the program should print

```
2
3
5
7
11
13
17
19
```

Recall that a number is a prime number if it is not divisible by any number except 1 and itself.

Supply a class `PrimeGenerator` with a method `nextPrime`.

★★ **P5.10** The *Heron method* is a method for computing square roots that was known to the ancient Greeks. If x is a guess for the value $\sqrt{a}$, then the average of x and a/x is a better guess.



Implement a class `RootApproximator` that starts with an initial guess of 1 and whose `nextGuess` method produces a sequence of increasingly better guesses. Supply a method `hasMoreGuesses` that returns `false` if two successive guesses are sufficiently close to each other (that is, they differ by no more than a small value ε). Then test your class like this:

```
RootApproximator approx = new RootApproximator(a, EPSILON);
while (approx.hasMoreGuesses())
   System.out.println(approx.nextGuess());
```

★★ **P5.11** The best known iterative method for computing the roots of a function f (that is, the x-values for which $f(x)$ is 0) is Newton–Raphson approximation. To find the zero of a function whose derivative is also known, compute

$$x_{\text{new}} = x_{\text{old}} - f(x_{\text{old}}) / f'(x_{\text{old}}).$$

For this exercise, write a program to compute nth roots of floating-point numbers. Prompt the user for a and n, then obtain $\sqrt[n]{a}$ by computing a zero of the function $f(x) = x^n - a$. Follow the approach of Exercise P5.10.

★★ **P5.12** The value of e^x can be computed as the power series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

where $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$.

Write a program that computes e^x using this formula. Of course, you can't compute an infinite sum. Just keep adding values until an individual summand (term) is less than a certain threshold. At each step, you need to compute the new term and add it to the total. Update these terms as follows:

```
term = term * x / n;
```

Follow the approach of the preceding two exercises, by implementing a class `ExpApproximator`. Its first guess should be 1.

★ **P5.13** Write a program `RandomDataAnalyzer` that generates 100 random numbers between 0 and 1000 and adds them to a `DataSet`. Print out the average and the maximum.

★★ **P5.14** Program the following simulation: Darts are thrown at random points onto the square with corners (1,1) and (–1,–1). If the dart lands inside the unit circle (that is, the circle with center (0,0) and radius 1), it is a hit. Otherwise it is a miss. Run this simulation and use it to determine an approximate value for π. Extra credit if you explain why this is a better method for estimating π than the Buffon needle program.

★★★G **P5.15** *Random walk.* Simulate the wandering of an intoxicated person in a square street grid. Draw a grid of 20 streets horizontally and 20 streets vertically. Represent the simulated drunkard by a dot, placed in the middle of the grid to start. For 100 times, have the simulated drunkard randomly pick a direction (east, west, north, south),

move one block in the chosen direction, and draw the dot. (One might expect that on average the person might not get anywhere because the moves to different directions cancel one another out in the long run, but in fact it can be shown with probability 1 that the person eventually moves outside any finite region. Use classes for the grid and the drunkard.

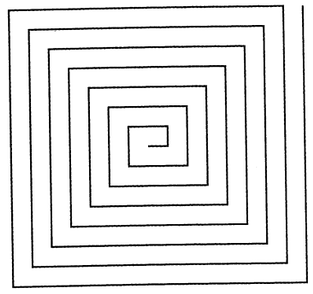
★★★G **P5.16** This exercise is a continuation of Exercise P5.4. Most cannonballs are not shot upright but at an angle. If the starting velocity has magnitude v and the starting angle is α, then the velocity is a vector with components $v_x = v \cdot \cos(\alpha)$, $v_y = v \cdot \sin(\alpha)$. In the x-direction the velocity does not change. In the y-direction the gravitational force takes its toll. Repeat the simulation from the previous exercise, but update the x and y components of the location and the velocity separately. In every iteration, plot the location of the cannonball on the graphics display as a tiny circle. Repeat until the cannonball has reached the earth again.

This kind of problem is of historical interest. The first computers were designed to carry out just such ballistic calculations, taking into account the diminishing gravity for high-flying projectiles and wind speeds.

★G **P5.17** Write a graphical application that displays a checkerboard with 64 squares, alternating white and black.

★★G **P5.18** Write a graphical application that prompts a user to enter a number n and that draws n circles with random diameter and random location. The circles should be completely contained inside the window.

★★★G **P5.19** Write a graphical application that draws a spiral, such as the following:



★★G **P5.20** It is easy and fun to draw graphs of curves with the Java graphics library. Simply draw 100 line segments joining the points $(x, f(x))$ and $(x + d, f(x + d))$, where x ranges from x_{min} to x_{max} and $d = (x_{max} - x_{min})/100$.

Draw the curve $f(x) = 0.00005x^3 - 0.03x^2 + 4x + 200$, where x ranges from 0 to 400 in this fashion.

★★★G **P5.21** Draw a picture of the "four-leaved rose" whose equation in polar coordinates is $r = \cos(2\theta)$. Let θ go from 0 to 2π in 100 steps. Each time, compute r and then compute the (x,y) coordinates from the polar coordinates by using the formula

$$x = r \cdot \cos(\theta), y = r \cdot \sin(\theta)$$

Programming Projects

Project 5.1 *Flesch Readability Index.* The following index was invented by Rudolf Flesch as a tool to gauge the legibility of a document without linguistic analysis.

- Count all words in the file. A *word* is any sequence of characters delimited by white space, whether or not it is an actual English word.
- Count all syllables in each word. To make this simple, use the following rules: Each *group* of adjacent vowels (a, e, i, o, u, y) counts as one syllable (for example, the "ea" in "real" contributes one syllable, but the "e . . . a" in "regal" count as two syllables). However, an "e" at the end of a word doesn't count as a syllable. Also, each word has at least one syllable, even if the previous rules give a count of 0.
- Count all sentences. A sentence is ended by a period, colon, semicolon, question mark, or exclamation mark.
- The index is computed by

$$\begin{aligned}\text{Index} = {} & 206.835 \\ & - 84.6 \times (\text{Number of syllables}/\text{Number of words}) \\ & - 1.015 \times (\text{Number of words}/\text{Number of sentences})\end{aligned}$$

rounded to the nearest integer.

The purpose of the index is to force authors to rewrite their text until the index is high enough. This is achieved by reducing the length of sentences and by removing long words. For example, the sentence

> The following index was invented by Flesch as a simple tool to estimate the legibility of a document without linguistic analysis.

can be rewritten as

> Flesch invented an index to check whether a text is easy to read. To compute the index, you need not look at the meaning of the words.

This index is a number, usually between 0 and 100, indicating how difficult the text is to read. Some example indices for random material from various publications are:

Comics	95
Consumer ads	82
Sports Illustrated	65
Time	57
New York Times	39
Auto insurance policy	10
Internal Revenue Code	−6

Translated into educational levels, the indices are:

91–100	5th grader
81–90	6th grader
71–80	7th grader
66–70	8th grader
61–65	9th grader
51–60	High school student
31–50	College student
0–30	College graduate
Less than 0	Law school graduate

Your program should read a text file in, compute the legibility index, and print out the equivalent educational level. Use classes `Word` and `Document`.

Project 5.2 *The game of Nim.* This is a well-known game with a number of variants. We will consider the following variant, which has an interesting winning strategy. Two players alternately take marbles from a pile. In each move, a player chooses how many marbles to take. The player must take at least one but at most half of the marbles. Then the other player takes a turn. The player who takes the last marble loses.

Write a program in which the computer plays against a human opponent. Generate a random integer between 10 and 100 to denote the initial size of the pile. Generate a random integer between 0 and 1 to decide whether the computer or the human takes the first turn. Generate a random integer between 0 and 1 to decide whether the computer plays *smart* or *stupid*. In stupid mode, the computer simply takes a random legal value (between 1 and $n/2$) from the pile whenever it has a turn. In smart mode the computer takes off enough marbles to make the size of the pile a power of 2 minus 1—that is, 3, 7, 15, 31, or 63. That is always a legal move, except if the size of the pile is currently one less than a power of 2. In that case, the computer makes a random legal move.

Note that the computer cannot be beaten in smart mode when it has the first move, unless the pile size happens to be 15, 31, or 63. Of course, a human player who has the first turn and knows the winning strategy can win against the computer.

When you implement this program, be sure to use classes `Pile`, `Player`, and `Game`. A player can be either stupid, smart, or human. (Human `Player` objects prompt for input.)

Answers to Self-Check Questions

1. Never.
2. The `waitForBalance` method would never return due to an infinite loop.
3.
```
int i = 1;
while (i <= numberOfYears)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
   i++;
}
```
4. 11 times.
5.
```
double total = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   if (value > 0) total = total + input;
}
```
6. The initial call to `in.nextDouble()` fails, terminating the program. One solution is to do all input in the loop and introduce a Boolean variable that checks whether the loop is entered for the first time.
```
double input = 0;
boolean first = true;
while (in.hasNextDouble())
{
   double previous = input;
   input = in.nextDouble();
   if (first) { first = false; }
   else if (input == previous) { System.out.println("Duplicate input"); }
}
```
7. Because we don't know whether the next input is a number or the letter `Q`.
8. No. If *all* input values are negative, the maximum is also negative. However, the `maximum` variable is initialized with 0. With this simplification, the maximum would be falsely computed as 0.
9. Change the inner loop to `for (int j = 1; j <= width; j++)`.
10. 20.
11. `int n = generator.nextInt(2); // 0 = heads, 1 = tails`
12. The program repeatedly calls `Math.toRadians(angle)`. You could simply call `Math.toRadians(180)` to compute π.
13. You should step over it because you are not interested in debugging the internals of the `println` method.
14. You should set a breakpoint. Stepping through loops can be tedious.

Be able to use common array algorithms.

- To count values, check all elements and count the matches until you reach the end.
- To compute the maximum or minimum value, initialize a candidate with the starting element. Then compare the candidate with the remaining elements and update it if you find a larger or smaller value.
- To find a value, check all elements until you have found a match.
- An array variable stores a reference to the array. Copying the variable yields a second reference to the same array.
- Use the `Arrays.copyOf` method to copy the elements of an array.

Describe the process of regression testing.

- A test suite is a set of tests for repeated testing.
- Regression testing involves repeating previously run tests to ensure that known failures of prior versions do not appear in new versions of the software.

Use two-dimensional arrays for data that is arranged in rows and columns.

- Two-dimensional arrays form a tabular, two-dimensional arrangement. You access elements with an index pair `a[i][j]`.

Classes, Objects, and Methods Introduced in this Chapter

```
java.lang.Boolean
   booleanValue
java.lang.Double
   doubleValue
java.lang.Integer
   intValue
java.util.Arrays
   copyOf
   toString
java.util.ArrayList<E>
   add
   get
   remove
   set
   size
```

Media Resources

www.wiley.com/go/global/horstmann

- ***Worked Example*** Rolling the Dice
- ***Worked Example*** A World Population Table
- Lab Exercises
- ***Animation*** Removing from an Array
- ***Animation*** Inserting into an Array
- Practice Quiz
- Code Completion Exercises

Review Exercises

★ **R6.1** What is an index? What are the bounds of an array or array list? What is a bounds error?

★ **R6.2** Write a program that contains a bounds error. Run the program. What happens on your computer? How does the error message help you locate the error?

★★ **R6.3** Write Java code for a loop that simultaneously computes the maximum and minimum values of an array list. Use an array list of accounts as an example.

★ **R6.4** Write a loop that reads 10 strings and inserts them into an array list. Write a second loop that prints out the strings in the opposite order from which they were entered.

★★ **R6.5** Consider the algorithm that we used for determining the maximum value in an array list. We set `largestYet` to the starting element, which meant that we were no longer able to use the "for each" loop. An alternate approach is to initialize `largestYet` with `null`, then loop through all elements. Of course, inside the loop you need to test whether `largestYet` is still `null`. Modify the loop that finds the bank account with the largest balance, using this technique. Is this approach more or less efficient than the one used in the text?

★★★ **R6.6** Consider another variation of the algorithm for determining the maximum value. Here, we compute the maximum value of an array of numbers.

```
double max = 0; // Contains an error!
for (double element : values)
{
   if (element > max) max = element;
}
```

However, this approach contains a subtle error. What is the error, and how can you fix it?

★ **R6.7** For each of the following sets of values, write code that fills an array a with the values.

a. 1 2 3 4 5 6 7 8 9 10

b. 0 2 4 6 8 10 12 14 16 18 20

c. 1 4 9 16 25 36 49 64 81 100

d. 0 0 0 0 0 0 0 0 0 0

e. 1 4 9 16 9 7 4 9 11

Use a loop when appropriate.

★★ **R6.8** Write a loop that fills an array a with 10 random numbers between 1 and 100. Write code (using one or more loops) to fill a with 10 different random numbers between 1 and 100.

★ **R6.9** What is wrong with the following loop?

```
double[] values = new double[10];
for (int i = 1; i <= 10; i++) values[i] = i * i;
```

Explain two ways of fixing the error.

★★★T **R6.10** Write a program that constructs an array of 20 integers and fills the first ten elements with the numbers 1, 4, 9, . . . , 100. Compile it and launch the debugger. After the array has been filled with three numbers, inspect it. What are the contents of the elements in the array beyond those that you filled?

R6.11 Rewrite the following loops without using the "for each" construct. Here, `values` has type `double`.

a.
```
for (double element : values) sum = sum + element;
```
b.
```
for (double element : values) if (element == target) return true;
```
c.
```
int i = 0;
for (double element : values) { values[i] = 2 * element; i++; }
```

R6.12 Rewrite the following loops, using the "for each" construct. Here, `values` has type `double`.

a.
```
for (int i = 0; i < values.length; i++) sum = sum + values[i];
```
b.
```
for (int i = 1; i < values.length; i++) sum = sum + values[i];
```
c.
```
for (int i = 0; i < values.length; i++)
   if (values[i] == target) return i;
```

★ **R6.13** What is wrong with these statements for printing an array list with separators?

```
System.out.print(values.get(0));
for (int i = 1; i < values.size(); i++)
{
   System.out.print(", " + values.get(i));
}
```

★ **R6.14** When finding the position of a match in Section 6.6.6, we used a `while` loop, not a `for` loop. What is wrong with using this loop instead?

```
for (pos = 0; pos < values.size() && !found; pos++)
{
   if (values.get(pos) > 100)
   {
      found = true;
   }
}
```

★★ **R6.15** When inserting an element into an array in Section 6.6.8, we moved the elements with larger index values, starting at the end of the array. Why is it wrong to start at the insertion location, like this?

```
for (int i = pos; i < size - 1; i++)
{
   values[i + 1] = values[i];
}
```

★★ **R6.16** In Section 6.6.9, we doubled the length of the array when growing it. Why didn't we just increase the size by one element?

★ **R6.17** What are parallel arrays? Why are parallel arrays indications of poor programming? How can they be avoided?

★ **R6.18** True or false?

a. All elements of an array are of the same type.
b. An array index must be an integer.
c. Arrays cannot contain string references as elements.
d. Arrays cannot contain `null` references as elements.
e. Parallel arrays must have equal length.
f. Two-dimensional arrays always have the same numbers of rows and columns.
g. Two parallel arrays can be replaced by a two-dimensional array.
h. Elements of different columns in a two-dimensional array can have different types.

★T **R6.19** Define the terms *regression testing* and *test suite*.

★T **R6.20** What is the debugging phenomenon known as *cycling*? What can you do to avoid it?

Programming Exercises

★ **P6.1** Implement a class `Purse`. A purse contains a collection of coins. For simplicity, we will only store the coin names in an `ArrayList<String>`. (We will discuss a better representation in Chapter 7.) Supply a method

```
void addCoin(String coinName)
```

Add a method `toString` to the `Purse` class that prints the coins in the purse in the format

```
Purse[Quarter,Dime,Nickel,Dime]
```

★ **P6.2** Write a method `reverse` that reverses the sequence of coins in a purse. Use the `toString` method of the preceding assignment to test your code. For example, if `reverse` is called with a purse

```
Purse[Quarter,Dime,Nickel,Dime]
```

then the purse is changed to

```
Purse[Dime,Nickel,Dime,Quarter]
```

★ **P6.3** Add a method to the `Purse` class

```
public void transfer(Purse other)
```

that transfers the contents of one purse to another. For example, if `a` is

```
Purse[Quarter,Dime,Nickel,Dime]
```

and `b` is

```
Purse[Dime,Nickel]
```

then after the call `a.transfer(b)`, `a` is

```
Purse[Quarter,Dime,Nickel,Dime,Dime,Nickel]
```

and `b` is empty.

★ **P6.4** Write a method for the `Purse` class

```
public boolean sameContents(Purse other)
```

that checks whether the other purse has the same coins in the same order.

★★ **P6.5** Write a method for the `Purse` class

```
public boolean sameCoins(Purse other)
```

that checks whether the other purse has the same coins, perhaps in a different order. For example, the purses

`Purse[Quarter,Dime,Nickel,Dime]` and `Purse[Nickel,Dime,Dime,Quarter]`

should be considered equal.

You will probably need one or more helper methods.

P6.6 A `Polygon` is a closed curve made up from line segments that join the polygon's corner points. Implement a class `Polygon` with methods

```
public double perimeter()
```

and

```
public double area()
```

that compute the circumference and area of a polygon. To compute the perimeter, compute the distance between adjacent points, and total up the distances. The area of a polygon with corners $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ is

$$\frac{1}{2}(x_0y_1 + x_1y_2 + \cdots + x_{n-1}y_0 - y_0x_1 - y_1x_2 - \cdots - y_{n-1}x_0)$$

As test cases, compute the perimeter and area of a rectangle and of a regular hexagon. *Note:* You need not draw the polygon — that is done in Exercise P6.18.

P6.7 Write a program that reads a sequence of integers into an array and that computes the alternating sum of all elements in the array. For example, if the program is executed with the input data

```
1  4  9  16  9  7  4  9  11
```

then it computes

$$1 - 4 + 9 - 16 + 9 - 7 + 4 - 9 + 11 = -2$$

★ **P6.8** Write a program that produces random permutations of the numbers 1 to 10. To generate a random permutation, you need to fill an array with the numbers 1 to 10 so that no two entries of the array have the same contents. You could do it by brute force, by calling `Random.nextInt` until it produces a value that is not yet in the array. Instead, you should implement a smart method. Make a second array and fill it with the numbers 1 to 10. Then pick one of those at random, remove it, and append it to the permutation array. Repeat 10 times. Implement a class `PermutationGenerator` with a method

```
int[] nextPermutation
```

★ **P6.9** A *run* is a sequence of adjacent repeated values. Write a program that generates a sequence of 20 random die tosses and that prints the die values, marking the runs by including them in parentheses, like this:

```
1 2 (5 5) 3 1 2 4 3 (2 2 2 2) 3 6 (5 5) 6 3 1
```

Use the following pseudocode:

```
Set a boolean variable inRun to false.
For each valid index i in the array list
   If inRun
      If values[i] is different from the preceding value
         Print )
         inRun = false
   Else
      If values[i] is the same as the following value
         Print (
         inRun = true
   Print values[i]
If inRun, print )
```


★★★ **P6.10** Write a program that generates a sequence of 20 random die tosses and that prints the die values, marking only the longest run, like this:

```
1 2 5 5 3 1 2 4 3 (2 2 2 2) 3 6 5 5 6 3 1
```

If there is more than one run of maximum length, mark the first one.

★★ **P6.11** It is a well-researched fact that men in a restroom generally prefer to maximize their distance from already occupied stalls, by occupying the middle of the longest sequence of unoccupied places.

For example, consider the situation where ten stalls are empty.

```
_ _ _ _ _ _ _ _ _ _
```

The first visitor will occupy a middle position:

```
_ _ _ _ _ X _ _ _ _
```

The next visitor will be in the middle of the empty area at the left.

```
_ _ X _ _ X _ _ _ _
```

Write a program that reads the number of stalls and then prints out diagrams in the format given above when the stalls become filled, one at a time. *Hint:* Use an array of `boolean` values to indicate whether a stall is occupied.

★★★ **P6.12** In this assignment, you will model the game of *Bulgarian Solitaire*. The game starts with 45 cards. (They need not be playing cards. Unmarked index cards work just as well.) Randomly divide them into some number of piles of random size. For example, you might start with piles of size 20, 5, 1, 9, and 10. In each round, you take one card from each pile, forming a new pile with these cards. For example, the sample starting configuration would be transformed into piles of size 19, 4, 8, 10, and 5. The solitaire is over when the piles have size 1, 2, 3, 4, 5, 6, 7, 8, and 9, in some order. (It can be shown that you always end up with such a configuration.)

In your program, produce a random starting configuration and print it. Then keep applying the solitaire step and print the result. Stop when the solitaire final configuration is reached.

★★ **P6.13** Add a method `getWinner` to the `TicTacToe` class of Section 6.8. It should return `"x"` or `"o"` to indicate a winner, or `" "` if there is no winner yet. Recall that a winning position has three matching marks in a row, column, or diagonal.

★★★ **P6.14** Write an application that plays tic-tac-toe. Your program should draw the game board, change players after every successful move, and pronounce the winner.

★★ **P6.15** *Magic squares.* An $n \times n$ matrix that is filled with the numbers 1, 2, 3, . . . , n^2 is a magic square if the sum of the elements in each row, in each column, and in the two diagonals is the same value. For example,

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Write a program that reads in n^2 values from the keyboard and tests whether they form a magic square when arranged as a square matrix.

You need to test three features:

- Did the user enter n^2 numbers for some n?
- Do each of the numbers $1, 2, \ldots, n^2$ occur exactly once in the user input?
- When the numbers are put into a square, are the sums of the rows, columns, and diagonals equal to each other?

If the size of the input is a square, test whether all numbers between 1 and n^2 are present. Then compute the row, column, and diagonal sums. Implement a class `Square` with methods

```
public void add(int i)
public boolean isMagic()
```

P6.16 Implement the following algorithm to construct magic n-by-n^2 squares; it works only if n is odd. Place a 1 in the middle of the bottom row. After k has been placed in the (i, j) square, place $k + 1$ into the square to the right and down, wrapping around the borders. However, if the square to the right and down has already been filled, or if you are in the lower-right corner, then you must move to the square straight up instead. Here is the 5×5 square that you get if you follow this method:

11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Write a program whose input is the number n and whose output is the magic square of order n if n is odd. Implement a class `MagicSquare` with a constructor that constructs the square and a `toString` method that returns a representation of the square.

G **P6.17** Implement a class `Cloud` that contains an array list of `Point2D.Double` objects. Support methods

```
public void add(Point2D.Double aPoint)
public void draw(Graphics2D g2)
```

Draw each point as a tiny circle.

Write a graphical application that draws a cloud of 100 random points.

G **P6.18** Implement a class `Polygon` that contains an array list of `Point2D.Double` objects. Support methods

```
public void add(Point2D.Double aPoint)
public void draw(Graphics2D g2)
```

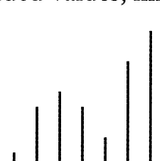
Draw the polygon by joining adjacent points with a line, and then closing it up by joining the end and start points.

Write a graphical application that draws a square and a pentagon using two `Polygon` objects.

G **P6.19** Write a class `Chart` with methods

```
public void add(int value)
public void draw(Graphics2D g2)
```


that displays a stick chart of the added values, like this:



You may assume that the values are pixel positions.

★★G **P6.20** Write a class `BarChart` with methods

```
public void add(double value)
public void draw(Graphics2D g2)
```

that displays a chart of the added values. You may assume that all added values are positive. Stretch the bars so that they fill the entire area of the screen. You must figure out the maximum of the values, and then scale each bar.

★★★G **P6.21** Improve the `BarChart` class of Exercise P6.20 to work correctly when the data contains negative values.

★★G **P6.22** Write a class `PieChart` with methods

```
public void add(double value)
public void draw(Graphics2D g2)
```

that displays a pie chart of the added values. You may assume that all data values are positive.

Programming Projects

Project 6.1 *Poker Simulator.* In this assignment, you will implement a simulation of a popular casino game usually called video poker. The card deck contains 52 cards, 13 of each suit. At the beginning of the game, the deck is shuffled. You need to devise a fair method for shuffling. (It does not have to be efficient.) Then the top five cards of the deck are presented to the player. The player can reject none, some, or all of the cards. The rejected cards are replaced from the top of the deck. Now the hand is scored. Your program should pronounce it to be one of the following:

- No pair—The lowest hand, containing five separate cards that do not match up to create any of the hands below.
- One pair—Two cards of the same value, for example two queens.
- Two pairs—Two pairs, for example two queens and two 5's.
- Three of a kind—Three cards of the same value, for example three queens.
- Straight—Five cards with consecutive values, not necessarily of the same suit, such as 4, 5, 6, 7, and 8. The ace can either precede a 2 or follow a king.
- Flush—Five cards, not necessarily in order, of the same suit.
- Full House—Three of a kind and a pair, for example three queens and two 5's
- Four of a Kind—Four cards of the same value, such as four queens.

- Straight Flush—A straight and a flush: Five cards with consecutive values of the same suit.
- Royal Flush—The best possible hand in poker. A 10, jack, queen, king, and ace, all of the same suit.

If you are so inclined, you can implement a wager. The player pays a `JavaDollar` for each game, and wins according to the following payout chart:

Hand	Payout	Hand	Payout
Royal Flush	250	Straight	4
Straight Flush	50	Three of a Kind	3
Four of a Kind	25	Two Pair	2
Full House	6	Pair of Jacks or Better	1
Flush	5		

Project 6.2 *The Game of Life* is a well-known mathematical game that gives rise to amazingly complex behavior, although it can be specified by a few simple rules. (It is not actually a game in the traditional sense, with players competing for a win.) Here are the rules. The game is played on a rectangular board. Each square can be either empty or occupied. At the beginning, you can specify empty and occupied cells in some way; then the game runs automatically. In each *generation*, the next generation is computed. A new cell is born on an empty square if it is surrounded by exactly three occupied neighbor cells. A cell dies of overcrowding if it is surrounded by four or more neighbors, and it dies of loneliness if it is surrounded by zero or one neighbor. A neighbor is an occupant of an adjacent square to the left, right, top, or bottom or in a diagonal direction. Figure 16 shows a cell and its neighbor cells.

Many configurations show interesting behavior when subjected to these rules. Figure 17 shows a *glider*, observed over five generations. Note how it moves. After four generations, it is transformed into the identical shape, but located one square to the right and below.

One of the more amazing configurations is the glider gun: a complex collection of cells that, after 30 moves, turns back into itself and a glider (see Figure 18).

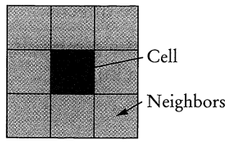


Figure 16
Neighborhood of a Cell

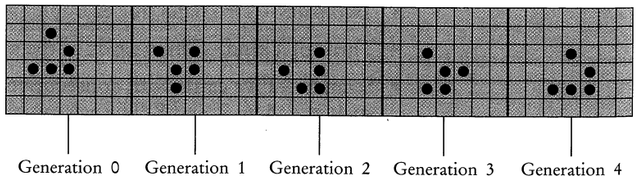


Figure 17
Glider

Program the game to eliminate the drudgery of computing successive generations by hand. Use a two-dimensional array to store the rectangular configuration. Write a program that shows successive generations of the game. You may get extra credit if you implement a graphical application that allows the user to add or remove cells by clicking with the mouse.

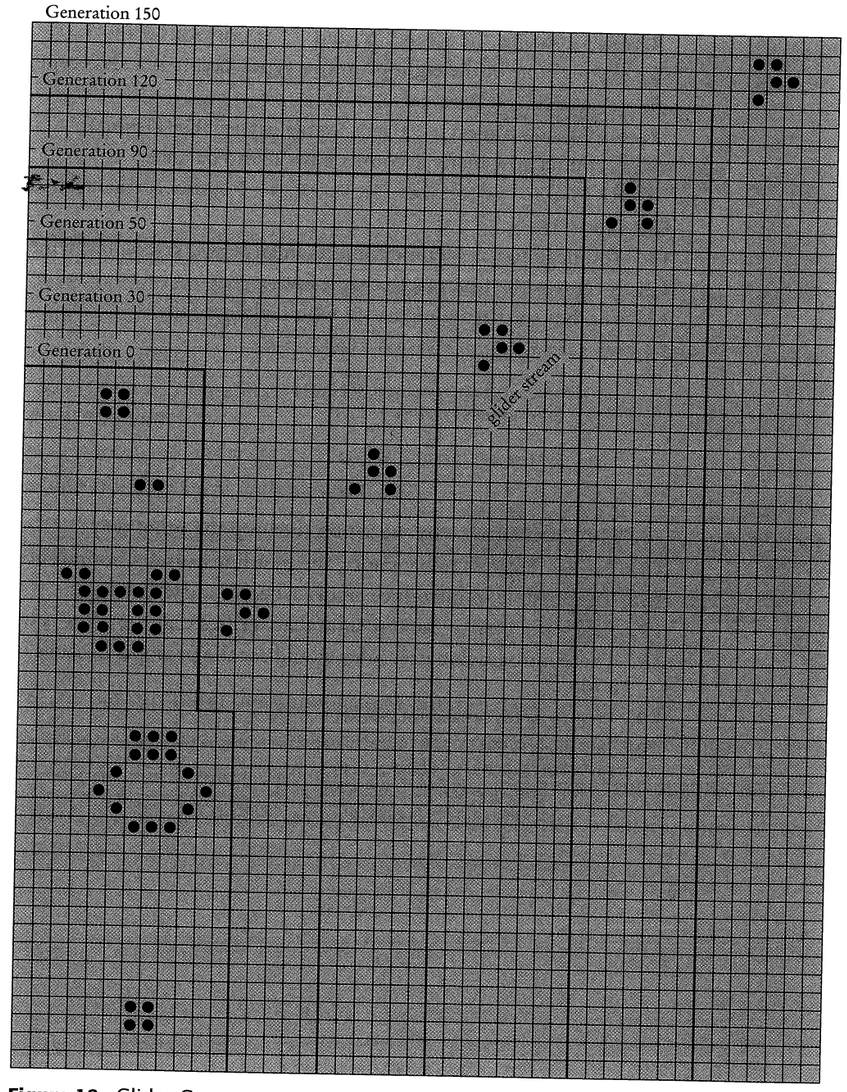


Figure 18 Glider Gun

Answers to Self-Check Questions

1. 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, but *not* 100.
2. (a) 0; (b) a run-time error: array index out of bounds; (c) a compile-time error: `c` is not initialized.
3. `new String[10];`
   `new ArrayList<String>();`
4. `names` contains the strings `"B"` and `"C"` at positions 0 and 1.
5. `double` is one of the eight primitive types. `Double` is a class type.
6. `values.set(0, values.get(0) + 1);`
7. `for (double element : values) System.out.println(element);`
8. It counts how many accounts have a zero balance.
9. `for (int i = valuesSize - 1; i >= 0; i--) System.out.println(values[i]);`
10. `valuesSize--;`
11. You need to use wrapper objects in an `ArrayList<Double>`, which is less efficient.
12. It returns the first match that it finds.
13. Yes, but the first comparison would always fail.
14.
```
for (int i = 0; i < values.size(); i++)
{
   System.out.print(values.get(i));
   if (i < values.size() - 1)
   {
      System.out.print(" | ");
   }
}
```
    Now you know why we set up the loop the other way.
15. If `names` happens to be empty, the first line causes a bounds error.
16. It is possible to introduce errors when modifying code.
17. Add a test case to the test suite that verifies that the error is fixed.
18. There is no human user who would see the prompts because input is provided from a file.
19. `int[][] array = new int[4][4];`
20.
```
int count = 0;
for (int i = 0; i < ROWS; i++)
   for (int j = 0; j < COLUMNS; j++)
      if (board[i][j].equals(" ")) count++;
```

y of Learning Objectives

ind classes that are appropriate for solving a programming problem.

- A class should represent a single concept from a problem domain, such as business, science, or mathematics.

nalyze cohesiveness and coupling of classes.

- The public interface of a class is cohesive if all of its features are related to the concept that the class represents.
- A class depends on another class if it uses objects of that class.
- It is a good practice to minimize the coupling (i.e., dependency) between classes.

ecognize immutable classes and their benefits.

- An immutable class has no mutator methods.
- References to objects of an immutable class can be safely shared.

ecognize side effects and the need to minimize them.

- A side effect of a method is any externally observable data modification.
- In Java, a method can never change parameters of primitive type.
- When designing methods, minimize side effects.

ocument preconditions and postconditions of methods.

- A precondition is a requirement that the caller of a method must meet.
- If a method is called in violation of a precondition, the method is not responsible for computing the correct result.
- An assertion is a logical condition in a program that you believe to be true.
- If a method has been called in accordance with its preconditions, then it must ensure that its postconditions are valid.

mplement static methods that do not operate on objects.

- A static method is not invoked on an object.
- When you design a static method, you must find a class into which it should be placed.

Use static variables to describe properties of a class.

- A static variable belongs to the class, not to any object of the class.

Determine the scopes of local variables and instance variables.

- The scope of a variable is the region of a program in which the variable can be accessed.
- The scope of a local variable cannot contain the declaration of another local variable with the same name.
- A local variable can shadow an instance variable with the same name. You can access the shadowed variable name through the `this` reference.
- You should give each variable the smallest scope that it needs.

Use packages to organize sets of related classes.

- A package is a set of related classes.
- The `import` directive lets you refer to a class of a package by its class name, without the package prefix.
- Use a domain name in reverse to construct an unambiguous package name.
- The path of a class file must match its package name.
- A field or method that is not declared as `public` or `private` can be accessed by all classes in the same package, which is usually not desirable.

Use JUnit for writing unit tests.

- Unit test frameworks simplify the task of writing classes that contain many test cases.
- The JUnit philosophy is to run all tests whenever you change your code.

Media Resources

www.wiley.com/go/global/horstmann

- Lab Exercises
- ***Animation*** A Method Cannot Modify a Numeric Parameter
- Practice Quiz
- Code Completion Exercises

Review Exercises

★★ **R7.1** Consider the following problem description:

> Users place coins in a vending machine and select a product by pushing a button. If the inserted coins are sufficient to cover the purchase price of the product, the product is dispensed and change is given. Otherwise, the inserted coins are returned to the user.

What classes should you use to implement it?

★★ **R7.2** Consider the following problem description:

> Employees receive their biweekly paychecks. They are paid their hourly rates for each hour worked; however, if they worked more than 40 hours per week, they are paid at 150 percent of their regular wage for those overtime hours.

What classes should you use to implement it?

★★★ **R7.3** Consider the following problem description:

> Customers order products from a store. Invoices are generated to list the items and quantities ordered, payments received, and amounts still due. Products are shipped to the shipping address of the customer, and invoices are sent to the billing address.

What classes should you use to implement it?

★★★ **R7.4** Look at the public interface of the `java.lang.System` class and discuss whether or not it is cohesive.

R7.5 Suppose an `Invoice` object contains descriptions of the products ordered, and the billing and shipping addresses of the customer. Draw a UML diagram showing the dependencies between the classes `Invoice`, `Address`, `Customer`, and `Product`.

R7.6 Suppose a vending machine contains products, and users insert coins into the vending machine to purchase products. Draw a UML diagram showing the dependencies between the classes `VendingMachine`, `Coin`, and `Product`.

R7.7 On which classes does the class `Integer` in the standard library depend?

R7.8 On which classes does the class `Rectangle` in the standard library depend?

R7.9 Classify the methods of the class `Scanner` that are used in this book as accessors and mutators.

★ **R7.10** Classify the methods of the class `Rectangle` as accessors and mutators.

★ **R7.11** Which of the following classes are immutable?

a. `Rectangle`
b. `String`
c. `Random`

★ **R7.12** Which of the following classes are immutable?

a. `PrintStream`
b. `Date`
c. `Integer`

★ **R7.13** What side effect, if any, do the following three methods have:

```
public class Coin
{
   . . .
   public void print()
   {
      System.out.println(name + " " + value);
   }

   public void print(PrintStream stream)
   {
      stream.println(name + " " + value);
   }

   public String toString()
   {
      return name + " " + value;
   }
}
```

★ **R7.14** Ideally, a method should have no side effects. Can you write a program in which no method has a side effect? Would such a program be useful?

★★ **R7.15** Write preconditions for the following methods. Do not implement the methods.

a. `public static double sqrt(double x)`
b. `public static String romanNumeral(int n)`
c. `public static double slope(Line2D.Double a)`
d. `public static String weekday(int day)`

★★ **R7.16** What preconditions do the following methods from the standard Java library have?

a. `Math.sqrt`
b. `Math.tan`
c. `Math.log`
d. `Math.pow`
e. `Math.abs`

★★ **R7.17** What preconditions do the following methods from the standard Java library have?

a. `Integer.parseInt(String s)`
b. `StringTokenizer.nextToken()`
c. `Random.nextInt(int n)`
d. `String.substring(int m, int n)`

★★★ **R7.18** When a method is called with parameters that violate its precondition(s), it can terminate (by throwing an exception or an assertion error), or it can return to its caller. Give two examples of library methods (standard or the library methods used in this book) that return some result to their callers when called with invalid parameters, and give two examples of library methods that terminate.

★★ **R7.19** Consider a `CashRegister` class with methods

- `public void enterPayment(int coinCount, Coin coinType)`
- `public double getTotalPayment()`

Give a reasonable postcondition of the `enterPayment` method. What preconditions would you need so that the `CashRegister` class can ensure that postcondition?

★★ **R7.20** Consider the following method that is intended to swap the values of two floating-point numbers:

```
public static void falseSwap(double a, double b)
{
   double temp = a;
   a = b;
   b = temp;
}

public static void main(String[] args)
{
   double x = 3;
   double y = 4;
   falseSwap(x, y);
   System.out.println(x + " " + y);
}
```

Why doesn't the method swap the contents of `x` and `y`?

★★★ **R7.21** How can you write a method that swaps two floating-point numbers?
Hint: `Point2D.Double`.

★★ **R7.22** Draw a memory diagram that shows why the following method can't swap two `BankAccount` objects:

```
public static void falseSwap(BankAccount a, BankAccount b)
{
   BankAccount temp = a;
   a = b;
   b = temp;
}
```

R7.23 Consider an enhancement of the `Die` class of Chapter 5 with a static variable

```
public class Die
{
   private int sides;
   private static Random generator = new Random();
   public Die(int s) { . . . }
   public int cast() { . . . }
}
```

Draw a memory diagram that shows three dice:

```
Die d4 = new Die(4);
Die d6 = new Die(6);
Die d8 = new Die(8);
```

Be sure to indicate the values of the `sides` and `generator` variables.

R7.24 Try compiling the following program. Explain the error message that you get.

```
public class Print13
{
   public void print(int x)
   {
      System.out.println(x);
   }

   public static void main(String[] args)
   {
      int n = 13;
      print(n);
   }
}
```

R7.25 Look at the methods in the `Integer` class. Which are static? Why?

R7.26 Look at the methods in the `String` class (but ignore the ones that take a parameter of type `char[]`). Which are static? Why?

R7.27 The `in` and `out` variables of the `System` class are public static variables of the `System` class. Is that good design? If not, how could you improve on it?

R7.28 In the following class, the variable `n` occurs in multiple scopes. Which declarations of `n` are legal and which are illegal?

```
public class X
{
   private int n;

   public int f()
   {
      int n = 1;
      return n;
   }

   public int g(int k)
   {
      int a;
      for (int n = 1; n <= k; n++)
         a = a + n;
      return a;
   }

   public int h(int n)
   {
      int b;
      for (int n = 1; n <= 10; n++)
         b = b + n;
      return b + n;
   }

   public int k(int n)
   {
      if (n < 0)
      {
         int k = -n;
         int n = (int) (Math.sqrt(k));
         return n;
      }
      else return n;
   }

   public int m(int k)
   {
      int a;
      for (int n = 1; n <= k; n++)
         a = a + n;
      for (int n = k; n >= 1; n++)
         a = a + n;
      return a;
   }
}
```


★★ **R7.29** Every Java program can be rewritten to avoid `import` statements. Explain how, and rewrite `RectangleComponent.java` from Chapter 2 to avoid `import` statements.

★ **R7.30** What is the default package? Have you used it before this chapter in your programming?

★★T **R7.31** What does JUnit do when a test method throws an exception? Try it out and report your findings.

Programming Exercises

★★ **P7.1** Implement the `Coin` class described in Section 7.2. Modify the `CashRegister` class so that coins can be added to the cash register, by supplying a method

```
void enterPayment(int coinCount, Coin coinType)
```

The caller needs to invoke this method multiple times, once for each type of coin that is present in the payment.

★★ **P7.2** Modify the `giveChange` method of the `CashRegister` class so that it returns the number of coins of a particular type to return:

```
int giveChange(Coin coinType)
```

The caller needs to invoke this method for each coin type, in decreasing value.

★ **P7.3** Real cash registers can handle both bills and coins. Design a single class that expresses the commonality of these concepts. Redesign the `CashRegister` class and

provide a method for entering payments that are described by your class. Your primary challenge is to come up with a good name for this class.

★ **P7.4** Enhance the `BankAccount` class by adding preconditions for the constructor and the `deposit` method that require the `amount` parameter to be at least zero, and a precondition for the `withdraw` method that requires `amount` to be a value between 0 and the current balance. Use assertions to test the preconditions.

★★ **P7.5** Write static methods

- `public static double sphereVolume(double r)`
- `public static double sphereSurface(double r)`
- `public static double cylinderVolume(double r, double h)`
- `public static double cylinderSurface(double r, double h)`
- `public static double coneVolume(double r, double h)`
- `public static double coneSurface(double r, double h)`

that compute the volume and surface area of a sphere with radius `r`, a cylinder with circular base with radius `r` and height `h`, and a cone with circular base with radius `r` and height `h`. Place them into a class `Geometry`. Then write a program that prompts the user for the values of `r` and `h`, calls the six methods, and prints the results.

★★ **P7.6** Solve Exercise P7.5 by implementing classes `Sphere`, `Cylinder`, and `Cone`. Which approach is more object-oriented?

★★ **P7.7** Modify the grade book application of How To 6.1 so that it can deal with multiple students. First, ask the user for all student names. Then read in the scores for all quizzes, prompting for the score of each student. Finally, print the names of all students and their final scores. Use a single class and only static methods.

★★★ **P7.8** Repeat Exercise P7.7, using multiple classes. Modify the `GradeBook` class so that it collects objects of type `Student`. Each such object should have a list of scores.

★★★ **P7.9** Write methods

```
public static double perimeter(Ellipse2D.Double e);
public static double area(Ellipse2D.Double e);
```

that compute the area and the perimeter of the ellipse `e`. Add these methods to a class `Geometry`. The challenging part of this assignment is to find and implement an accurate formula for the perimeter. Why does it make sense to use a static method in this case?

★★ **P7.10** Write methods

```
public static double angle(Point2D.Double p, Point2D.Double q)
public static double slope(Point2D.Double p, Point2D.Double q)
```

that compute the angle between the x-axis and the line joining two points, measured in degrees, and the slope of that line. Add the methods to the class `Geometry`. Supply suitable preconditions. Why does it make sense to use a static method in this case?

★★★ **P7.11** Write methods

```
public static boolean isInside(Point2D.Double p, Ellipse2D.Double e)
public static boolean isOnBoundary(Point2D.Double p, Ellipse2D.Double e)
```

that test whether a point is inside or on the boundary of an ellipse. Add the methods to the class `Geometry`.

★ **P7.12** Write a method

```
public static int readInt(
      Scanner in, String prompt, String error, int min, int max)
```

that displays the prompt string, reads an integer, and tests whether it is between the minimum and maximum. If not, print an error message and repeat reading the input. Add the method to a class `Input`.

★★ **P7.13** Consider the following algorithm for computing x^n for an integer n. If n < 0, x^n is $1/x^{-n}$. If n is positive and even, then $x^n = (x^{n/2})^2$. If n is positive and odd, then $x^n = x^{n-1} \cdot x$. Implement a static method `double intPower(double x, int n)` that uses this algorithm. Add it to a class called `Numeric`.

★★ **P7.14** Improve the `Needle` class of Chapter 5. Turn the `generator` variable into a static variable so that all needles share a single random number generator.

★★ **P7.15** Implement a `Coin` and `CashRegister` class as described in Exercise P7.1. Place the classes into a package called `money`. Keep the `CashRegisterTester` class in the default package.

★ **P7.16** Place a `BankAccount` class in a package whose name is derived from your e-mail address, as described in Section 7.9. Keep the `BankAccountTester` class in the default package.

★★T **P7.17** Provide a JUnit test class `BankTest` with three test methods, each of which tests a different method of the `Bank` class in Chapter 6.

★★T **P7.18** Provide JUnit test class `TaxReturnTest` with three test methods that test different tax situations for the `TaxReturn` class in Chapter 4.

★G **P7.19** Write methods

- `public static void drawH(Graphics2D g2, Point2D.Double p);`
- `public static void drawE(Graphics2D g2, Point2D.Double p);`
- `public static void drawL(Graphics2D g2, Point2D.Double p);`
- `public static void drawO(Graphics2D g2, Point2D.Double p);`

that show the letters H, E, L, O on the graphics window, where the point `p` is the top-left corner of the letter. Then call the methods to draw the words "HELLO" and "HOLE" on the graphics display. Draw lines and ellipses. Do not use the `drawString` method. Do not use `System.out`.

★★G **P7.20** Repeat Exercise P7.17 by designing classes `LetterH`, `LetterE`, `LetterL`, and `LetterO`, each with a constructor that takes a `Point2D.Double` parameter (the top-left corner) and a method `draw(Graphics2D g2)`. Which solution is more object-oriented?

Programming Projects

Project 7.1 Implement a program that prints paychecks for a group of student assistants. Deduct federal and Social Security taxes. (You may want to use the tax computation used in Chapter 4. Find out about Social Security taxes on the Internet.) Your program should prompt for the names, hourly wages, and hours worked of each student.

Project 7.2 For faster sorting of letters, the United States Postal Service encourages companies that send large volumes of mail to use a bar code denoting the ZIP code (see Figure 7).

The encoding scheme for a five-digit ZIP code is shown in Figure 8. There are full-height frame bars on each side. The five encoded digits are followed by a check digit, which is computed as follows: Add up all digits, and choose the check digit to make the sum a multiple of 10. For example, the sum of the digits in the ZIP code 95014 is 19, so the check digit is 1 to make the sum equal to 20.

Each digit of the ZIP code, and the check digit, is encoded according to the table at right, where 0 denotes a half bar and 1 a full bar. Note that they represent all combinations of two full and three half bars. The digit can be computed easily from the bar code using the column weights 7, 4, 2, 1, 0. For example, 01100 is

$$0 \cdot 7 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 + 0 \cdot 0 = 6$$

The only exception is 0, which would yield 11 according to the weight formula.

	7	4	2	1	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	1	0	0	0	0
8	1	0	0	1	1
9	1	0	1	0	0
0	1	1	0	0	0

Write a program that asks the user for a ZIP code and prints the bar code. Use : for half bars, | for full bars. For example, 95014 becomes

```
||:|:::|:|:||::::::||:|::|:::|||
```

(Alternatively, write a graphical application that draws real bars.)

Your program should also be able to carry out the opposite conversion: Translate bars into their ZIP code, reporting any errors in the input format or a mismatch of the digits.

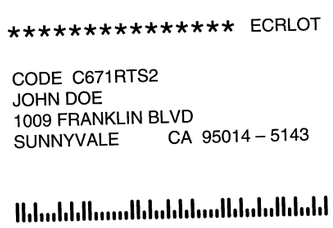


Figure 7 A Postal Bar Code

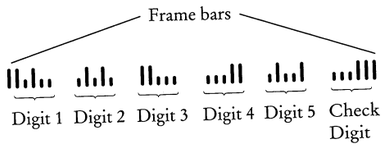


Figure 8 Encoding for Five-Digit Bar Codes

Answers to Self-Check Questions

1. Look for nouns in the problem description.
2. Yes (`ChessBoard`) and no (`MovePiece`).
3. Some of its features deal with payments, others with coin values.
4. None of the coin operations require the `CashRegister` class.
5. If a class doesn't depend on another, it is not affected by interface changes in the other class.
6. It is an accessor—calling `substring` doesn't modify the string on which the method is invoked. In fact, all methods of the `String` class are accessors.
7. No—`translate` is a mutator.
8. It is a side effect; this kind of side effect is common in object-oriented programming.
9. Yes—the method affects the state of the `Scanner` parameter.
10. Then you don't have to worry about checking for invalid values—it becomes the caller's responsibility.
11. No—you can take any action that is convenient for you.
12. `Math m = new Math(); y = m.sqrt(x);`
13. You cannot add a method to the `ArrayList` class—it is a class in the standard Java library that you cannot modify.
14. `System.in` and `System.out`.
15. Yes, it works. Static methods can access static variables of the same class. But it is a terrible idea. As your programming tasks get more complex, you will want to use objects and classes to organize your programs.
16. Yes. The scopes are disjoint.
17. It starts at the beginning of the class and ends at the end of the class.
18. (a) No; (b) Yes; (c) Yes; (d) No
19. No—you simply use fully qualified names for all other classes, such as `java.util.Random` and `java.awt.Rectangle`.
20. `/home/me/cs101/hw1/problem1` or, on Windows, `c:\Users\Me\cs101\hw1\problem1`.
21. Here is one possible answer.

```
public class EarthquakeTest
{
   @Test public void testLevel4()
   {
      Earthquake quake = new Earthquake(4);
      Assert.assertEquals("Felt by many people, no destruction",
            quake.getDescription());
   }
}
```

22. It is a tolerance threshold for comparing floating-point numbers. We want the equality test to pass if there is a small roundoff error.