**1.5   Input and Output**

In this section we extend the set of simple abstractions (command-line input and standard output) that we have been using as the interface between our Java programs and the outside world to include *standard input*, *standard drawing*, and *standard audio*. Standard input makes it convenient for us to write programs that process arbitrary amounts of input and to interact with our programs; standard draw makes it possible for us to work with graphics; and standard audio adds sound.

**Bird's-eye view.**

A Java program takes input values from the command line and prints a string of characters as output. By default, both command-line arguments and standard output are associated with an application that takes commands, which we refer to as the *terminal window*. Here are some instructions for using the command line on your system. [ [Mac](https://introcs.cs.princeton.edu/java/mac/index.html#CLI) · [Windows](https://introcs.cs.princeton.edu/java/windows/index.html#CLI) · [Linux](https://introcs.cs.princeton.edu/java/15inout/linux-cmd.html) ]

* *Command-line arguments.* All of our classes have a main() method that takes a String array args[] as argument. That array is the sequence of command-line arguments that we type. If we intend for an argument to be a number, we must use a method such as Integer.parseInt() to convert it from String to the appropriate type.
* *Standard output.* To print output values in our programs, we have been using System.out.println(). Java sends the results to an abstract stream of characters known as *standard output*. By default, the operating system connects standard output to the terminal window. All of the output in our programs so far has been appearing in the terminal window.

[RandomSeq.java](https://introcs.cs.princeton.edu/java/15inout/RandomSeq.java.html) uses this model: It takes a command-line argument *n* and prints to standard output a sequence of *n* random numbers between 0 and 1.

To complete our programming model, we add the following libraries:

* *Standard input.* Read numbers and strings from the user.
* *Standard drawing.* Plot graphics.
* *Standard audio.* Create sound.

**Standard output.**

Java's System.out.print() and System.out.println() methods implement the basic standard output abstraction that we need. Nevertheless, to treat standard input and standard output in a uniform manner (and to provide a few technical improvements), we use similar methods that are defined in our [StdOut](https://introcs.cs.princeton.edu/java/stdlib/javadoc/StdOut.html) library:

Java's print() and println() methods are the ones that you have been using. The printf() method gives us more control over the appearance of the output.

* *Formatted printing basics.* In its simplest form, printf() takes two arguments. The first argument is called the *format string*. It contains a *conversion specification* that describes how the second argument is to be converted to a string for output.

Format strings begin with % and end with a one-letter *conversion code*. The following table summarizes the most frequently used codes:

* *Format string.* The format string can contain characters in addition to those for the conversion specification. The conversion specification is replaced by the argument value (converted to a string as specified) and all remaining characters are passed through to the output.
* *Multiple arguments.* The printf() function can take more than two arguments. In this case, the format string will have an additional conversion specification for each additional argument.

Here is more documentation on [printf format string syntax](http://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html#syntax).

**Standard input.**

Our [StdIn](https://introcs.cs.princeton.edu/java/stdlib/javadoc/StdIn.html) library takes data from a *standard input stream* that contains a sequence of values separated by whitespace. Each value is a string or a value from one of Java's primitive types. One of the key features of the standard input stream is that your program *consumes* values when it reads them. Once your program has read a value, it cannot back up and read it again. The library is defined by the following API:

We now consider several examples in detail.

* *Typing input.* When you use the java command to invoke a Java program from the command line, you actually are doing three things: (1) issuing a command to start executing your program, (2) specifying the values of the command-line arguments, and (3) beginning to define the standard input stream. The string of characters that you type in the terminal window after the command line is the standard input stream. For example, [AddInts.java](https://introcs.cs.princeton.edu/java/15inout/AddInts.java.html) takes a command-line argument n, then reads n numbers from standard input and adds them, and prints the result to standard output:
* *Input format.* If you type abc or 12.2 or true when StdIn.readInt() is expecting an int, then it will respond with an InputMismatchException. StdIn treats strings of consecutive whitespace characters as identical to one space and allows you to delimit your numbers with such strings.
* *Interactive user input.* [TwentyQuestions.java](https://introcs.cs.princeton.edu/java/15inout/TwentyQuestions.java.html) is a simple example of a program that interacts with its user. The program generates a random integer and then gives clues to a user trying to guess the number. The fundamental difference between this program and others that we have written is that the user has the ability to change the control flow *while* the program is executing.
* *Processing an arbitrary-size input stream.* Typically, input streams are finite: your program marches through the input stream, consuming values until the stream is empty. But there is no restriction of the size of the input stream. [Average.java](https://introcs.cs.princeton.edu/java/15inout/Average.java.html) reads in a sequence of real numbers from standard input and prints their average.

**Redirection and piping.**

For many applications, typing input data as a standard input stream from the terminal window is untenable because doing so limits our program's processing power by the amount of data that we can type. Similarly, we often want to save the information printed on the standard output stream for later use. We can use operating system mechanisms to address both issues.

* *Redirecting standard output to a file.* By adding a simple directive to the command that invokes a program, we can *redirect* its standard output to a file, either for permanent storage or for input to some other program at a later time. For example, the command

specifies that the standard output stream is not to be printed in the terminal window, but instead is to be written to a text file named data.txt. Each call to StdOut.print() or StdOut.println() appends text at the end of that file. In this example, the end result is a file that contains 1,000 random values.

* *Redirecting standard output from a file.* Similarly, we can redirect standard input so that StdIn reads data from a file instead of the terminal window. For example, the command

reads a sequence of numbers from the file data.txt and computes their average value. Specifically, the < symbol is a directive to implement the standard input stream by reading from the file data.txt instead of by waiting for the user to type something into the terminal window. When the program calls StdIn.readDouble(), the operating system reads the value from the file. This facility to redirect standard input from a file enables us to process huge amounts of data from any source with our programs, limited only by the size of the files that we can store.

* *Connecting two programs.* The most flexible way to implement the standard input and standard output abstractions is to specify that they are implemented by our own programs! This mechanism is called *piping*. For example, the following command

specifies that the standard output for RandomSeq and the standard input stream for Average are the *same* stream.

* *Filters.* For many common tasks, it is convenient to think of each program as a filter that converts a standard input stream to a standard output stream in some way, [RangeFilter.java](https://introcs.cs.princeton.edu/java/15inout/RangeFilter.java.html) takes two command-line arguments and prints on standard output those numbers from standard input that fall within the specified range.

Your operating system also provides a number of filters. For example, the sort filter puts the lines on standard input in sorted order:

|  |
| --- |
| % **java RandomSeq 5 | sort**  0.035813305516568916  0.14306638757584322  0.348292877655532103  0.5761644592016527  0.9795908813988247 |

Another useful filter is more, which reads data from standard input and displays it in your terminal window one screenful at a time. For example, if you type

|  |
| --- |
| % **java RandomSeq 1000 | more** |

you will see as many numbers as fit in your terminal window, but more will wait for you to hit the space bar before displaying each succeeding screenful.

**Standard drawing.**

Now we introduce a simple abstraction for producing drawings as output. We imagine an abstract drawing device capable of drawing lines and points on a two-dimensional canvas. The device is capable of responding to the commands that our programs issue in the form of calls to static methods in [StdDraw](https://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html). The primary interface consists of two kinds of methods: *drawing commands* that cause the device to take an action (such as drawing a line or drawing a point) and *control commands* that set parameters such as the pen size or the coordinate scales.

* *Basic drawing commands.* We first consider the drawing commands:

These methods are nearly self-documenting: StdDraw.line(x0, y0, x1, y1) draws a straight line segment connecting the point (*x*0, *y*0) with the point (*x*1, *y*1). StdDraw.point(x, y) draws a spot centered on the point (*x*, *y*). The default coordinate scale is the unit square (all *x*- and *y*-coordinates between 0 and 1). The standard implementation displays the canvas in a window on your computer's screen, with black lines and points on a white background.

*Your first drawing.* The HelloWorld for graphics programming with StdDraw is to draw a triangle with a point inside. [Triangle.java](https://introcs.cs.princeton.edu/java/15inout/Triangle.java.html) accomplishes this with three calls to StdDraw.line() and one call to StdDraw.point().

* *Control commands.* The default canvas size is 512-by-512 pixels and the default coordinate system is the unit square, but we often want to draw plots at different scales. Also, we often want to draw line segments of different thickness or points of different size from the standard. To accommodate these needs, StdDraw has the following methods:

For example, the two-call sequence

|  |
| --- |
| StdDraw.setXscale(x0, x1);  StdDraw.setYscale(y0, y1); |

sets the drawing coordinates to be within a *bounding box* whose lower-left corner is at (*x*0, *y*0) and whose upper-right corner is at (*x*1, *y*1).

* + *Filtering data to a standard drawing.* [PlotFilter.java](https://introcs.cs.princeton.edu/java/15inout/PlotFilter.java.html) reads a sequence of points defined by (*x*, *y*) coordinates from standard input and draws a spot at each point. It adopts the convention that the first four numbers on standard input specify the bounding box, so that it can scale the plot.

% **java PlotFilter <** [**USA.txt**](https://introcs.cs.princeton.edu/java/15inout/USA.txt)

* + *Plotting a function graph.* [FunctionGraph.java](https://introcs.cs.princeton.edu/java/15inout/FunctionGraph.java.html) plots the function *y* = sin(4*x*) + sin(20*x*) in the interval (0, π). There are an infinite number of points in the interval, so we have to make do with evaluating the function at a finite number of points within the interval. We *sample* the function by choosing a set of *x*-values, then computing *y*-values by evaluating the function at each *x*-value. Plotting the function by connecting successive points with lines produces what is known as a *piecewise linear approximation*.
* *Outline and filled shapes.* StdDraw also includes methods to draw circles, rectangles, and arbitrary polygons. Each shape defines an outline. When the method name is just the shape name, that outline is traced by the drawing pen. When the method name begins with filled, the named shape is instead filled solid, not traced.

The arguments for circle() define a circle of radius r; the arguments for square() define a square of side length 2r centered on the given point; and the arguments for polygon() define a sequence of points that we connect by lines, including one from the last point to the first point.

* *Text and color.* To annotate or highlight various elements in your drawings, StdDraw includes methods for drawing text, setting the font, and setting the the ink in the pen.

In this code, [java.awt.Font](http://docs.oracle.com/javase/8/docs/api/java/awt/Font.html) and [java.awt.Color](http://docs.oracle.com/javase/8/docs/api/java/awt/Color.html) are abstractions that are implemented with non-primitive types that you will learn about in Section 3.1. Until then, we leave the details to StdDraw. The default ink color is black; the default font is a 16-point plain Serif font.

* *Double buffering.* StdDraw supports a powerful computer graphics feature known as *double buffering*. When double buffering is enabled by calling enableDoubleBuffering(), all drawing takes place on the *offscreen canvas*. The offscreen canvas is not displayed; it exists only in computer memory. Only when you call show() does your drawing get copied from the offscreen canvas to the *onscreen canvas*, where it is displayed in the standard drawing window. You can think of double buffering as collecting all of the lines, points, shapes, and text that you tell it to draw, and then drawing them all simultaneously, upon request. One reason to use double buffering is for efficiency when performing a large number of drawing commands.
* *Computer animations.* Our most important use of double buffering is to produce *computer animations*, where we create the illusion of motion by rapidly displaying static drawings. We can produce animations by repeating the following four steps:
  + Clear the offscreen canvas.
  + Draw objects on the offscreen
  + Copy the offscreen canvas to the onscreen canvas.
  + Wait for a short while.

In support of these steps, the StdDraw has several methods:

The "Hello, World" program for animation is to produce a black ball that appears to move around on the canvas, bouncing off the boundary according to the laws of elastic collision. Suppose that the ball is at position (*x*, *y*) and we want to create the impression of having it move to a new position, say (*x* + 0.01, *y* + 0.02). We do so in four steps:

* + Clear the offscreen canvas to white.
  + Draw a black ball at the new position on the offscreen canvas.
  + Copy the offscreen canvas to the onscreen canvas.
  + Wait for a short while.

To create the illusion of movement, [BouncingBall.java](https://introcs.cs.princeton.edu/java/15inout/BouncingBall.java.html) iterates these steps for a whole sequence of positions of the ball.

* *Images.* Our standard draw library supports drawing pictures as well as geometric shapes. The command StdDraw.picture(x, y, filename) plots the image in the given filename (either JPEG, GIF, or PNG format) on the canvas, centered on (x, y). [BouncingBallDeluxe.java](https://introcs.cs.princeton.edu/java/15inout/BouncingBallDeluxe.java.html) illustrates an example where the bouncing ball is replaced by an image of a tennis ball.
* *User interaction.* Our standard draw library also includes methods so that the user can interact with the window using the mouse.

|  |
| --- |
| double mouseX() return x-coordinate of mouse  double mouseY() return y-coordinate of mouse  boolean mousePressed() is the mouse currently being pressed? |

* + *A first example.* [MouseFollower.java](https://introcs.cs.princeton.edu/java/15inout/MouseFollower.java.html) is the HelloWorld of mouse interaction. It draws a blue ball, centered on the location of the mouse. When the user holds down the mouse button, the ball changes color from blue to cyan.
  + *A simple attractor.* [OneSimpleAttractor.java](https://introcs.cs.princeton.edu/java/15inout/OneSimpleAttractor.java.html) simulates the motion of a blue ball that is attracted to the mouse. It also accounts for a drag force.
  + *Many simple attractors.* [SimpleAttractors.java](https://introcs.cs.princeton.edu/java/15inout/SimpleAttractors.java.html) simulates the motion of 20 blue balls that are attracted to the mouse. It also accounts for a drag force. When the user clicks, the balls disperse randomly.
  + *Springs.* [Springs.java](https://introcs.cs.princeton.edu/java/15inout/Springs.java.html) implements a spring system.

**Standard audio.**

[StdAudio](https://introcs.cs.princeton.edu/java/stdlib/javadoc/StdAudio.html) is a library that you can use to play and manipulate sound files. It allows you to play, manipulate and synthesize sound.

We introduce some some basic concepts behind one of the oldest and most important areas of computer science and scientific computing: *digital signal processing*.

* *Concert A.* Concert A is a sine wave, scaled to oscillate at a frequency of 440 times per second. The function sin(*t*) repeats itself once every 2π units on the *x*-axis, so if we measure *t* in seconds and plot the function sin(2π*t* × 440) we get a curve that oscillates 440 times per second. The amplitude (*y*-value) corresponds to the volume. We assume it is scaled to be between −1 and +1.
* *Other notes.* A simple mathematical formula characterizes the other notes on the chromatic scale. They are divided equally on a logarithmic (base 2) scale: there are twelve notes on the chromatic scale, and we get the *i*th note above a given note by multiplying its frequency by the (*i*/12)th power of 2.

When you double or halve the frequency, you move up or down an octave on the scale. For example 880 hertz is one octave above concert A and 110 hertz is two octaves below concert A.

* *Sampling.* For digital sound, we represent a curve by sampling it at regular intervals, in precisely the same manner as when we plot function graphs. We sample sufficiently often that we have an accurate representation of the curve—a widely used sampling rate is 44,100 samples per second. It is that simple: we represent sound as an array of numbers (real numbers that are between −1 and +1).

|  |  |  |
| --- | --- | --- |
|  |  |  |

* For example, the following code fragment plays concert A for 10 seconds.

|  |
| --- |
| int SAMPLING\_RATE = 44100;  double hz = 440.0;  double duration = 10.0;  int n = (int) (SAMPLING\_RATE \* duration);  double[] a = new double[n+1];  for (int i = 0; i <= n; i++) {  a[i] = Math.sin(2 \* Math.PI \* i \* hz / SAMPLING\_RATE);  }  StdAudio.play(a); |

* *Play that tune.* [PlayThatTune.java](https://introcs.cs.princeton.edu/java/15inout/PlayThatTune.java.html) is an example that shows how easily we can create music with StdAudio. It takes notes from standard input, indexed on the chromatic scale from concert A, and plays them on standard audio.

**Exercises**

1. Write a program [MaxMin.java](https://introcs.cs.princeton.edu/java/15inout/MaxMin.java.html) that reads in integers (as many as the user enters) from standard input and prints out the maximum and minimum values.
2. Write a program [Stats.java](https://introcs.cs.princeton.edu/java/15inout/Stats.java.html) that takes an integer command-line argument *n*, reads *n* floating-point numbers from standard input, and prints their *mean* (average value) and *sample standard deviation* (square root of the sum of the squares of their differences from the average, divided by *n*−1).
3. Write a program [LongestRun.java](https://introcs.cs.princeton.edu/java/15inout/LongestRun.java.html) that reads in a sequence of integers and prints out both the integer that appears in a longest consecutive run and the length of the run. For example, if the input is 1 2 2 1 5 1 1 7 7 7 7 1 1, then your program should print Longest run: 4 consecutive 7s.
4. Write a program [WordCount.java](https://introcs.cs.princeton.edu/java/15inout/WordCount.java.html) that reads in text from standard input and prints out the number of words in the text. For the purpose of this exercise, a word is a sequence of non-whitespace characters that is surrounded by whitespace.
5. Write a program [Closest.java](https://introcs.cs.princeton.edu/java/15inout/Closest.java.html) that takes three floating-point command-line arguments *x*,*y*,*z*

, reads from standard input a sequence of point coordinates (*xi*,*yi*,*zi*), and prints the coordinates of the point closest to (*x*,*y*,*z*). Recall that the square of the distance between (*x*,*y*,*z*) and (*xi*,*yi*,*zi*) is (*x*−*xi*)2+(*y*−*yi*)2+(*z*−*zi*)2

1. . For efficiency, do not use Math.sqrt() or Math.pow().
2. Given the positions and masses of a sequence of objects, write a program to compute their center-of-mass or *centroid.* The centroid is the average position of the *n* objects, weighted by mass. If the positions and masses are given by (*xi*, *yi*, *mi*), then the centroid (*x*, *y*, *m*) is given by:

|  |
| --- |
| m = *m1* + *m2* + ... + *mn*  x = (*m1x1* + ... + *mnxn*) / m  y = (*m1y1* + ... + *mnyn*) / m |

Write a program Centroid.java that reads in a sequence of positions and masses (*xi*, *yi*, *mi*) from standard input and prints out their center of mass (*x*, *y*, *m*). *Hint*: model your program after [Average.java](https://introcs.cs.princeton.edu/java/15inout/Average.java.html).

1. Write a program [Checkerboard.java](https://introcs.cs.princeton.edu/java/15inout/Checkerboard.java.html) that takes a command-line argument n and plots an n-by-n checkerboard with red and black squares. Color the lower-left square red.

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. Write a program [Rose.java](https://introcs.cs.princeton.edu/java/15inout/Rose.java.html) that takes a command-line argument n and plots a rose with n petals (if n is odd) or 2n petals (if n is even) by plotting the polar coordinates (r, θ) of the function *r = sin(n × θ)* for θ ranging from 0 to 2π radians. Below is the desired output for n = 4, 7, and 8.

|  |
| --- |
|  |

1. Write a program [Banner.java](https://introcs.cs.princeton.edu/java/15inout/Banner.java.html) that takes a string s from the command line and display it in banner style on the screen, moving from left to right and wrapping back to the beginning of the string as the end is reached. Add a second command-line argument to control the speed.
2. Write a program Circles.java that draws filled circles of random size at random positions in the unit square, producing images like those below. Your program should take four command-line arguments: the number of circles, the probability that each circle is black, the minimum radius, and the maximum radius.

**Creative Exercises**

1. **Spirographs.** Write a program [Spirograph.java](https://introcs.cs.princeton.edu/java/15inout/Spirograph.java.html) that takes three command-line arguments R, r, and a and draws the resulting spirograph. A [spirograph](http://www.math.dartmouth.edu/%7Edlittle/java/SpiroGraph) (technically, an epicycloid) is a curve formed by rolling a circle of radius r around a larger fixed circle or radius R. If the pen offset from the center of the rolling circle is (r+a), then the equation of the resulting curve at time t is given by

|  |
| --- |
| x(t) = (R+r)\*cos(t) - (r+a)\*cos(((R+r)/r)\*t)  y(t) = (R+r)\*sin(t) - (r+a)\*sin(((R+r)/r)\*t) |

Such curves were popularized by a best-selling toy that contains discs with gear teeth on the edges and small holes that you could put a pen in to trace spirographs.

For a dramatic 3d effect, draw a circular image, e.g., [earth.gif](https://introcs.cs.princeton.edu/java/15inout/earth.gif) instead of a dot, and show it rotating over time. Here's a picture of the resulting [spirograph](https://introcs.cs.princeton.edu/java/15inout/spirograph.png) when R = 180, r = 40, and a = 15.

1. **Clock.** Write a program [Clock.java](https://introcs.cs.princeton.edu/java/15inout/Clock.java.html) that displays an animation of the second, minute, and hour hands of an analog clock. Use the method StdDraw.show(1000) to update the display roughly once per second.

*Hint*: this may be one of the rare times when you want to use the % operator with a double - it works the way you would expect.

1. **Oscilloscope.** Write a program [Oscilloscope.java](https://introcs.cs.princeton.edu/java/15inout/Oscilloscope.java.html) to simulate the output of an oscilloscope and produce Lissajous patterns. These patterns are named after the French physicist, Jules A. Lissajous, who studied the patterns that arise when two mutually perpendicular periodic disturbances occur simultaneously. Assume that the inputs are sinusoidal, so tha the following parametric equations describe the curve:

|  |
| --- |
| x = Ax sin (wxt + θx)  y = Ay sin (wyt + θy)  Ax, Ay = amplitudes  wx, wy = angular velocity  θx, θy = phase factors |

Take the six parameters Ax, wx, θx, θy, wy, and θy from the command line.

For example, the first image below has Ax = Ay = 1, wx = 2, wy = 3, θx = 20 degrees, θy = 45 degrees. The other has parameters (1, 1, 5, 3, 30, 45)

|  |  |
| --- | --- |
|  |  |

**Web Exercises**

1. **Word and line count.** Modify [WordCount.java](https://introcs.cs.princeton.edu/java/15inout/WordCount.java.html) so that reads in text from standard input and prints out the number of characters, words, and lines in the text.
2. **Rainfall problem.** Write a program Rainfall.java that reads in nonnegative integers (representing rainfall) one at a time until 999999 is entered, and then prints out the average of value (not including 999999).
3. **Remove duplicates.** Write a program Duplicates.java that reads in a sequence of integers and prints back out the integers, except that it removes repeated values if they appear consecutively. For example, if the input is 1 2 2 1 5 1 1 7 7 7 7 1 1, your program should print out 1 2 1 5 1 7 1.
4. **Run length encoding.** Write a program [RunLengthEncoder.java](https://introcs.cs.princeton.edu/java/15inout/RunLengthEncoder.java.html) that encodes a binary input using run length encoding. Write a program RunLengthDecoder.java that decodes a run length encoded message.
5. **Head and tail.** Write programs Head.java and Tail.java that take an integer command line input N and print out the first or last N lines of the given file. (Print the whole file if it consists of <= N lines of text.)
6. **Print a random word.** Read a list of N words from standard input, where N is unknown ahead of time, and print out one of the N words uniformly at random. Do not store the word list. Instead, use Knuth's method: when reading in the ith word, select it with probability 1/i to be the selected word, replacing the previous champion. Print out the word that survives after reading in all of the data.
7. **Caesar cipher.** Julius Caesar sent secret messages to Cicero using a scheme that is now known as a *Caesar cipher*. Each letter is replaced by the letter k positions ahead of it in the alphabet (and you wrap around if needed). The table below gives the Caesar cipher when k = 3.

|  |
| --- |
| Original: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  Caesar: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C |

For example the message "VENI, VIDI, VICI" is converted to "YHQL, YLGL, YLFL". Write a program Caesar.java that takes a command-line argument k and applies a Caesar cipher with shift = k to a sequence of letters read from standard input. If a letter is not an uppercase letter, simply print it back out.

1. **Caesar cipher decoding.** How would you decode a message encrypted using a Caesar cipher? *Hint*: you should not need to write any more code.
2. **Parity check.** A Boolean matrix has the *parity property* when each row and each column has an even sum. This is a simple type of error-correcting code because if one bit is corrupted in transmission (bit is flipped from 0 to 1 or from 1 to 0) it can be detected and repaired. Here's a 4 x 4 input file which has the parity property:

|  |
| --- |
| 1 0 1 0  0 0 0 0  1 1 1 1  0 1 0 1 |

Write a program ParityCheck.java that takes an integer N as a command line input and reads in an N-by-N Boolean matrix from standard input, and outputs if (i) the matrix has the parity property, or (ii) indicates which single corrupted bit (i, j) can be flipped to restore the parity property, or (iii) indicates that the matrix was corrupted (more than two bits would need to be changed to restore the parity property). Use as little internal storage as possible. Hint: you do not even have to store the matrix!

1. **Takagi's function.** Plot Takagi's function: everywhere continuous, nowhere differentiable.
2. **Hitchhiker problem.** You are interviewing N candidates for the sole position of American Idol. Every minute you get to see a new candidate, and you have one minute to decide whether or not to declare that person the American Idol. You may not change your mind once you finish interviewing the candidate. Suppose that you can immediately rate each candidate with a single real number between 0 and 1, but of course, you don't know the rating of the candidates not yet seen. Devise a strategy and write a program AmericanIdol that has at least a 25% chance of picking the best candidate (assuming the candidates arrive in random order), reading the 500 data values from standard input.

*Solution:* interview for N/2 minutes and record the rating of the best candidate seen so far. In the next N/2 minutes, pick the first candidate that has a higher rating than the recorded one. This yields at least a 25% chance since you will get the best candidate if the second best candidate arrives in the first N/2 minutes, and the best candidate arrives in the final N/2 minutes. This can be improved slightly to 1/e = 0.36788 by using essentially the same strategy, but switching over at time N/e.

1. **Nested diamonds.** Write a program Diamonds.java that takes a command line input N and plots N nested squares and diamonds. Below is the desired output for N = 3, 4, and 5.

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. **Regular polygons.** Create a function to plot an N-gon, centered on (x, y) of size length s. Use the function to draws nested polygons like the picture below.
2. **Bulging squares.** Write a program BulgingSquares.java that draws the following optical illusion from [Akiyoshi Kitaoka](http://www.ritsumei.ac.jp/%7Eakitaoka/index-e.html) The center appears to bulge outwards even though all squares are the same size.
3. **Spiraling mice.** Suppose that N mice that start on the vertices of a regular polygon with N sides, and they each head toward the nearest other mouse (in counterclockwise direction) until they all meet. Write a program to draw the logarithmic spiral paths that they trace out by drawing nested N-gons, rotated and shrunk as in [this animation](http://mathworld.wolfram.com/Whirl.html).
4. **Spiral.** Write a program to draw a spiral like the one below.

|  |
| --- |
|  |

1. **Globe.** Write a program [Globe.java](https://introcs.cs.princeton.edu/java/15inout/Globe.java.html) that takes a real command-line argument α and plots a globe-like pattern with parameter α. Plot the polar coordinates (r, θ) of the function *f(θ) = cos(α × θ)* for θ ranging from 0 to 7200 degrees. Below is the desired output for α = 0.8, 0.9, and 0.95.

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. **Drawing strings.** Write a program [RandomText.java](https://introcs.cs.princeton.edu/java/15inout/RandomText.java.html) that takes a string s and an integer N as command line inputs, and writes the string N times at a random location, and in a random color.

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. **2D random walk.** Write a program [RandomWalk.java](https://introcs.cs.princeton.edu/java/15inout/RandomWalk.java.html) to simulate a 2D random walk and animate the results. Start at the center of a 2N-by-2N grid. The current location is displayed in blue; the trail in white.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

1. **Rotating table.** You are seated at a rotating square table (like a lazy Susan), and there are four coins placed in the four corners of the table. Your goal is to flip the coins so that they are either all heads or all tails, at which point a bell rings to notify you that you are done. You may select any two of them, determine their orientation, and (optionally) flip either or both of them over. To make things challenging, you are blindfolded, and the table is spun after each time you select two coins. Write a program RotatingTable.java that initializes the coins to random orientations. Then, it prompts the user to select two positions (1-4), and identifies the orientation of each coin. Next, the user can specify which, if any of the two coins to flip. The process repeats until the user solves the puzzle.
2. **Rotating table solver.** Write another program RotatingTableSolver.java to solve the rotating table puzzle. One effective strategy is to choose two coins at random and flip them to heads. However, if you get really unlucky, this could take an arbitrary number of steps. Goal: devise a strategy that always solves the puzzle in at most 5 steps.
3. **Hex.** [Hex](http://mathworld.wolfram.com/GameofHex.html) is a two-player board game popularized by John Nash while a graduate student at Princeton University, and later commercialized by Parker Brothers. It is played on a hexagonal grid in the shape of an [11-by-11 diamond](http://www.wikipedia.org/wiki/Hex_%28game%29). Write a program Hex.java that draws the board.
4. **Projectile motion with drag.** Write a program [BallisticMotion.java](https://introcs.cs.princeton.edu/java/15inout/BallisticMotion.java.html) that plots the trajectory of a ball that is shot with velocity v at an angle theta. Account for gravitational and drag forces. Assume that the drag force is proportional to the square of the velocity. Using Newton's equations of motions and the Euler-Cromer method, update the position, velocity, and acceleration according to the following equations:

|  |
| --- |
| v = sqrt(vx\*vx + vy\*vy)  ax = - C \* v \* vx ay = -G - C \* v \* vy  vx = vx + ax \* dt vy = vy + ay \* dt  x = x + vx \* dt y = y + vy \* dt |

Use G = 9.8, C = 0.002, and set the initial velocity to 180 and the angle to 60 degrees.

1. **Heart.** Write a program [Heart.java](https://introcs.cs.princeton.edu/java/15inout/Heart.java.html) to draw a pink heart: Draw a diamond, then draw two circles to the upper left and upper right sides.
2. **Changing square.** Write a program that draws a square and changes its color each second.
3. **Simple harmonic motion.** Repeat the previous exercise, but animate the Lissajous patterns as in [this applet](http://www.dcs.napier.ac.uk/%7Eandrew/shm/liss3by2.html). Ex: A = B = wx = wy = 1, but at each time t draw 100 (or so) points with φx ranging from 0 to 720 degrees, and φx ranging from 0 to 1080 degrees.
4. **Bresenham's line drawing algorithm.** To plot a line segment from (x1, y1) to (x2, y2) on a monitor, say 1024-by-1024, you need to make a discrete approximation to the continuous line and determine exactly which pixels to turn on. [Bresenham's line drawing algorithm](http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html) is a clever solution that works when the slope is between 0 and 1 and x1 < x2.

|  |
| --- |
| int dx = x2 - x1;  int dy = y2 - y1;  int y = y1;  int eps = 0;    for (int x = x1; x <= x2; x++) {  StdDraw.point(x, y);  eps += dy;  if (2\*eps >= dx) {  y++;  eps -= dx;  }  } |

1. Modify Bresenham's algorithm to handle arbitrary line segments.
2. **Miller's madness.** Write a program [Madness.java](https://introcs.cs.princeton.edu/java/15inout/Madness.java.html) to plot the parametric equation:

|  |
| --- |
| x = sin(0.99 t) - 0.7 cos( 3.01 t)  y = cos(1.01 t) + 0.1 sin(15.03 t) |

where the parameter t is in radians. You should get the following [complex picture](https://introcs.cs.princeton.edu/java/15inout/images/madness.png). Experiment by changing the parameters and produce original pictures.

1. **Fay's butterfly.** Write a program [Butterfly.java](https://introcs.cs.princeton.edu/java/15inout/Butterfly.java.html) to plot the polar equation:

|  |
| --- |
| r = e^(cos t) - 2 cos(4t) + (sin(t/12)^5) |

where the parameter t is in radians. You should get an image like the following butterfly-like figure. Experiment by changing the parameters and produce original pictures.

1. **Student database.** The file [students.txt](https://introcs.cs.princeton.edu/java/15inout/students.txt) contains a list of students enrolled in an introductory computer science class at Princeton. The first line contains an integer N that specifies the number of students in the database. Each of the next N lines consists of four pieces of information, separated by whitespace: first name, last name, email address, and section number. The program [Students.java](https://introcs.cs.princeton.edu/java/15inout/Students.java.html) reads in the integer N and then N lines of data of standard input, stores the data in four parallel arrays (an integer array for the section number and string arrays for the other fields). Then, the program prints out a list of students in section 4 and 5.
2. **Shuffling.** In the October 7, 2003 California state runoff election for governor, there were [135 official candidates](https://introcs.cs.princeton.edu/java/datafiles/california-gov.txt). To avoid the natural prejudice against candidates whose names appear at the end of the alphabet (Jon W. Zellhoefer), California election officials sought to order the candidates in random order. Write a program program [Shuffle.java](https://introcs.cs.princeton.edu/java/15inout/Shuffle.java.html) that takes a command-line argument N, reads in N strings from standard input, and prints them back out in shuffled order. (California decided to [randomize the alphabet](http://www.cnn.com/2003/ALLPOLITICS/08/12/calif.recall/) instead of shuffling the candidates. Using this strategy, not all N! possible outcomes are equally likely or even possible! For example, two candidates with very similar last names will always end up next to each other.)
3. **Reverse.** Write a program [Reverse.java](https://introcs.cs.princeton.edu/java/15inout/Reverse.java.html) that reads in an arbitrary number of real values from standard input and prints them in reverse order.
4. **Time series analysis.** This problem investigates two methods for forecasting in time series analysis. Moving average or exponential smoothing.
5. **Polar plots.** Create any of these [polar plots](http://www.mapleapps.com/powertools/calcII/html/L27-polarGraphs.html).
6. **Java games.** Use StdDraw.java to implement one of the games at [javaunlimited.net](http://javaunlimited.net/games/java4k_2006.php).
7. Consider the following program.

|  |
| --- |
| public class Mystery {  public static void main(String[] args) {  int N = Integer.parseInt(args[0]);  int[] a = new int[M];  while(!StdIn.isEmpty()) {  int num = StdIn.readInt();  a[num]++;  }  for (int i = 0; i < M; i++)  for (int j = 0; j < a[i]; j++)  System.out.print(i + " ");  System.out.println();  }  } |

Suppose the file input.txt contains the following integers:

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

What is the contents of the array a after running the following command

|  |
| --- |
| java Mystery 10 < input.txt |

1. **High-low.** Shuffle a deck of cards, and deal one to the player. Prompt the player to guess whether the next card is higher or lower than the current card. Repeat until player guesses it wrong. Game show: ???? used this.
2. **Elastic collisions.** Write a program CollidingBalls.java that takes a command-line argument n and plots the trajectories of n bouncing balls that bounce of the walls and each other according to the laws of elastic collisions. Assume all the balls have the same mass.
3. **Elastic collisions with obstacles.** Each ball should have its own mass. Put a large ball in the center with zero initial velocity. Brownian motion.
4. **Statistical outliers.** Modify [Average.java](https://introcs.cs.princeton.edu/java/15inout/Average.java.html) to print out all the values that are larger than 1.5 standard deviations from the mean. You will need an array to store the values.
5. **Optical illusions.** Create a [Kofka ring](http://web.mit.edu/persci/gaz/gaz-teaching/flash/koffka-movie.swf) or one of the other [optical illusions](http://web.mit.edu/persci/gaz/gaz-teaching/index.html) collected by Edward Adelson.
6. **Computer animation.** In 1995 James Gosling presented a demonstration of Java to Sun executives, illustrating its potential to deliver dynamic and interactive Web content. At the time, web pages were fixed and non-interactive. To demonstrate what the Web could be, Gosling presented applets to rotate 3D molecules, visualize sorting routines, and Duke cart-wheeling across the screen. Java was officially introduced in May 1995 and widely adopted in the technology sector. The Internet would never be the same.

Program [Duke.java](https://introcs.cs.princeton.edu/java/15inout/Duke.java.html) reads in the 17 images [T1.gif](https://introcs.cs.princeton.edu/java/15inout/T1.gif) through [T17.gif](https://introcs.cs.princeton.edu/java/15inout/T17.gif) and produces the animation. To execute on your computer, download the 17 GIF files and put in the same directory as Duke.java. (Alternatively, download and unzip the file [duke.zip](https://introcs.cs.princeton.edu/java/15inout/duke.zip) or [duke.jar](https://introcs.cs.princeton.edu/java/15inout/duke.jar) to extract all 17 GIFs.)

1. **Cart-wheeling Duke.** Modify [Duke.java](https://introcs.cs.princeton.edu/java/15inout/Duke.java.html) so that it cartwheels 5 times across the screen, from right to left, wrapping around when it hits the window boundary. Repeat this cart-wheeling cycle 100 times. *Hint*: after displaying a sequence of 17 frames, move 57 pixels to the left and repeat. Name your program [MoreDuke.java](https://introcs.cs.princeton.edu/java/15inout/MoreDuke.java.html).
2. **Tac (cat backwards).** Write a program Tac.java that reads lines of text from standard input and prints the lines out in reverse order.
3. **Game.** Implement the game [dodge](http://www.lewpen.com/game/) using StdDraw: move a blue disc within the unit square to touch a randomly placed green disc, while avoiding the moving red discs. After each touch, add a new moving red disc.
4. **Simple harmonic motion.** Create an animation like the one below from [Wikipedia](http://en.wikipedia.org/wiki/Image:Simple_harmonic_motion_animation.gif) of simple harmonic motion.
5. **Yin yang.** Draw a [yin yang](http://www.krazydad.com/bestiary/bestiary_yinyang.html) using StdDraw.arc().
6. **Twenty questions.** Write a program [QuestionsTwenty.java](https://introcs.cs.princeton.edu/java/15inout/QuestionsTwenty.java.html) that plays 20 questions from the opposite point of view: the user thinks of a number between 1 and a million and the computer makes the guesses. Use *binary search* to ensure that the computer needs at most 20 guesses.
7. Write a program DeleteX.java that reads in text from standard input and deletes all occurrences of the letter X. To filter a file and remove all X's, run your program with the following command:

|  |
| --- |
| % **java DeleteX < input.txt > output.txt** |

1. Write a program ThreeLargest.java that reads integers from standard input and prints out the three largest inputs.
2. Write a program Pnorm.java that takes a command-line argument p, reads in real numbers from standard input, and prints out their *p-norm*. The p-norm norm of a vector (x1, ..., xN) is defined to be the pth root of (|x1|p + |x2|p + ... + |xN|p).
3. Consider the following Java program.

|  |
| --- |
| public class Mystery {  public static void main(String[] args) {  int i = StdIn.readInt();  int j = StdIn.readInt();  System.out.println((i-1));  System.out.println((j\*i));  }  } |

Suppose that the file input.txt contains

|  |
| --- |
| 5 1 |

What does the following command do?

|  |
| --- |
| java Mystery < input.txt |

1. Repeat the previous exercise but use the following command instead

|  |
| --- |
| java Mystery < input.txt | java Mystery | java Mystery | java Mystery |

1. Consider the following Java program.

|  |
| --- |
| public class Mystery {  public static void main(String[] args) {  int i = StdIn.readInt();  int j = StdIn.readInt();  int k = i + j;  System.out.println(j);  System.out.println(k);  }  } |

Suppose that the file input.txt contains the integers 1 and 1. What does the following command do?

|  |
| --- |
| java Mystery < input.txt | java Mystery | java Mystery | java Mystery |

1. Consider the Java program [Ruler.java](https://introcs.cs.princeton.edu/java/15inout/Ruler.java.html).

|  |
| --- |
| public class Ruler {  public static void main(String[] args) {  int n = StdIn.readInt();  String s = StdIn.readString();  System.out.println((n+1) + " " + s + (n+1) + s);  }  } |

Suppose that the file input.txt contains the integers 1 and 1. What does the following command do?

|  |
| --- |
| java Ruler < input.txt | java Ruler | java Ruler | java Ruler |

1. Modify [Add.java](https://introcs.cs.princeton.edu/java/15inout/Add.java.html) so that it re-asks the user to enter two positive integers if the user types in a non-positive integer.
2. Modify [TwentyQuestions.java](https://introcs.cs.princeton.edu/java/15inout/TwentyQuestions.java.html) so that it re-asks the user to enter a response if the user types in something other than true or false. Hint: add a do-while loop within the main loop.
3. **Nonagram.** Write a program to plot a [nonagram](http://mathworld.wolfram.com/Nonagram.html).
4. **Star polygons.** Write a program StarPolygon.java that takes two command line inputs p and q, and plots the [{p/q}-star polygon](http://mathworld.wolfram.com/StarPolygon.html).
5. **Complete graph.** Write a program to plot that takes an integer N, plots an N-gon, where each vertex lies on a circle of radius 256. Then draw a gray line connecting each pair of vertices.
6. **Necker cube.** Write a program NeckerCube.java to plot a [Necker cube](http://mathworld.wolfram.com/NeckerCube.html).
7. What happens if you move the StdDraw.clear(Color.BLACK) command to before the beginning of the while loop in [BouncingBall.java](https://introcs.cs.princeton.edu/java/15inout/BouncingBall.java.html)? *Answer*: try it and observe a nice woven 3d pattern with the given starting velocity and position.
8. What happens if you change the parameter of StdDraw.show() to 0 or 1000 in [BouncingBall.java](https://introcs.cs.princeton.edu/java/15inout/BouncingBall.java.html)?
9. Write a program to plot a circular ring of width 10 like the one below using two calls to StdDraw.filledCircle().
10. Write a program to plot a circular ring of width 10 like the one below using a nested for loop and many calls to StdDraw.point().
11. Write a program to plot the Olympic rings.
12. Write a program [BouncingBallDeluxe.java](https://introcs.cs.princeton.edu/java/15inout/BouncingBallDeluxe.java.html) that embellishes [BouncingBall.java](https://introcs.cs.princeton.edu/java/15inout/BouncingBall.java.html) by playing a sound effect upon collision with the wall using StdAudio and the sound file [pipebang.wav](https://introcs.cs.princeton.edu/java/15inout/pipepang.wav).

*Last modified on February 25, 2017.*  
  
Copyright © 2000–2018 [Robert Sedgewick](https://www.cs.princeton.edu/%7Ers) and [Kevin Wayne](https://www.cs.princeton.edu/%7Ewayne). All rights reserved.

In the programs that we have examined to this point, each of the statements is executed once, in the order given. Most programs are more complicated because the sequence of statements and the number of times each is executed can vary. We use the term *control flow* to refer to statement sequencing in a program.

**If statements.**

Most computations require different actions for different inputs.

* The following code fragment uses an if statement to put the smaller of two int values in x and the larger of the two values in y, by exchanging the values in the two variables if necessary.
* [Flip.java](https://introcs.cs.princeton.edu/java/13flow/Flip.java.html) uses Math.random() and an if-else statement to print the results of a coin flip.
* The table below summarizes some typical situations where you might need to use an if or if-else statement.

**While loops.**

Many computations are inherently repetitive. The while loop enables us to execute a group of statements many times. This enables us to express lengthy computations without writing lots of code.

* The following code fragment computes the largest power of 2 that is less than or equal to a given positive integer *n*.

* [TenHellos.java](https://introcs.cs.princeton.edu/java/13flow/TenHellos.java.html) prints "Hello World" 10 times.
* [PowersOfTwo.java](https://introcs.cs.princeton.edu/java/13flow/PowersOfTwo.java.html) takes an integer command-line argument *n* and prints all of the powers of 2 less than or equal to *n*.

**For loops.**

The *for loop* is an alternate Java construct that allows us even more flexibility when writing loops.

* *For notation.* Many loops follow the same basic scheme: initialize an index variable to some value and then use a while loop to test an exit condition involving the index variable, using the last statement in the while loop to modify the index variable. Java's for loop is a direct way to express such loops.
* *Compound assignment idioms.* The idiom i++ is a shorthand notation for i = i + 1.
* *Scope.* The *scope* of a variable is the part of the program that can refer to that variable by name. Generally the scope of a variable comprises the statements that follow the declaration in the same block as the declaration. For this purpose, the code in the for loop header is considered to be in the same block as the for loop body.

**Nesting.**

The if, while, and for statements have the same status as assignment statements or any other statements in Java; that is, we can use them wherever a statement is called for. In particular, we can use one or more of them in the body of another statement to make *compound statements*. To emphasize the nesting, we use indentation in the program code.

* [DivisorPattern.java](https://introcs.cs.princeton.edu/java/13flow/DivisorPattern.java.html) has a for loop whose body contains a for loop (whose body is an if-else statement) and a print statement. It prints a pattern of asterisks where the *i*th row has an asterisk in each position corresponding to divisors of *i* (the same holds true for the columns).
* [MarginalTaxRate.java](https://introcs.cs.princeton.edu/java/13flow/MarginalTaxRate.java.html) computes the marginal tax rate for a given income. It uses several nested if-else statements to test from among a number of mutually exclusive possibilities.

**Loop examples.**

**Applications.**

The ability to program with loops and conditionals immediately opens up the world of computation to us.

* *Ruler subdivisions.* [RulerN.java](https://introcs.cs.princeton.edu/java/13flow/RulerN.java.html) takes an integer command-line argument n and prints the string of ruler subdivision lengths. This program illustrates one of the essential characteristics of loops—the program could hardly be simpler, but it can produce a huge amount of output.
* *Finite sums.* The computational paradigm used in [PowersOfTwo.java](https://introcs.cs.princeton.edu/java/13flow/PowersOfTwo.java.html) is one that you will use frequently. It uses two variables—one as an index that controls a loop, and the other to accumulate a computational result. Program [HarmonicNumber.java](https://introcs.cs.princeton.edu/java/13flow/HarmonicNumber.java.html) uses the same paradigm to evaluate the sum

*Hn*=11+12+13+14+…+1*n*

* These numbers, which are known as the *harmonic numbers*, arise frequently in the analysis of algorithms.
* *Newton's method.* [Sqrt.java](https://introcs.cs.princeton.edu/java/13flow/Sqrt.java.html) uses a classic iterative technique known as *Newton's* method to compute the square root of a positive number *x*: Start with an estimate *t*. If *t* is equal to *x/t* (up to machine precision), then *t* is equal to a square root of *x*, so the computation is complete. If not, refine the estimate by replacing *t* with the average of *t* and *x/t*. Each time we perform this update, we get closer to the desired answer.
* *Number conversion.* [Binary.java](https://introcs.cs.princeton.edu/java/13flow/Binary.java.html) prints the binary (base 2) representation of the decimal number typed as the command-line argument. It is based on decomposing the number into a sum of powers of 2. For example, the binary representation of 106 is 11010102, which is the same as saying that 106 = 64 + 32 + 8 + 2. To compute the binary representation of *n*, we consider the powers of 2 less than or equal to *n* in decreasing order to determine which belong in the binary decomposition (and therefore correspond to a 1 bit in the binary representation).
* *Gambler's ruin.* Suppose a gambler makes a series of fair $1 bets, starting with $50, and continue to play until she either goes broke or has $250. What are the chances that she will go home with $250, and how many bets might she expect to make before winning or losing? [Gambler.java](https://introcs.cs.princeton.edu/java/13flow/Gambler.java.html) is a simulation that can help answer these questions. It takes three command-line arguments, the initial stake ($50), the goal amount ($250), and the number of times we want to simulate the game.
* *Prime factorization.* [Factors.java](https://introcs.cs.princeton.edu/java/13flow/Factors.java.html) takes an integer command-line argument n and prints its prime factorization. In contrast to many of the other programs that we have seen (which we could do in a few minutes with a calculator or pencil and paper), this computation would not be feasible without a computer.

**Other conditional and loop constructs.**

To be complete, we consider four more Java constructs related to conditionals and loops. They are used much less frequently than the if, while, and for statements that we've been working with, but it is worthwhile to be aware of them.

* *Break statements.* In some situations, we want to immediate exit a loop without letting it run to completion. Java provides the break statement for this purpose. [Prime.java](https://introcs.cs.princeton.edu/java/13flow/Prime.java.html) takes an integer command-line argument n and prints true if n is prime, and false otherwise. There are two different ways to leave this loop: either the break statement is executed (because n is not prime) or the loop-continuation condition is not satisfied (because n is prime).

Note that the break statement does not apply to if or if-else statements. In a [famous programming bug](http://users.csc.calpoly.edu/%7Ejdalbey/SWE/Papers/att_collapse.html), the U.S. telephone network crashed because a programmer intended to use a break statement to exit a complicated if statement.

* *Continue statements.* Java also provides a way to skip to the next iteration of a loop: the continue statement. When a continue is executed within the body of a for loopy, the flow of control transfers directly to the increment statement for the next iteration of the loop.
* *Switch statements.* The if and if-else statements allow one or two alternatives. Sometimes, a computation naturally suggests more than two mutually exclusive alternatives. Java provides the switch statement for this purpose. [NameOfDay.java](https://introcs.cs.princeton.edu/java/13flow/NameOfDay.java.html) takes an integer between 0 and 6 as a command-line argument and uses a switch statement to print the corresponding name of the day (Sunday to Saturday).
* *Do–while loops.* A do-while loop is almost the same as a while loop except that the loop-continuation condition is omitted the first time through the loop. [RandomPointInCircle.java](https://introcs.cs.princeton.edu/java/13flow/RandomPointInCircle.java.html) sets x and y so that (x, y) is randomly distributed inside the circle centered at (0, 0) with radius 1.

With Math.random() we get points that are randomly distributed in the 2-by-2 square center at (0, 0). We just generate points in this region until we find one that lies inside the unit disk. We always want to generate at least one point so a do-while loop is most appropriate. We must declare x and y outside the loop since we will want to access their values after the loop terminates.

We don't use the following two flow control statements in this textbook, but include them here for completeness.

* *Conditional operator.* The conditional operator ?: is a ternary operator (three operands) that enables you to embed a conditional within an expression. The three operands are separated by the ? and : symbols. If the first operand (a boolean expression) is true, the result has the value of the second expression; otherwise it has the value of the third expression.

|  |
| --- |
| int min = (x < y) ? x : y; |

* *Labeled break and continue statements.* The break and continue statements apply to the innermost for or while loop. Sometimes we want to jump out of several levels of nested loops. Java provides the labeled break and labeled continue statements to accomplish this. Here is an [example](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html).

**Exercises**

1. Write a program [AllEqual.java](https://introcs.cs.princeton.edu/java/13flow/AllEqual.java.html) that takes three integer command-line arguments and prints equal if all three are equal, and not equal otherwise.
2. Write a program [RollLoadedDie.java](https://introcs.cs.princeton.edu/java/13flow/RollLoadedDie.java.html) that prints the result of rolling a loaded die such that the probability of getting a 1, 2, 3, 4, or 5 is 1/8 and the probability of getting a 6 is 3/8.
3. Rewrite [TenHellos.java](https://introcs.cs.princeton.edu/java/13flow/TenHellos.java.html) to make a program [Hellos.java](https://introcs.cs.princeton.edu/java/13flow/Hellos.java.html) that takes the number of lines to print as a command-line argument. You may assume that the argument is less than 1000. Hint: consider using i % 10 and i % 100 to determine whether to use "st", "nd", "rd", or "th" for printing the ith Hello.
4. Write a program [FivePerLine.java](https://introcs.cs.princeton.edu/java/13flow/FivePerLine.java.html) that, using one for loop and one if statement, prints the integers from 1000 to 2000 with five integers per line. *Hint*: use the % operator.
5. Write a program [FunctionGrowth.java](https://introcs.cs.princeton.edu/java/13flow/FunctionGrowth.java.html) that prints a table of the values of *ln n*, *n*, *n ln n*, *n2*, *n3*, and *2n* for *n* = 16, 32, 64, ..., 2048. Use tabs ('\t' characters) to line up columns.
6. What is the value of m and n after executing the [following code](https://introcs.cs.princeton.edu/java/13flow/DigitReverser.java.html)?

|  |
| --- |
| int n = 123456789;  int m = 0;  while (n != 0) {  m = (10 \* m) + (n % 10);  n = n / 10;  } |

1. What does the [following code](https://introcs.cs.princeton.edu/java/13flow/Fibonacci.java.html) print out?

|  |
| --- |
| int f = 0, g = 1;  for (int i = 0; i <= 15; i++) {  System.out.println(f);  f = f + g;  g = f - g;  } |

1. Unlike the harmonic numbers, the sum 1/1 + 1/4 + 1/9 + 1/16 + ... + 1/n2 *does* converge to a constant as *n* grows to infinity. (Indeed, the constant is π2 / 6, so this formula can be used to estimate the value of π.) Which of the following [for loops](https://introcs.cs.princeton.edu/java/13flow/SeriesSum.java.html) computes this sum? Assume that n is an int initialized to 1000000 and sum is a double initialized to 0.

|  |
| --- |
| (a) for (int i = 1; i <= n; i++)  sum = sum + 1 / (i \* i);  (b) for (int i = 1; i <= n; i++)  sum = sum + 1.0 / i \* i;  (c) for (int i = 1; i <= n; i++)  sum = sum + 1.0 / (i \* i);  (d) for (int i = 1; i <= n; i++)  sum = sum + 1 / (1.0 \* i \* i); |

1. Modify [Binary.java](https://introcs.cs.princeton.edu/java/13flow/Binary.java.html) to get a program Modify Kary.java that takes a second command-line argument K and converts the first argument to base K. Assume the base is between 2 and 16. For bases greater than 10, use the letters A through F to represent the 11th through 16th digits, respectively.
2. Write a program [code fragment](https://introcs.cs.princeton.edu/java/13flow/IntegerToBinaryString.java.html) that puts the binary representation of a positive integer n into a String variable s.

**Creative Exercises**

1. **Ramanujan's taxi.** S. Ramanujan was an Indian mathematician who became famous for his intuition for numbers. When the English mathematician G. H. Hardy came to visit him in the hospital one day, Hardy remarked that the number of his taxi was 1729, a rather dull number. To which Ramanujan replied, "No, Hardy! No, Hardy! It is a very interesting number. It is the smallest number expressible as the sum of two cubes in two different ways." Verify this claim by writing a program [Ramanujan.java](https://introcs.cs.princeton.edu/java/13flow/Ramanujan.java.html) that takes an integer command-line argument n and prints all integers less than or equal to n that can be expressed as the sum of two cubes in two different ways - find distinct positive integers *a*, *b*, *c*, and *d* such that *a3 + b3 = c3 + d3*. Use four nested for loops.

Now, the license plate 87539319 seems like a rather dull number. Determine why it's not.

1. **Checksums.** The International Standard Book Number ([ISBN](http://mathworld.wolfram.com/ISBN.html)) is a 10 digit code that uniquely specifies a book. The rightmost digit is a *checksum* digit which can be uniquely determined from the other 9 digits from the condition that *d1 + 2d2 + 3d3 + ... + 10d10* must be a multiple of 11 (here *di* denotes the ith digit from the right). The checksum digit *d1* can be any value from 0 to 10: the ISBN convention is to use the value X to denote 10. *Example*: the checksum digit corresponding to 020131452 is 5 since is the only value of *d1* between 0 and and 10 for which *d1 + 2\*2 + 3\*5 + 4\*4 + 5\*1 + 6\*3 + 7\*1 + 8\*0 + 9\*2 + 10\*0* is a multiple of 11. Write a program [ISBN.java](https://introcs.cs.princeton.edu/java/13flow/ISBN.java.html) that takes a 9-digit integer as a command-line argument, computes the checksum, and prints the 10-digit ISBN number. It's ok if you don't print any leading 0s.
2. **Exponential function.** Assume that x is a positive variable of type double. Write a program [Exp.java](https://introcs.cs.princeton.edu/java/13flow/Exp.java.html) that computes e^x using the Taylor series expansion

*ex*=1+*x*+*x*22!+*x*33+*x*44!+…

  **Trigonometric functions.** Write two programs [Sin.java](https://introcs.cs.princeton.edu/java/13flow/Sin.java.html) and Cos.java that compute *sin x* and *cos x* using the Taylor series expansions

sin*x*=*x*−*x*33!+*x*55!−*x*77!+…

cos*x*=1−*x*22!+*x*44!−*x*66!+…

2. **Game simulation.** In the game show *Let's Make a Deal*, a contestant is presented with three doors. Behind one door is a valuable prize, behind the other two are gag gifts. After the contestant chooses a door, the host opens up one of the other two doors (never revealing the prize, of course). The contestant is then given the opportunity to switch to the other unopened door. Should the contestant do so? Intuitively, it might seem that the contestant's initial choice door and the other unopened door are equally likely to contain the prize, so there would be no incentive to switch. Write a program [MonteHall.java](https://introcs.cs.princeton.edu/java/13flow/MonteHall.java.html) to test this intuition by simulation. Your program should take an integer command-line argument *n*, play the game *n* times using each of the two strategies (switch or don't switch) and print the chance of success for each strategy. Or you can [play the game here](http://www.stat.sc.edu/%7Ewest/javahtml/LetsMakeaDeal.html).
3. **Euler's sum-of-powers conjecture.** In 1769 Leonhard Euler formulated a generalized version of Fermat's Last Theorem, conjecturing that at least *n* *n*th powers are needed to obtain a sum that is itself an *n*th power, for *n* > 2. Write a program [Euler.java](https://introcs.cs.princeton.edu/java/13flow/Euler.java.html) to disprove Euler's conjecture (which stood until 1967), using a quintuply nested loop to find four positive integers whose 5th power sums to the 5th power of another positive integer. That is, find *a*, *b*, *c*, *d*, and *e* such that *a*5 + *b*5 + *c*5 + *d*5 = *e*5. Use the long data type.

**Web Exercises**

1. Write a program [RollDie.java](https://introcs.cs.princeton.edu/java/13flow/RollDie.java.html) that generates the result of rolling a fair six-sided die (an integer between 1 and 6).
2. Write a program that takes three integer command-line arguments a, b, and c and print the number of distinct values (1, 2, or 3) among a, b, and c.
3. Write a program that takes five integer command-line arguments and prints the *median* (the third largest one).
4. (hard) Now, try to compute the median of 5 elements such that when executed, it never makes more than 6 total comparisons.
5. How can I create in an infinite loop with a for loop?

*Solution*: for(;;) is the same as while(true).

1. What's wrong with the following loop?

|  |
| --- |
| boolean done = false;  while (done = false) {  ...  } |

The while loop condition uses = instead of == so it is an assignment statement (which makes done always false and the body of the loop will never be executed). It's better to style to avoid using ==.

|  |
| --- |
| boolean done = false;  while (!done) {  ...  } |

1. What's wrong with the following loop that is intended to compute the sum of the integers 1 through 100?

|  |
| --- |
| for (int i = 1; i <= N; i++) {  int sum = 0;  sum = sum + i;  }  System.out.println(sum); |

The variable sum should be defined outside the loop. By defining it inside the loop, a new variable sum is initialized to 0 each time through the loop; also it is not even accessible outside the loop.

1. Write a program [Hurricane.java](https://introcs.cs.princeton.edu/java/13flow/Hurricane.java.html) that that takes the wind speed (in miles per hour) as an integer command-line argument and prints whether it qualifies as a hurricane, and if so, whether it is a Category 1, 2, 3, 4, or 5 hurricane. Below is a table of the wind speeds according to the [Saffir-Simpson scale](http://www.marinewaypoints.com/marine/wind.shtml).

|  |  |
| --- | --- |
| **Category** | **Wind Speed (mph)** |
| 1 | 74 - 95 |
| 2 | 96 - 110 |
| 3 | 111 - 130 |
| 4 | 131 - 155 |
| 5 | 155 and above |

1. What is wrong with the following code fragment?

|  |
| --- |
| double x = -32.2;  boolean isPositive = (x > 0);  if (isPositive = true) System.out.println(x + " is positive");  else System.out.println(x + " is not positive"); |

*Solution*: It uses the assignment operator = instead of the equality operator ==. A better solution is to write if (isPositive).

1. Change/add one character so that the following program prints 20 xs. There are two different solutions.

|  |
| --- |
| int i = 0, n = 20;  for (i = 0; i < n; i--)  System.out.print("x"); |

*Solution*: Replace the i < n condition with -i < n. Replace the i-- with n--. ( In C, there is a third: replace the < with a +.)

1. What does the following code fragment do?

|  |
| --- |
| if (x > 0);  System.out.println("positive"); |

*Solution*: always prints positive regardless of the value of x because of the extra semicolon after the if statement.

1. **RGB to HSB converter.** Write a program RGBtoHSV.java that takes an RGB color (three integers between 0 and 255) and transforms it to an [HSB color](http://en.wikipedia.org/wiki/HSV_color_space) (three different integers between 0 and 255). Write a program HSVtoRGB.java that applies the inverse transformation.
2. **Boys and girls.** A couple beginning a family decides to keep having children until they have at least one of either sex. Estimate the average number of children they will have via simulation. Also estimate the most common outcome (record the frequency counts for 2, 3, and 4 children, and also for 5 and above). Assume that the probability p of having a boy or girl is 1/2.
3. What does the following program do?

|  |
| --- |
| public static void main(String[] args) {  int N = Integer.parseInt(args[0]);  int x = 1;  while (N >= 1) {  System.out.println(x);  x = 2 \* x;  N = N / 2;  }  } |

*Solution*: Prints all of the powers of 2 less than or equal to n.

1. **Boys and girls.** Repeat the previous question, but assume the couple keeps having children until they have another child which is of the same sex as the first child. How does your answer change if p is different from 1/2?

*Surprisingly, the average number of children is 2 if p = 0 or 1, and 3 for all other values of p. But the most likely value is 2 for all values of p.*

1. Given two positive integers a and b, what result does the following code fragment leave in c

|  |
| --- |
| c = 0;  while (b > 0) {  if (b % 2 == 1) c = c + a;  b = b / 2;  a = a + a;  } |

*Solution*: a \* b.

1. Write a program using a loop and four conditionals to print

|  |
| --- |
| 12 midnight  1am  2am  ...  12 noon  1pm  ...  11pm |

1. What does the following program print?

|  |
| --- |
| public class Test {  public static void main(String[] args) {  if (10 > 5);  else; {  System.out.println("Here");  };  }  } |

1. Alice tosses a fair coin until she sees two consecutive heads. Bob tosses another fair coin until he sees a head followed by a tail. Write a program to estimate the probability that Alice will make fewer tosses than Bob? *Solution*: [39/121](http://math.smsu.edu/%7Eles/Adv51.html).
2. Rewrite [DayOfWeek.java](https://introcs.cs.princeton.edu/java/12types/DayOfWeek.java.html) from Exercise 1.2.29 so that it prints the day of the week as Sunday, Monday, and so forth instead of an integer between 0 and 6. Use a switch statement.
3. **Number-to-English.** Write a program to read in a command line integer between -999,999,999 and 999,999,999 and print the English equivalent. Here is an exhaustive list of words that your program should use: negative, zero, one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen, eighteen, nineteen, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, hundred, thousand, million . Don't use hundred, when you can use thousand, e.g., use one thousand five hundred instead of fifteen hundred. [Reference](http://acm.pku.edu.cn/JudgeOnline/showproblem?problem_id=2121).
4. **Gymnastics judging.** A gymnast's score is determined by a panel of 6 judges who each decide a score between 0.0 and 10.0. The final score is determined by discarding the high and low scores, and averaging the remaining 4. Write a program GymnasticsScorer.java that takes 6 real command line inputs representing the 6 scores and prints their average, after throwing out the high and low scores.
5. **Quarterback rating.** To compare NFL quarterbacks, the NFL devised a the [quarterback rating](http://www.mathnotes.com/aw_quarterback.html) formula based on the quarterbacks number of completed passes (A), pass attempts (B), passing yards (C), touchdown passes (D), and interception (E) as follows:
   1. Completion ratio: W = 250/3 \* ((A / B) - 0.3).
   2. Yards per pass: X = 25/6 \* ((C / B) - 3).
   3. Touchdown ratio: Y = 1000/3 \* (D / B)
   4. Interception ratio: Z = 1250/3 \* (0.095 - (E / B))

The *quarterback rating* is computed by summing up the above four quantities, but rounding up or down each value so that it is at least 0 and and at most 475/12. Write a program QuarterbackRating.java that takes five command line inputs A, B, C, D, and E, and prints the quarterback rating. Use your program to compute Steve Young's 1994 record-setting season (112.8) in which he completed 324 of 461 passes for 3,969 yards, and threw 35 touchdowns and 10 interceptions. As of 2014, the all-time single-season record is 122.5 by Aaron Rodgers in 2011.

1. **Decimal expansion of rational numbers.** Given two integers p and q, the decimal expansion of p/q has an infinitely repeating cycle. For example, 1/33 = 0.03030303.... We use the notation 0.(03) to denote that 03 repeats indefinitely. As another example, 8639/70000 = 0.1234(142857). Write a program DecimalExpansion.java that reads in two command line integers p and q and prints the decimal expansion of p/q using the above notation. *Hint*: use Floyd's rule.
2. **Friday the 13th.** What is the maximum number of consecutive days in which no Friday the 13th occurs? *Hint*: The Gregorian calendar repeats itself every 400 years (146097 days) so you only need to worry about a 400 year interval.

*Solution*: 426 (e.g., from 8/13/1999 to 10/13/2000).

1. **January 1.** Is January 1 more likely to fall on a Saturday or Sunday? Write a program to determine the number of times each occurs in a 400 year interval.

*Solution:* Sunday (58 times) is more likely than Saturday (56 times).

1. What do the following two code fragments do?

|  |
| --- |
| for (int i = 0; i < N; i++)  for (int j = 0; j < N; j++)  if (i != j) System.out.println(i + ", " + j);  for (int i = 0; i < N; i++)  for (int j = 0; (i != j) && (j < N); j++)  System.out.println(i + ", " + j); |

1. Determine what value gets printed out without using a computer. Choose the correct answer from 0, 100, 101, 517, or 1000.

|  |
| --- |
| int cnt = 0;  for (int i = 0; i < 10; i++)  for (int j = 0; j < 10; j++)  for (int k = 0; k < 10; k++)  if (2\*i + j >= 3\*k)  cnt++;  System.out.println(cnt); |

1. Rewrite [CarLoan.java](https://introcs.cs.princeton.edu/java/22types/CarLoan.java.html) from Creative Exercise XYZ so that it properly handles an interest rate of 0% and avoids dividing by 0.
2. Write the shortest Java program you can that takes an integer command-line argument n and prints true if (1 + 2 + ... + n) 2 is equal to (13 + 23 + ... + n3).

*Solution*: Always print true.

1. Modify [Sqrt.java](https://introcs.cs.princeton.edu/java/13flow/Sqrt.java.html) so that it reports an error if the user enters a negative number and works properly if the user enters zero.
2. What happens if we initialize t to -x instead of x in program [Sqrt.java](https://introcs.cs.princeton.edu/java/13flow/Sqrt.java.html)?
3. **Sample standard deviation of uniform distribution.** Modify Exercise 8 so that it prints the sample standard deviation in addition to the average.
4. **Sample standard deviation of normal distribution.** that takes an integer N as a command-line argument and uses Web Exercise 1 from [Section 1.2](https://introcs.cs.princeton.edu/java/12types) to print N standard normal random variables, and their average value, and sample standard deviation.
5. **Loaded dice.** [Stephen Rudich] Suppose you have three, three sided dice. A: {2, 6, 7}, B: { 1, 5, 9}, and C: {3, 4, 8}. Two players roll a die and the one with the highest value wins. Which die would you choose? *Solution*: A beats B with probability 5/9, B beats C with probability 5/9 and C beats A with probability 5/9. Be sure to choose second!
6. **Thue–Morse sequence.** Write a program [ThueMorse.java](https://introcs.cs.princeton.edu/java/13flow/ThueMorse.java.html) that reads in a command line integer n and prints the [Thue–Morse sequence](http://mathworld.wolfram.com/Thue-MorseSequence.html) of order n. The first few strings are 0, 01, 0110, 01101001. Each successive string is obtained by flipping all of the bits of the previous string and concatenating the result to the end of the previous string. The sequence has many amazing properties. For example, it is a binary sequence that is *cube-free*: it does not contain 000, 111, 010101, or sss where s is any string. It is *self-similar*: if you delete every other bit, you get another Thue–Morse sequence. It arises in diverse areas of mathematics as well as chess, graphic design, [weaving patterns](http://www.cs.arizona.edu/patterns/weaving/webdocs/gre_mt.pdf), and music composition.
7. Program [Binary.java](https://introcs.cs.princeton.edu/java/13flow/Binary.java.html) prints the binary representation of a decimal number n by casting out powers of 2. Write an alternate version Program [Binary2.java](https://introcs.cs.princeton.edu/java/13flow/Binary2.java.html) that is based on the following method: Write 1 if n is odd, 0 if n is even. Divide n by 2, throwing away the remainder. Repeat until n = 0 and read the answer backwards. Use % to determine whether n is even, and use string concatenation to form the answer in reverse order.
8. What does the following code fragment do?

|  |
| --- |
| int digits = 0;  do {  digits++;  n = n / 10;  } while (n > 0); |

*Solution*: The number of bits in the binary representation of a natural number n. We use a do-while loop so that code output 1 if n = 0.

1. Write a program NPerLine.java that takes an integer command-line argument n and prints the integers from 10 to 99 with n integers per line.
2. Modify NPerLine.java so that it prints the integers from 1 to 1000 with n integers per line. Make the integers line up by printing the right number of spaces before an integer (e.g., three for 1-9, two for 10-99, and one for 100-999).
3. Suppose a, b, and c are random number uniformly distributed between 0 and 1. What is the probability that a, b, and c form the side length of some triangle? *Hint*: they will form a triangle if and only if the sum of every two values is larger than the third.
4. Repeat the previous question, but calculate the probability that the resulting triangle is obtuse, given that the three numbers for a triangle. *Hint*: the three lengths will form an obtuse triangle if and only if (i) the sum of every two values is larger than the third and (ii) the sum of the squares of every two side lengths is greater than or equal to the square of the third.

[Answer](http://math.smsu.edu/%7Eles/Adv28.html).

1. What is the value of s after executing the [following code](https://introcs.cs.princeton.edu/java/13flow/DigitReverser.java.html)?

|  |
| --- |
| int M = 987654321;  String s = "";  while (M != 0) {  int digit = M % 10;  s = s + digit;  M = M / 10;  } |

1. What is the value of i after the following [confusing code](https://introcs.cs.princeton.edu/java/13flow/Confusion.java.html) is executed?

|  |
| --- |
| int i = 10;  i = i++;  i = ++i;  i = i++ + ++i; |

Moral: don't write code like this.

1. **Formatted ISBN number.** Write a program [ISBN2.java](https://introcs.cs.princeton.edu/java/13flow/ISBN2.java.html) that reads in a 9 digit integer from a command-line argument, computes the check digit, and prints the fully formatted ISBN number, e.g, 0-201-31452-5.
2. **UPC codes.** The Universal Product Code ([UPC](http://mathworld.wolfram.com/UPC.html)) is a 12 digit code that uniquely specifies a product. The least significant digit d1(rightmost one) is a check digit which is the uniquely determined by making the following expression a multiple of 10:

(d1 + d3 + d5 + d7 + d9 + d11) + 3 (d2 + d4 + d6 + d8 + d10 + d12)

As an example, the check digit corresponding to 0-48500-00102 (Tropicana Pure Premium Orange Juice) is 8 since

(8 + 0 + 0 + 0 + 5 + 4) + 3 (2 + 1 + 0 + 0 + 8 + 0) = 50

and 50 is a multiple of 10. Write a program that reads in a 11 digit integer from a command line parameter, computes the check digit, and prints the the full UPC. *Hint*: use a variable of type long to store the 11 digit number.

1. Write a program that reads in the wind speed (in knots) as a command line argument and prints its force according to the [Beaufort scale](http://boating.ncf.ca/beaufort.html). Use a switch statement.
2. **Making change.** Write a program that reads in a command line integer N (number of pennies) and prints the best way (fewest number of coins) to make change using US coins (quarters, dimes, nickels, and pennies only). For example, if N = 73 then print

|  |
| --- |
| 2 quarters  2 dimes  3 pennies |

*Hint*: use the greedy algorithm. That is, dispense as many quarters as possible, then dimes, then nickels, and finally pennies.

1. Write a program [Triangle.java](https://introcs.cs.princeton.edu/java/13flow/Triangle.java.html) that takes a command-line argument N and prints an N-by-N triangular pattern like the one below.

|  |
| --- |
| \* \* \* \* \* \*  . \* \* \* \* \*  . . \* \* \* \*  . . . \* \* \*  . . . . \* \*  . . . . . \* |

1. Write a program [Ex.java](https://introcs.cs.princeton.edu/java/13flow/Ex.java.html) that takes a command-line argument N and prints a (2N + 1)-by-(2N + 1) ex like the one below. Use two for loops and one if-else statement.

|  |
| --- |
| \* . . . . . \*  . \* . . . \* .  . . \* . \* . .  . . . \* . . .  . . \* . \* . .  . \* . . . \* .  \* . . . . . \* |

1. Write a program [BowTie.java](https://introcs.cs.princeton.edu/java/13flow/BowTie.java.html) that takes a command-line argument N and prints a (2N + 1)-by-(2N + 1) bowtie like the one below. Use two for loops and one if-else statement.

|  |
| --- |
| \* . . . . . \*  \* \* . . . \* \*  \* \* \* . \* \* \*  \* \* \* \* \* \* \*  \* \* \* . \* \* \*  \* \* . . . \* \*  \* . . . . . \* |

1. Write a program [Diamond.java](https://introcs.cs.princeton.edu/java/13flow/Diamond.java.html) that takes a command-line argument N and prints a (2N + 1)-by-(2N + 1) diamond like the one below.

|  |
| --- |
| % java Diamond 4  . . . . \* . . . .  . . . \* \* \* . . .  . . \* \* \* \* \* . .  . \* \* \* \* \* \* \* .  \* \* \* \* \* \* \* \* \*  . \* \* \* \* \* \* \* .  . . \* \* \* \* \* . .  . . . \* \* \* . . .  . . . . \* . . . . |

1. Write a program [Heart.java](https://introcs.cs.princeton.edu/java/13flow/Heart.java.html) that takes a command-line argument N and prints a heart.
2. What does the program [Circle.java](https://introcs.cs.princeton.edu/java/13flow/Circle.java.html) print out when N = 5?

|  |
| --- |
| for (int i = -N; i <= N; i++) {  for (int j = -N; j <= N; j++) {  if (i\*i + j\*j <= N\*N) System.out.print("\* ");  else System.out.print(". ");  }  System.out.println();  } |

1. **Seasons.** Write a program Season.java that takes two command line integers M and D and prints the season corresponding to month M (1 = January, 12 = December) and day D in the northern hemisphere. Use the following table

|  |  |  |
| --- | --- | --- |
| **SEASON** | **FROM** | **TO** |
| Spring | March 21 | June 20 |
| Summer | June 21 | September 22 |
| Fall | September 23 | December 21 |
| Winter | December 21 | March 20 |

1. **Zodiac signs.** Write a program Zodiac.java that takes two command line integers M and D and prints the Zodiac sign corresponding to month M (1 = January, 12 = December) and day D. Use the following table

|  |  |  |
| --- | --- | --- |
| **SIGN** | **FROM** | **TO** |
| Capricorn | December 22 | January 19 |
| Aquarius | January 20 | February 17 |
| Pisces | February 18 | March 19 |
| Aries | March 20 | April 19 |
| Taurus | April 20 | May 20 |
| Gemini | May 21 | June 20 |
| Cancer | June 21 | July 22 |
| Leo | July 23 | August 22 |
| Virgo | August 23 | September 22 |
| Libra | September 23 | October 22 |
| Scorpio | October 23 | November 21 |
| Sagittarius | November 22 | December 21 |

1. **Muay Thai kickboxing.** Write a program that reads in the weight of a Muay Thai kickboxer (in pounds) as a command-line argument and prints their weight class. Use a switch statement.

|  |  |  |
| --- | --- | --- |
| **CLASS** | **FROM** | **TO** |
| Flyweight | 0 | 112 |
| Super flyweight | 112 | 115 |
| Bantamweight | 115 | 118 |
| Super bantamweight | 118 | 122 |
| Featherweight | 122 | 126 |
| Super featherweight | 126 | 130 |
| Lightweight | 130 | 135 |
| Super lightweight | 135 | 140 |
| Welterweight | 140 | 147 |
| Super welterweight | 147 | 154 |
| Middleweight | 154 | 160 |
| Super middleweight | 160 | 167 |
| Light heavyweight | 167 | 175 |
| Super light heavyweight | 175 | 183 |
| Cruiserweight | 183 | 190 |
| Heavyweight | 190 | 220 |
| Super heavyweight | 220 | - |

1. **Euler's sum of powers conjecture.** In 1769 Euler generalized Fermat's Last Theorem and conjectured that it is impossible to find three 4th powers whose sum is a 4th power, or four 5th powers whose sum is a 5th power, etc. The conjecture was disproved in 1966 by exhaustive computer search. Disprove the conjecture by finding positive integers a, b, c, d, and e such that a5 + b5 + c5 + d5= e5. Write a program [Euler.java](https://introcs.cs.princeton.edu/java/13flow/Euler.java.html) that reads in a command line parameter N and exhaustively searches for all such solutions with a, b, c, d, and e less than or equal to N. No counterexamples are known for powers greater than 5, but you can join [EulerNet](http://euler.free.fr/index.htm), a distributed computing effort to find a counterexample for sixth powers.
2. **Blackjack.** Write a program Blackjack.java that takes three command line integers x, y, and z representing your two blackjack cards x and y, and the dealers face-up card z, and prints the "standard strategy" for a 6 card deck in Atlantic city. Assume that x, y, and z are integers between 1 and 10, representing an ace through a face card. Report whether the player should hit, stand, or split according to these [strategy tables](http://www.blackjackinfo.com/cgi-bin/bjbse.cgi?game=ac6). (When you learn about arrays, you will encounter an alternate strategy that does not involve as many if-else statements).
3. **Blackjack with doubling.** Modify the previous exercise to allow *doubling*.
4. **Projectile motion.** The following equation gives the trajectory of a ballistic missile as a function of the initial angle theta and windspeed: xxxx. Write a java program to print the (x, y) position of the missile at each time step t. Use trial and error to determine at what angle you should aim the missile if you hope to incinerate a target located 100 miles due east of your current location and at the same elevation. Assume the windspeed is 20 mph due east.
5. **World series.** The baseball world series is a best of 7 competition, where the first team to win four games wins the World Series. Suppose the stronger team has probability p > 1/2 of winning each game. Write a program to estimate the chance that the weaker teams wins the World Series and to estimate how many games on average it will take.
6. Consider the equation (9/4)^x = x^(9/4). One solution is 9/4. Can you find another one using Newton's method?
7. **Sorting networks.** Write a program [Sort3.java](https://introcs.cs.princeton.edu/java/13flow/Sort3.java.html) with three if statements (and no loops) that reads in three integers *a*, *b*, and *c* from the command line and prints them out in ascending order.

|  |
| --- |
| if (a > b) swap a and b  if (a > c) swap a and c  if (b > c) swap b and c |

1. **Oblivious sorting network.** Convince yourself that the following code fragment rearranges the integers stored in the variables A, B, C, and D so that A <= B <= C <= D.

|  |
| --- |
| if (A > B) { t = A; A = B; B = t; }  if (B > C) { t = B; B = C; C = t; }  if (A > B) { t = A; A = B; B = t; }  if (C > D) { t = C; C = D; D = t; }  if (B > C) { t = B; B = C; C = t; }  if (A > B) { t = A; A = B; B = t; }  if (D > E) { t = D; D = E; E = t; }  if (C > D) { t = C; C = D; D = t; }  if (B > C) { t = B; B = C; C = t; }  if (A > B) { t = A; A = B; B = t; } |

Devise a sequence of statements that would sort 5 integers. How many if statements does your program use?

1. **Optimal oblivious sorting networks.** Create a program that sorts four integers using only 5 if statements, and one that sorts five integers using only 9 if statements of the type above? Oblivious sorting networks are useful for implementing sorting algorithms in hardware. How can you check that your program works for all inputs?

*Solution*: [Sort4.java](https://introcs.cs.princeton.edu/java/13flow/Sort4.java.html) sorts 4 elements using 5 compare-exchanges. [Sort5.java](https://introcs.cs.princeton.edu/java/13flow/Sort5.java.html) sorts 5 elements using 9 compare-exchanges.

The *0-1 principle* asserts that you can verify the correctness of a (deterministic) sorting algorithm by checking whether it correctly sorts an input that is a sequence of 0s and 1s. Thus, to check that Sort5.java works, you only need to test it on the 2^5 = 32 possible inputs of 0s and 1s.

1. **Optimal oblivious sorting (challenging).** Find an optimal sorting network for 6, 7, and 8 inputs, using 12, 16, and 19 if statements of the form in the previous problem, respectively.

*Solution*: [Sort6.java](https://introcs.cs.princeton.edu/java/13flow/Sort6.java.html) is the solution for sorting 6 elements.

1. **Optimal non-oblivious sorting.** Write a program that sorts 5 inputs using only 7 comparisons. *Hint*: First compare the first two numbers, the second two numbers, and the larger of the two groups, and label them so that a < b < d and c < d. Second, insert the remaining element e into its proper place in the chain a < b < d by first comparing against b, then either a or d depending on the outcome. Third, insert c into the proper place in the chain involving a, b, d, and e in the same manner that you inserted e (with the knowledge that c < d). This uses 3 (first step) + 2 (second step) + 2 (third step) = 7 comparisons. This method was first discovered by H. B. Demuth in 1956.
2. **Weather balloon.** (Etter and Ingber, p. 123) Suppose that h(t) = 0.12t4 + 12t3 - 380t2 + 4100t + 220 represents the height of a weather balloon at time t (measured in hours) for the first 48 hours after its launch. Create a table of the height at time t for t = 0 to 48. What is its maximum height? *Solution*: t = 5.
3. Will the following code fragment compile? If so, what will it do?

|  |
| --- |
| int a = 10, b = 18;  if (a = b) System.out.println("equal");  else System.out.println("not equal"); |

*Solution*: It uses the assignment operator = instead of the equality operator == in the conditional. In Java, the result of this statement is an integer, but the compiler expects a boolean. As a result, the program will not compile. In some languages (notably C and C++), this code fragment will set the variable a to 18 and print equal without an error.

1. **Gotcha 1.** What does the following code fragment do?

|  |
| --- |
| boolean a = false;  if (a = true) System.out.println("yes");  else System.out.println("no"); |

*Solution*: it prints yes. Note that the conditional uses = instead of ==. This means that a is assigned the value true As a result, the conditional expression evaluates to true. Java is not immune to the = vs. == error described in the previous exercise. For this reason, it is much better style to use if (a) or if (!a) when testing booleans.

1. **Gotcha 2.** What does the following code fragment do?

|  |
| --- |
| int a = 17, x = 5, y = 12;  if (x > y);  {  a = 13;  x = 23;  }  System.out.println(a); |

*Solution*: Always prints 13 since there is a spurious semicolon after the if statement. Thus, the assignment statement a = 13; will be executed even though (x <= y) It is legal (but uncommon) to have a block that does not belong to a conditional statement, loop, or method.

1. **Gotcha 3.** What does the following code fragment do?

|  |
| --- |
| for (int x = 0; x < 100; x += 0.5) {  System.out.println(x);  } |

*Solution*: It goes into an infinite loop printing 0. The compound assignment statement x += 0.5 is equivalent to x = (int) (x + 0.5).

1. What does the following code fragment do?

|  |
| --- |
| int income = Integer.parseInt(args[0]);  if (income >= 311950) rate = .35;  if (income >= 174700) rate = .33;  if (income >= 114650) rate = .28;  if (income >= 47450) rate = .25;  if (income >= 0) rate = .22;  System.out.println(rate); |

It does not compile because the compile cannot guarantee that rate is initialized. Use if-else instead.

1. **Application of Newton's method.** Write a program BohrRadius.java that finds the radii where the probability of finding the electron in the 4s excited state of hydrogen is zero. The probability is given by: *(1 - 3r/4 + r2/8 - r3/192)2 e-r/2*, where *r* is the radius in units of the Bohr radius (0.529173E-8 cm). Use Newton's method. By starting Newton's method at different values of *r*, you can discover all three roots. *Hint*: use initial values of r= 0, 5, and 13. *Challenge*: explain what happens if you use an initial value of r = 4 or 12.
2. **Pepys problem.** In 1693, Samuel Pepys asked Isaac Newton which was more likely: getting at least one 1 when rolling a fair die 6 times or getting at least two 1's when rolling a fair die 12 times. Write a program [Pepys.java](https://introcs.cs.princeton.edu/java/13flow/Pepys.java.html) that uses simulation to determine the correct answer.
3. What is the value of the variable s after running the following loop when N = 1, 2, 3, 4, and 5.

|  |
| --- |
| String s = "";  for (int i = 1; i <= N; i++) {  if (i % 2 == 0) s = s + i + s;  else s = i + s + i;  } |

*Solution*: [Palindrome.java](https://introcs.cs.princeton.edu/java/13flow/Palindrome.java.html).

1. **Body mass index.** The [body mass index](http://en.wikipedia.org/wiki/Body_mass_index) (BMI) is the ratio of the weight of a person (in kilograms) to the square of the height (in meters). Write a program BMI.java that takes two command-line arguments, weight and height, computes the BMI, and prints the corresponding BMI category:
   1. Starvation: less than 15
   2. Anorexic: less than 17.5
   3. Underweight: less than 18.5
   4. Ideal: greater than or equal to 18.5 but less than 25
   5. Overweight: greater than or equal to 25 but less than 30
   6. Obese: greater than or equal to 30 but less than 40
   7. Morbidly Obese: greater than or equal to 40
2. **Reynolds number.** The *Reynolds number* is the ratio if inertial forces to viscous forces and is an important quantity in fluid dynamics. Write a program that takes in 4 command-line arguments, the diameter d, the velocity v, the density rho, and the viscosity mu, and prints the Reynold's number d \* v \* rho / mu (assuming all arguments are in SI units). If the Reynold's number is less than 2000, print laminar flow, if it's between 2000 and 4000, print transient flow, and if it's more than 4000, print turbulent flow.
3. **Wind chill revisited.** The wind chill formula from Exercise 1.2.14 is only valid if the wind speed is above 3MPH and below 110MPH and the temperature is below 50 degrees Fahrenheit and above -50 degrees. Modify your solution to print an error message if the user types in a value outside the allowable range.
4. **Point on a sphere.** Write a program to print the (x, y, z) coordinates of a random point on the surface of a sphere. Use [Marsaglia' method](http://mathworld.wolfram.com/SpherePointPicking.html): pick a random point (a, b) in the unit circle as in the do-while example. Then, set x = 2a sqrt(1 - a^2 - b^2), y = 2b sqrt(1 - a^2 - b^2), z = 1 - 2(a^2 + b^2).
5. **Powers of k.** Write a program PowersOfK.java that takes an integer K as command-line argument and prints all the positive powers of K in the Java long data type. *Note*: the constant Long.MAX\_VALUE is the value of the largest integer in long.
6. **Square root, revisited.** Why not use the loop-continuation condition (Math.abs(t\*t - c) > EPSILON) in [Sqrt.java](https://introcs.cs.princeton.edu/java/13flow/Sqrt.java.html) instead of Math.abs(t - c/t) > t\*EPSILON)?

*Solution*: Surprisingly, it can lead to inaccurate results or worse. For example, if you supply [SqrtBug.java](https://introcs.cs.princeton.edu/java/13flow/SqrtBug.java.html) with the command-line argument 1e-50, you get 1e-50 as the answer (instead of 1e-25); if you supply 16664444, you get an infinite loop!

1. What happens when you try to compile the following code fragment?

|  |
| --- |
| double x;  if (a >= 0) x = 3.14;  if (a < 0) x = 2.71;  System.out.println(x); |

*Solution*: It complains that the variable x might not have been initialized (even though we can clearly see that x will be initialized by one of the two if statements). You can avoid this problem here by using if-else.

<https://introcs.cs.princeton.edu/java/33design/>

In this section we discuss *encapsulation*, *immutability*, and *inheritance*, with particular attention to the use of these mechanisms in *data-type design* to enable modular programming, facilitate debugging, and write clear and correct code.

**Encapsulation.**

The process of separating clients from implementations by hiding information is known as *encapsulation*. We use encapsulation to enable modular programming, facilitate debugging, and clarify program code.

* *Complex numbers revisited.* [Complex.java](https://introcs.cs.princeton.edu/java/33design/Complex.java.html) has the same API as [Complex.java](https://introcs.cs.princeton.edu/java/32class/Complex.java.html), except that it represents a complex number using *polar coordinates* *r*(cos*θ*+*i*sin*θ*)

instead of *Cartesian coordinates* as *x*+*iy*

* . The idea of encapsulation is that we can substitute one of these programs for the other without changing client code.
* *Private.* When you declare an instance variable (or method) to be private, you are making it impossible for any client (code in another class) to directly access that instance variable (or method). This helps enforce encapsulation.
* *Limiting the potential for error.* Encapsulation also helps programmers ensure that their code operates as intended. To understand the problem, consider [Counter.java](https://introcs.cs.princeton.edu/java/33design/Counter.java.html), which encapsulates a single integer and ensures that the only operation that can be performed on the integer is *increment by 1*.

Without the private modifier, a client could write code like the following:

|  |
| --- |
| Counter counter = new Counter("Volusia");  counter.count = -16022; |

With the private modifier, code like this will not compile.

**Immutability.**

An object from a data type is *immutable* if its data-type value cannot change once created. An *immutable data type* is one in which all objects of that type are immutable.

* *Advantages of immutability.* We can use immutable objects in assignment statements (or as arguments and return values from methods) without having to worry about their values changing. This makes immutable type easier to reason about and debug.
* *Cost of immutability.* The main drawback of immutability is that a new object must be created for every value.
* *Final.* When you declare an instance variable as final, you are promising to assign it a value only once. This helps enforce immutability.
* *Reference types.* The final access modifier does not guarantee immutability for instance variables of mutable types. In such cases, you must make a *defensive copy*.

**Spatial vectors.**

A *spatial vector* is an abstract entity that has a *magnitude* and a *direction*. A sequence of *n* real numbers suffices to specify a vector in *n*-dimensional space. We use a boldface letter like ***x*** to denote the vector (*x*0,*x*1,…,*xn*−1).

* *API.* The basic operations on vectors are to add two vectors, multiply a vector by a scalar, compute the dot product of two vectors, and to compute the magnitude and direction, as follows:
  + *Addition*: ***x***+***y***=(*x*0+*y*0,*x*1+*y*1,…,*xn*−1+*yn*−1)

  *Vector scaling*: *α****x***=(*αx*0,*αx*1,…,*αxn*−1)

  *Dot product* ***x***⋅***y***=*x*0*y*0+*x*1*y*1+…+*xn*−1*yn*−1

  *Magnitude*: |***x***|=*x*20+*x*21+…+*x*2*n*−1−−−−−−−−−−−−−−−−−√

  *Direction*: ***x***/|***x***|=(*x*0/|***x***|,*x*1/|***x***|,…,*xn*−1/|***x***|)

These basic mathematical definitions lead immediately to an API:

* *Implementation.* [Vector.java](https://introcs.cs.princeton.edu/java/33design/Vector.java.html) is an immutable data type that implements this API. Internally, it uses an array of length *n* to store the Cartesian coordinates.
* *The this reference.* Within an instance method (or constructor), the this keyword gives us a way to refer to the object whose instance method (or constructor) is being called. For example, the magnitude() method in Vector uses the this keyword in two ways: to invoke the dot() method and as the argument to the dot() method.

|  |
| --- |
| // return the magnitude of this Vector  public double magnitude() {  return Math.sqrt(this.dot(this));  } |

**Interface inheritance (subtyping).**

Java provides the interface construct for declaring a relationship between otherwise unrelated classes, by specifying a common set of methods that each implementing class must include. Interfaces enable us to write client programs that can manipulate objects of varying types, by invoking common methods from the interface.

* *Defining an interface.* [Function.java](https://introcs.cs.princeton.edu/java/33design/Function.java.html) defines an interface for real-valued functions of a single variable.

|  |
| --- |
| public interface Function {  public abstract double evaluate (double x);  } |

The body of the interface contains a list of *abstract methods*. An abstract method is a method that is declared but does not include any implementation code; it contains only the method signature. You must save a Java interface in a file whose name matches the name of the interface, with a .java extension.

* *Implementing an interface.* To write a class that implements an interface, you must do two things.
  + Include an implements clause in the class declaration with the name of the interface.
  + Implement each of the abstract methods in the interface.

For example, [Square.java](https://introcs.cs.princeton.edu/java/33design/Square.java.html) and [GaussianPDF.java](https://introcs.cs.princeton.edu/java/33design/GaussianPDF.java.html) implements the Function interface.

* *Using an interface.* An interface is a reference type. So, you can declare the type of a variable to be the name of an interface. When you do so, any object you assign to that variable must be an instance of a class that implements the interface. For example, a variable of type Function may store an object of type Square or GaussianPDF.

|  |
| --- |
| Function f1 = new Square();  Function f2 = new GaussianPDF();  Function f3 = new Complex(1.0, 2.0); // compile-time error |

When a variable of an interface type invokes a method declared in the interface, Java knows which method to call because it knows the type of the invoking object. This powerful programming mechanism is known as *polymorphism* or *dynamic dispatch*.

* *Plotting functions.* [FunctionGraph.java](https://introcs.cs.princeton.edu/java/33design/FunctionGraph.java.html) plots the graph of a real-valued function *f* in the interval [*a*, *b*] by sampling the function at *n* + 1 evenly spaced points. It works for any sufficiently smooth function f that implements the Function interface.
* *Numerical integration.* [RectangleRule.java](https://introcs.cs.princeton.edu/java/33design/RectangleRule.java.html) estimates the integral of a positive real-valued function *f* in an interval (*a*, *b*) using the *rectangle rule*. It works for any sufficiently smooth function f that implements the Function interface.
* *Lambda expressions.* To simplify syntax, Java provides a powerful functional programming feature known as *lambda expressions*. You should think of a lambda expression as a block of code that you can pass around and execute later. In its simplest form, a lambda expression consists of the three elements:
  + A list of parameters variables, separated by commas, and enclosed in parentheses
  + The *lambda operator* ->
  + A single expression, which is the value returned by the lambda expression

For example, the following lambda expression implements the hypotenuse function:

Our primary use of lambda expressions is as a concise way to implement a *functional interface* (an interface with a single abstract method). Specifically, you can use a lambda expression wherever an object from a functional interface is expected. For example, all of the following expressions implement the [Function.java](https://introcs.cs.princeton.edu/java/33design/Function.java.html) interface:

Consequently, you can can integrate the square function with the call integrate(x -> x\*x, 0, 10, 1000), bypassing the need to define a separate Square class.

* *Built-in interfaces.* Java includes three built-in interfaces that we will consider later this book.
  + The interface [java.util.Comparable](https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html) defines an order in which to compare objects of the same type, such as alphabetical order for strings or ascending order for integers.
  + The interfaces [java.util.Iterator](https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html) and [java.lang.Iterable](https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html) enable clients to iterate over the items in a collection, without relying on the underlying representation.

**Implementation inheritance (subclassing).**

Java also supports another inheritance mechanism known as *subclassing*. The idea is to define a new class (*subclass*, or *derived class*) that inherits instance variables (state) and instance methods (behavior) from another class (*superclass*, or *base class*), enabling code reuse. Typically, the subclass redefines or *overrides* some of the methods in the superclass. For example, Java provides an elaborate inheritance hierarchy for GUI components:

In this book, we avoid subclassing because it works against encapsulation and immutability (e.g., the [fragile base class problem](https://en.wikipedia.org/wiki/Fragile_base_class) and the [circle–ellipse](https://en.wikipedia.org/wiki/Circle-ellipse_problem) problem).

* *Java's Object superclass.* Certain vestiges of subclassing are built into Java and therefore unavoidable. Specifically, every class is a subclass of [java.lang.Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html). When programming in Java, you will often override one or more of these inherited methods:
* *String conversion.* Every Java class inherits the [toString()](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#toString--) method, so any client can invoke toString() for any object. This convention is the basis for Java’s automatic conversion of one operand of the string concatenation operator + to a string whenever the other operand is a string.
* *Reference equality.* If we test equality with (x == y), where x and y are object references, we are testing whether they have the same identity: whether the *object references* are equal.
* *Object equality.* The purpose of the [equals()](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-) method is to test whether two *objects* are equal (correspond to the same data-type value). It must implement an *equivalence relation*:
  + Reflexive: x.equals(x) is true.
  + Symmetric: x.equals(y) is true if and only if y.equals(x) is true.
  + Transitive: if x.equals(y) is true and y.equals(z) is true, then x.equals(z) is true.

In addition, the following two properties must hold:

* + Multiple calls to x.equals(y) return the same truth value, provided neither object is modified between calls.
  + x.equals(null) returns false.

Overriding the equals() method is unexpectedly intricate because its argument can be a reference to an object of any type (or null).

* *Hashing.* The purpose of the [hashCode()](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode--), method is to support *hashing*, which is a fundamental operation that maps an object to an integer, known as a *hash code*. It must satisfy the following two properties:
  + If x.equals(y) is true, then x.hashCode() is equal to y.hashCode().
  + Multiple calls of x.hashCode() return the same integer, provided the object is not modified between calls.

Typically, we use the hash code to map an object x to an integer in a small range, say between 0 and m-1, using this *hash function*:

|  |
| --- |
| private int hash(Object x) {  return Math.abs(x.hashCode() % m);  } |

Objects whose values are not equal can have the same hash function value but we expect the hash function to divide n typical objects from the class into m groups of roughly equal size.

* *Wrapper types.* The toString(), hashCode(), and equals() methods apply only to reference types, not primitive types. For example, the expression x.hashCode() works if x is a variable of type Integer but not if it is of type int. For situations where we wish want to represent a value from a primitive type as an object, Java supplies built-in reference types known as *wrapper types*, one for each of the eight primitive types.
* *Autoboxing and unboxing.* Java automatically converts between values from a wrapper type and the corresponding primitive type, so that you can write code like the following:

|  |
| --- |
| Integer x = 17; // Autoboxing (int -> Integer)  int a = x; // Unboxing (Integer -> int) |

**Application: data mining.**

We consider a *data mining* application in which the goal is to associate with each document a vector known as a *sketch* so that so that documents that are different have sketches that are different and documents that are similar have sketches that are similar. Our API abstracts away this notion into the method similarTo(), which is a real number between 0 (not similar) and 1 (similar). The parameters *k* and *d* control the quality of the sketch.

* *Computing sketches.* [Sketch.java](https://introcs.cs.princeton.edu/java/33design/Sketch.java.html) uses a simple frequency count approach to compute the sketch of a text document. In its simplest form, it counts the number of time each *k*-gram (substring of length *k*) appears in the text. The sketch that we use is the direction of the vector defined by these frequencies.
* *Hashing.* For ASCII text strings there are 128 different possible values for each character, so there are 128*k* possible *k*-grams. For efficiency, [Sketch.java](https://introcs.cs.princeton.edu/java/33design/Sketch.java.html) uses *hashing*. That is, instead of counting the number of times each *k*-gram appears, we hash each *k*-gram to an integer between 0 and *d*−1 and count the number of times each hash value appears.
* *Comparing sketches.* [Sketch.java](https://introcs.cs.princeton.edu/java/33design/Sketch.java.html) uses the *cosine similarity measure* to compare two sketches:

*x*⋅*y*=*x*0*y*0+*x*1*y*1+…+*xd*−1*yd*−1

* It is a real number between 0 and 1.
* *Comparing all pairs.* [CompareDocuments.java](https://introcs.cs.princeton.edu/java/33design/CompareDocuments.java.html) prints the cosine similarity measure for all pairs of documents on an input list.

**Design by contract.**

We briefly discuss two Java language mechanisms that enable you to verify assumptions about your program while it is running—exceptions and assertions.

* *Exceptions.* An *exception* is a disruptive event that occurs *while* a program is running, often to signal an error. The action taken is known as *throwing an exception*. Java includes an elaborate inheritance hierarchy of predefined exceptions, several of which we have encountered previously.

It is good practice to use exceptions when they can be helpful to the user. For example, in [Vector.java](https://introcs.cs.princeton.edu/java/33design/Vector.java.html), we should throw an exception in plus() if the two vectors to be added have different dimensions:

|  |
| --- |
| if (this.length() != that.length())  throw new IllegalArgumentException("Dimensions disagree."); |

* *Assertions.* An *assertion* is a boolean expression that you are affirming is true at some point *during* the execution of a program. If the expression is false, the program will throw an AssertionError, which typically terminates the program and reports an error message. For example, in [Counter.java](https://introcs.cs.princeton.edu/java/33design/Counter.java.html), we might check that the counter is never negative by adding the following assertion as the last statement in increment():

|  |
| --- |
| assert count >= 0 : "Negative count detected in increment()"; |

By default, assertions are disabled, but you can enable them from the command line by using the -enableassertions flag (-ea for short). Assertions are for debugging only; your program should not rely on assertions for normal operation since they may be disabled.

In the *design-by-contract* model of programming, the designer expresses conditions about the behavior of the program using assertions.

* + *Precondition.* A condition that the client promises to satisfy when calling a method.
  + *Postcondition.* A condition that the implementation promises to achieve when returning from a method.
  + *Invariant.* A condition that the implementation promises to satisfy while the method is executing.

**Exercises**

1. Give an implementation of minus() for [Vector.java](https://introcs.cs.princeton.edu/java/33design/Vector.java.html) solely in terms of the other Vector methods, such as direction() and magnitude().

*Solution*:

|  |
| --- |
| public Vector minus(Vector that) {  return this.plus(that.scale(-1.0));  } |

1. Add a toString() method to [Vector.java](https://introcs.cs.princeton.edu/java/33design/Vector.java.html) that returns the vector components, separated by commas, and enclosed in matching parentheses.

**Creative Exercises**

1. **Statistics.** Develop a data type for maintaining statistics for a set of real numbers. Provide a method to add data points and methods that return the number of points, the mean, the standard deviation, and the variance.

*x*¯*s*2==1*n*∑*ixi*∑*i*(*xi*−*μ*)2*n*−1=*n*∑*ix*2*i*−(∑*ixi*)2*n*(*n*−1)

Develop two implementations: [OnePass.java](https://introcs.cs.princeton.edu/java/33design/OnePass.java.html) whose instance values are the number of points and the sum of the values, and the sum of the squares of the values, [TwoPass.java](https://introcs.cs.princeton.edu/java/33design/TwoPass.java.html) that keeps an array containing all the points. For simplicity, you may take the maximum number of points in the constructor. Your first implementation is likely to be faster and use substantially less space, but is also likely to be susceptible to roundoff error.

*Solution*: [StableOnePass.java](https://introcs.cs.princeton.edu/java/33design/StableOnePass.java.html) is a well-engineered alternative that is is numerically stable and does not require an array to store the elements.

*m*0*s*0*mnsnx*¯*s*2======00*mn*−1+1*n*(*xn*−*mn*−1)*sn*−1+*n*−1*n*(*xn*−*mn*−1)2*mn*1*n*−1*sn*

1. **Genome.** Develop a data type to store the genome of an organism. Biologists often abstract away the genome to a sequence of nucleotides (A, C, G, or T). The data type should support the method addNucleotide(), nucleotideAt(), as well as isPotentialGene(). Develop three implementations.
   * Use one instance variable of type String, implementing addCodon() with string concatenation. Each method call takes time proportional to the length of the current genome.
   * Use an array of characters, doubling the length of the array each time it fills up.
   * Use a boolean array, using two bits to encode each codon, and doubling the length of the array each time it fills up.

*Solution*: [StringGenome.java](https://introcs.cs.princeton.edu/java/33design/StringGenome.java.html), [Genome.java](https://introcs.cs.princeton.edu/java/33design/Genome.java.html), and [CompactGenome.java](https://introcs.cs.princeton.edu/java/33design/CompactGenome.java.html).

1. **Encapsulation.** Is the following class immutable?

|  |
| --- |
| import java.util.Date  public class Appointment {  private Date date;  private String contact;  public Appointment(Date date) {  this.date = date;  this.contact = contact;  }  public Date getDate() {  return date;  } |

*Solution*: No, because Java's [java.util.Date](https://docs.oracle.com/javase/8/docs/api/java/util/Date.html) is mutable. To correct, make a defensive copy of the date in the constructor and make a defensive copy of the date before returning to the client.

1. **Date.** Design an implementation of Java's [java.util.Date](https://docs.oracle.com/javase/8/docs/api/java/util/Date.html) API that is immutable and therefore corrects the defects of the previous exercise.

*Partial solution*: [Date.java](https://introcs.cs.princeton.edu/java/33design/Date.java.html).

**Web Exercises**

1. Add methods to [Genome.java](https://introcs.cs.princeton.edu/java/33design/Genome.java.html) to test for equality and return the reverse-complemented genome.
2. Add methods to [Date.java](https://introcs.cs.princeton.edu/java/33design/Date.java.html) to check which season (Spring, Summer, Fall, Winter) or astrological sign (Pisces, Libra, ...) a given date lies. Be careful about events that span December to January.
3. Add a method daysUntil() to [Date.java](https://introcs.cs.princeton.edu/java/33design/Date.java.html) that takes a Date as an argument and returns the number of days between the two dates.
4. Create an implementation [Date2.java](https://introcs.cs.princeton.edu/java/33design/Date2.java.html) that represents a date a single integer that counts the number of days since January 1, 1970. Compare to [Date.java](https://introcs.cs.princeton.edu/java/33design/Date.java.html).
5. Create a Rectangle ADT that represents a rectangle. Represent a rectangle by two points. Include a constructor, a toString method, a method for computing the area, and a method for drawing using our graphics library.
6. Repeat the previous exercise, but this time represent a Rectangle as the lower left endpoint and the width and height.
7. Repeat the previous exercise, but this time represent a Rectangle as the center and the width and height.
8. **Sparse vector.** Create a data type for sparse vectors. Represent a sparse vector by an array of indices (of nonzeros) and a parallel array of the corresponding nonzero values. Assume the indices are in ascending order. Implement the dot product operation.
9. **Copy constructor.** Only needed if data type is mutable. Otherwise, assignment statement works as desired.

|  |
| --- |
| public Counter(Counter x) {  count = x.count;  }  Counter counter1 = new Counter();  counter1.hit();  counter1.hit();  counter1.hit();  Counter counter2 = new Counter(counter1);  counter2.hit();  counter2.hit();  StdOut.println(counter1.get() + " " + counter2.get()); // 3 5 |

1. Define an interface [DifferentiableFunction.java](https://introcs.cs.princeton.edu/java/33design/DifferentiableFunction.java.html) for twice-differentiable function. Write a class [Sqrt.java](https://introcs.cs.princeton.edu/java/33design/Sqrt.java.html) that implements the function f(x) = c - x^2.
2. Write a program [Newton.java](https://introcs.cs.princeton.edu/java/33design/Newton.java.html) that implements Newton's method to find a real root of a sufficiently smooth function, given that you start sufficiently close to a root. When method converges, it does so quadratically. Assume that it takes a DifferentiableFunction as argument.
3. **Generating random numbers.** Different methods to generate a random number from the standard Gaussian distribution. Here, encapsulation enables us to replace one version with another that is more accurate or efficient. Trigonometric method is simple, but may be slow due to calling several transcendental functions. More importantly, it suffers from numerical stability problems when x1 is close to 0. Better method is alternate form of Box-Muller method. [reference](http://en.wikipedia.org/wiki/Box-Muller_transform). Both methods require two values from a uniform distribution and produce two values from the Gaussian distribution with mean 0 and standard deviation 1. Can save work by remembering the second value for the next call. (This is how it is implemented in java.util.Random.) Their implementation is the polar method of Box-Muller, saving the second random number for a subsequent call. (See Knuth, ACP, Section 3.4.1 Algorithm C.)
4. **LAX airport shutdown.** On September 14, 2004 Los Angeles airport was [shut down](http://msdn.microsoft.com/library/default.asp?url=%20/library/en-us/sysinfo/base/gettickcount.asp) due to software breakdown of a radio system used by air traffic controllers to communicate with pilots. The program used a Windows API function call GetTickCount() which returns the number of milliseconds since the system was last rebooted. The value is returned as a 32 bit integer, so after approximately 49.7 days it "wraps around." The software developers were aware of the bug, and instituted a policy that a technician would reboot the machine every month so that it would never exceed 31 days of uptime. Oops. LA Times blamed the technician, but the developers are more to blame for shoddy design.
5. **Polar representation of points.** Re-implement the [Point.java](https://introcs.cs.princeton.edu/java/33design/Point.java.html) data type using polar coordinates.

*Solution*: [PointPolar.java](https://introcs.cs.princeton.edu/java/33design/PointPolar.java.html).

1. **Spherical coordinates.** Represent a point in 3D space using Cartesian coordinates (*x*,*y*,*z*)

or spherical coordinates (*r*,*θ*,*ϕ*). To convert from one to the other, use

*rθϕ*===*x*2+*y*2+*z*2−−−−−−−−−−√tan−1(*y*/*x*)cos−1(*z*/*r*)*xyz*===*r*cos*θ*sin*ϕr*sin*θ*sin*ϕr*cos*ϕ*

1. **Colors.** Can represent in RGB, CMYK, or HSV formats. Natural to have different implementations of same interface.
2. **ZIP codes.** Implement an ADT that represents a USPS ZIP code. Support both the original 5 digit format and the newer (but optional) ZIP+4 format.