Media Resources

WILEY PLUS

www.wiley.com/go/global/horstmann

- ***Worked Example*** Computing the Volume and Surface Area of a Pyramid
- ***Worked Example*** Extracting Initials
- Lab Exercises
- ⊕ Practice Quiz
- ⊕ Code Completion Exercises

Review Exercises

★★ **R3.1** Write the following mathematical expressions in Java.

$$s = s_0 + v_0 t + \frac{1}{2} g t^2$$

$$G = 4\pi^2 \frac{a^3}{P^2 (m_1 + m_2)}$$

$$\text{FV} = \text{PV} \cdot \left(1 + \frac{\text{INT}}{100}\right)^{\text{YRS}}$$

$$c = \sqrt{a^2 + b^2 - 2ab\cos\gamma}$$

★★ **R3.2** Write the following Java expressions in mathematical notation.

a. `dm = m * (Math.sqrt(1 + v / c) / (Math.sqrt(1 - v / c) - 1));`

b. `volume = Math.PI * r * r * h;`

c. `volume = 4 * Math.PI * Math.pow(r, 3) / 3;`

d. `p = Math.atan2(z, Math.sqrt(x * x + y * y));`

★★★ **R3.3** What is wrong with this version of the quadratic formula?

```
x1 = (-b - Math.sqrt(b * b - 4 * a * c)) / 2 * a;
x2 = (-b + Math.sqrt(b * b - 4 * a * c)) / 2 * a;
```

★★ **R3.4** Give an example of integer overflow. Would the same example work correctly if you used floating-point?

★★ **R3.5** Give an example of a floating-point roundoff error. Would the same example work correctly if you used integers and switched to a sufficiently small unit, such as cents instead of dollars, so that the values don't have a fractional part?

★★ **R3.6** Consider the following code:

```
CashRegister register = new CashRegister();
register.recordPurchase(19.93);
register.enterPayment(20, 0, 0, 0, 0);
System.out.print("Change: ");
System.out.println(register.giveChange());
```

The code segment prints the total as `0.07000000000000028`. Explain why. Give a recommendation to improve the code so that users will not be confused.

★ **R3.7** Let `n` be an integer and `x` a floating-point number. Explain the difference between

```
n = (int) x;
```

and

```
n = (int) Math.round(x);
```

★★★ **R3.8** Let `n` be an integer and `x` a floating-point number. Explain the difference between

```
n = (int) (x + 0.5);
```

and

```
n = (int) Math.round(x);
```

For what values of `x` do they give the same result? For what values of `x` do they give different results?

★ **R3.9** Consider the vending machine implementation in How To 3.1 on page 116. What happens if the `givePennyStamps` method is invoked before the `giveFirstClassStamps` method?

★ **R3.10** Explain the differences between `2`, `2.0`, `'2'`, `"2"`, and `"2.0"`.

★ **R3.11** Explain what each of the following two program segments computes:

```
int x = 2;
int y = x + x;
```

and

```
String s = "2";
String t = s + s;
```

★★ **R3.12** True or false? (`x` is an `int` and `s` is a `String`)

a. `Integer.parseInt("" + x)` is the same as `x`

b. `"" + Integer.parseInt(s)` is the same as `s`

c. `s.substring(0, s.length())` is the same as `s`

★★ **R3.13** How do you get the first character of a string? The last character? How do you remove the first character? The last character?

★★★ **R3.14** How do you get the last digit of an integer? The first digit? That is, if `n` is `23456`, how do you find out that the first digit is `2` and the last digit is `6`? Do not convert the number to a string. *Hint:* `%`, `Math.log`.

★★ **R3.15** This chapter contains several recommendations regarding variables and constants that make programs easier to read and maintain. Summarize these recommendations.

★★★ **R3.16** What is a `final` variable? Can you declare a `final` variable without supplying its value? (Try it out.)

★ **R3.17** What are the values of the following expressions? In each line, assume that

```
double x = 2.5;
double y = -1.5;
int m = 18;
int n = 4;
```

a. `x + n * y - (x + n) * y`

b. `m / n + m % n`

c. `5 * x - n / 5`
d. `Math.sqrt(Math.sqrt(n))`
e. `(int) Math.round(x)`
f. `(int) Math.round(x) + (int) Math.round(y)`
g. `1 - (1 - (1 - (1 - (1 - n))))`

★ **R3.18** What are the values of the following expressions? In each line, assume that

```
int n = 4;
String s = "Hello";
String t = "World";
```

a. `s + t`
b. `s + n`
c. `n + t`
d. `s.substring(1, n)`
e. `s.length() + t.length()`

Programming Exercises

★ **P3.1** Enhance the `CashRegister` class by adding separate methods `enterDollars`, `enterQuarters`, `enterDimes`, `enterNickels`, and `enterPennies`.

Use this tester class:

```
public class CashRegisterTester
{
   public static void main (String[] args)
   {
      CashRegister register = new CashRegister();
      register.recordPurchase(20.37);
      register.enterDollars(20);
      register.enterQuarters(2);
      System.out.println("Change: " + register.giveChange());
      System.out.println("Expected: 0.13");
   }
}
```

★ **P3.2** Enhance the `CashRegister` class so that it keeps track of the total number of items in a sale. Count all recorded purchases and supply a method

```
int getItemCount()
```

that returns the number of items of the current purchase. Remember to reset the count at the end of the purchase.

★★ **P3.3** Implement a class `IceCreamCone` with methods `getSurfaceArea()` and `getVolume()`. In the constructor, supply the height and radius of the cone. Be careful when looking up the formula for the surface area—you should only include the outside area along the side of the cone since the cone has an opening on the top to hold the ice cream.

★★ **P3.4** Write a program that prompts the user for two numbers, then prints

- The sum
- The difference
- The product

- The average
- The distance (absolute value of the difference)
- The maximum (the larger of the two)
- The minimum (the smaller of the two)

To do so, implement a class

```
public class Pair
{
   /**
      Constructs a pair.
      @param aFirst the first value of the pair
      @param aSecond the second value of the pair
   */
   public Pair(double aFirst, double aSecond) { . . . }

   /**
      Computes the sum of the values of this pair.
      @return the sum of the first and second values
   */
   public double getSum() { . . . }
   . . .
}
```

Then implement a class `PairTester` that constructs a `Pair` object, invokes its methods, and prints the results.

★ **P3.5** Declare a class `DataSet` that computes the sum and average of a sequence of integers. Supply methods

- `void addValue(int x)`
- `int getSum()`
- `double getAverage()`

Hint: Keep track of the sum and the count of the values.

Then write a test program `DataSetTester` that calls `addValue` four times and prints the expected and actual results.

★★ **P3.6** Write a class `DataSet` that computes the largest and smallest values in a sequence of numbers. Supply methods

- `void addValue(int x)`
- `int getLargest()`
- `int getSmallest()`

Keep track of the smallest and largest values that you've seen so far. Then use the `Math.min` and `Math.max` methods to update them in the `addValue` method. What should you use as initial values? *Hint:* `Integer.MIN_VALUE`, `Integer.MAX_VALUE`.

Write a test program `DataSetTester` that calls `addValue` four times and prints the expected and actual results.

★ **P3.7** Write a program that prompts the user for a measurement in meters and then converts it into miles, feet, and inches. Use a class

```
public class Converter
{
   /**
      Constructs a converter that can convert between two units.
      @param aConversionFactor the factor by which to multiply
      to convert to the target unit
   */
   public Converter(double aConversionFactor) { . . . }

   /**
      Converts from a source measurement to a target measurement.
      @param fromMeasurement the measurement
      @return the input value converted to the target unit
   */
   public double convertTo(double fromMeasurement) { . . . }

   /**
      Converts from a target measurement to a source measurement.
      @param toMeasurement the target measurement
      @return the value whose conversion is the target measurement
   */
   public double convertFrom(double toMeasurement) { . . . }
}
```

In your ConverterTester class, construct and test the following Converter object:

```
final double MILE_TO_KM = 1.609;
Converter milesToMeters = new Converter(1000 * MILE_TO_KM);
```

★ **P3.8** Write a class Square whose constructor receives the length of the sides. Then supply methods to compute

- The area and perimeter of the square
- The length of the diagonal (use the Pythagorean theorem)

★★ **P3.9** Implement a class SodaCan whose constructor receives the height and diameter of the soda can. Supply methods getVolume and getSurfaceArea. Supply a SodaCanTester class that tests your class.

★★★ **P3.10** Implement a class Balloon that models a spherical balloon that is being filled with air. The constructor constructs an empty balloon. Supply these methods:

- void addAir(double amount) adds the given amount of air
- double getVolume() gets the current volume
- double getSurfaceArea() gets the current surface area
- double getRadius() gets the current radius

Supply a BalloonTester class that constructs a balloon, adds 100 cm^3 of air, tests the three accessor methods, adds another 100 cm^3 of air, and tests the accessor methods again.

★★ **P3.11** *Giving change.* Enhance the CashRegister class so that it directs a cashier how to give change. The cash register computes the amount to be returned to the customer, in pennies. Add the following methods to the CashRegister class:

- int giveDollars()
- int giveQuarters()
- int giveDimes()
- int giveNickels()
- int givePennies()

Each method computes the number of dollar bills or coins to return to the customer, and reduces the change due by the returned amount. You may assume that the methods are called in this order. Here is a test class:

```
public class CashRegisterTester
{
   public static void main(String[] args)
   {
      CashRegister register = new CashRegister();

      register.recordPurchase(8.37);
      register.enterPayment(10, 0, 0, 0, 0);
      System.out.println("Dollars: " + register.giveDollars());
      System.out.println("Expected: 1");
      System.out.println("Quarters: " + register.giveQuarters());
      System.out.println("Expected: 2");
      System.out.println("Dimes: " + register.giveDimes());
      System.out.println("Expected: 1");
      System.out.println("Nickels: " + register.giveNickels());
      System.out.println("Expected: 0");
      System.out.println("Pennies: " + register.givePennies());
      System.out.println("Expected: 3");
   }
}
```

★★ **P3.12** In How To 3.1 on page 116, we represented the state of the vending machine by storing the balance in pennies. This is ingenious, but it is perhaps not the most obvious solution. Another possibility is to store the number of dollars that the customer inserted and the change that remains after giving out the first class stamps. Reimplement the vending machine in this way. Of course, the public interface should remain unchanged.

★★★ **P3.13** Write a program that reads in an integer and breaks it into a sequence of individual digits in reverse order. For example, the input 16384 is displayed as

```
4
8
3
6
1
```

You may assume that the input has no more than five digits and is not negative.

Declare a class DigitExtractor:

```
public class DigitExtractor
{
   /**
      Constructs a digit extractor that gets the digits
      of an integer in reverse order.
      @param anInteger the integer to break up into digits
   */
   public DigitExtractor(int anInteger) { . . . }

   /**
      Returns the next digit to be extracted.
      @return the next digit
   */
   public int nextDigit() { . . . }
}
```

In your main class DigitPrinter, call System.out.println(myExtractor.nextDigit()) five times.

★★ **P3.14** Implement a class `QuadraticEquation` whose constructor receives the coefficients `a`, `b`, `c` of the quadratic equation $ax^2 + bx + c = 0$. Supply methods `getSolution1` and `getSolution2` that get the solutions, using the quadratic formula. Write a test class `QuadraticEquationTester` that constructs a `QuadraticEquation` object, and prints the two solutions.

★★★ **P3.15** Write a program that reads two times in military format (0900, 1730) and prints the number of hours and minutes between the two times. Here is a sample run. User input is in color.

```
Please enter the first time: 0900
Please enter the second time: 1730
8 hours 30 minutes
```

Extra credit if you can deal with the case where the first time is later than the second:

```
Please enter the first time: 1730
Please enter the second time: 0900
15 hours 30 minutes
```

Implement a class `TimeInterval` whose constructor takes two military times. The class should have two methods `getHours` and `getMinutes`.

★ **P3.16** *Writing large letters.* A large letter `H` can be produced like this:

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

Use the class

```
public class LetterH
{
   public String toString()
   {
      return "*   *\n*   *\n*****\n*   *\n*   *\n";
   }
}
```

Declare similar classes for the letters `E`, `L`, and `O`. Then write the message

```
H
E
L
L
O
```

in large letters.

★★ **P3.17** Write a class `ChristmasTree` whose `toString` method yields a string depicting a Christmas tree:

```
   /\
  /  \
 /    \
/      \
--------
  "  "
  "  "
  "  "
```

Remember to use escape sequences.

★★ **P3.18** Your job is to transform numbers `1`, `2`, `3`, . . ., `12` into the corresponding month names `January`, `February`, `March`, . . ., `December`. Implement a class `Month` whose constructor parameter is the month number and whose `getName` method returns the month name. *Hint:* Make a very long string `"January February March . . . "`, in which you add spaces such that each month name has the same length. Then use `substring` to extract the month you want.

★★ **P3.19** Write a class to compute the date of Easter Sunday. Easter Sunday is the first Sunday after the first full moon of spring. Use this algorithm, invented by the mathematician Carl Friedrich Gauss in 1800:

1. Let `y` be the year (such as 1800 or 2001).
2. Divide `y` by `19` and call the remainder `a`. Ignore the quotient.
3. Divide `y` by `100` to get a quotient `b` and a remainder `c`.
4. Divide `b` by `4` to get a quotient `d` and a remainder `e`.
5. Divide `8 * b + 13` by `25` to get a quotient `g`. Ignore the remainder.
6. Divide `19 * a + b - d - g + 15` by `30` to get a remainder `h`. Ignore the quotient.
7. Divide `c` by `4` to get a quotient `j` and a remainder `k`.
8. Divide `a + 11 * h` by `319` to get a quotient `m`. Ignore the remainder.
9. Divide `2 * e + 2 * j - k - h + m + 32` by `7` to get a remainder `r`. Ignore the quotient.
10. Divide `h - m + r + 90` by `25` to get a quotient `n`. Ignore the remainder.
11. Divide `h - m + r + n + 19` by `32` to get a remainder `p`. Ignore the quotient.

Then Easter falls on day `p` of month `n`. For example, if `y` is `2001`:

```
a = 6                g = 6                r = 6
b = 20               h = 18               n = 4
c = 1                j = 0, k = 1         p = 15
d = 5, e = 0         m = 0
```

Therefore, in 2001, Easter Sunday fell on April 15. Write a class `Easter` with methods `getEasterSundayMonth` and `getEasterSundayDay`.

Programming Projects

Project 3.1 In this project, you will perform calculations with triangles. A triangle is defined by the *x*- and *y*-coordinates of its three corner points.

Your job is to compute the following properties of a given triangle:

- the lengths of all sides
- the angles at all corners
- the perimeter
- the area

Of course, you should implement a `Triangle` class with appropriate methods. Supply a program that prompts a user for the corner point coordinates and produces a nicely formatted table of the triangle properties.

This is a good team project for two students. Both students should agree on the `Triangle` interface. One student implements the `Triangle` class, the other simultaneously implements the user interaction and formatting.

Project 3.2 The `CashRegister` class has an unfortunate limitation: It is closely tied to the coin system in the United States and Canada. Research the system used in most of Europe. Your goal is to produce a cash register that works with euros and cents. Rather than designing another limited `CashRegister` implementation for the European market, you should design a separate `Coin` class and a cash register that can work with coins of all types.

Answers to Self-Check Questions

1. `int` and `double`.
2. The world's most populous country, China, has about 1.2 x 10^9 inhabitants. Therefore, individual population counts could be held in an `int`. However, the world population is over 6×10^9. If you compute totals or averages of multiple countries, you can exceed the largest `int` value. Therefore, `double` is a better choice. You could also use `long`, but there is no benefit because the exact population of a country is not known at any point in time.
3. The first initialization is incorrect. The right hand side is a value of type `double`, and it is not legal to initialize an `int` variable with a `double` value. The second initialization is correct—an `int` value can always be converted to a `double`.
4. The first declaration is used inside a method, the second inside a class.
5. (1) You should use a named constant, not the "magic number" 3.14.
   (2) 3.14 is not an accurate representation of π.
6. One less than it was before.
7. 17 and 29.
8. Only `s3` is divided by 3. To get the correct result, use parentheses. Moreover, if `s1`, `s2`, and `s3` are integers, you must divide by `3.0` to avoid integer division:

   ```
   (s1 + s2 + s3) / 3.0
   ```
9. $\sqrt{x^2 + y^2}$
10. When the fractional part of `x` is ≥ 0.5.
11. By using a cast: `(int) Math.round(x)`.
12. `x` is a number, not an object, and you cannot invoke methods on numbers.
13. No—the `println` method is called on the object `System.out`.
14. `s` is set to the string "`Agent5`".
15. The strings `"i"` and `"ssissi"`.
16. The class only has a method to read a single byte. It would be very tedious to form characters, strings, and numbers from those bytes.
17. The value is `"John"`. The `next` method reads the next *word*.

Use the Boolean data type to store and combine conditions that can be true or false.

- The `boolean` type has two values: `true` and `false`.
- A predicate method returns a `boolean` value.
- You can form complex tests with the Boolean operators `&&` (*and*), `||` (*or*), and `!` (*not*).
- You can store the outcome of a condition in a Boolean variable.

Design test cases that cover all parts of a program.

- Black-box testing describes a testing method that does not take the structure of the implementation into account.
- White-box testing uses information about the structure of a program.
- Code coverage is a measure of how many parts of a program have been tested.
- Boundary test cases are test cases that are at the boundary of acceptable inputs.
- You should calculate test cases by hand to double-check that your application computes the correct answer.

Use the Java logging library for messages that can be easily turned on or off.

- Logging messages can be deactivated when testing is complete.

Classes, Objects, and Methods Introduced in this Chapter

```
java.lang.Character
   isDigit
   isLetter
   isLowerCase
   isUpperCase
java.lang.Object
   equals
java.lang.String
   equals
   equalsIgnoreCase
   compareTo
java.util.Scanner
   hasNextDouble
   hasNextInt
java.util.logging.Level
   INFO
   OFF
java.util.logging.Logger
   getGlobal
   info
   setLevel
```

Media Resources

WILEY PLUS

www.wiley.com/go/global/horstmann

- ***Worked Example*** Extracting the Middle
- Lab Exercises
- ⊕ Practice Quiz
- ⊕ Code Completion Exercises

Review Exercises

★ **R4.1** What is the value of each variable after the `if` statement?

a. `int n = 1; int k = 2; int r = n; if (k < n) r = k;`
b. `int n = 1; int k = 2; int r; if (n < k) r = k; else r = k + n;`
c. `int n = 1; int k = 2; int r = k; if (r < k) n = r; else k = n;`
d. `int n = 1; int k = 2; int r = 3; if (r < n + k) r = 2 * n; else k = 2 * r;`

★★ **R4.2** Find the errors in the following `if` statements.

a. `if (1 + x > Math.pow(x, Math.sqrt(2)) y = y + x;`
b. `if (x = 1) y++; else if (x = 2) y = y + 2;`
c.
```
int x = Integer.parseInt(input);
if (x != null) y = y + x;
```

★★ **R4.3** Find the error in the following `if` statement that is intended to select a language from a given country and state/province.

```
language = "English";
if (country.equals("Canada"))
   if (stateOrProvince.equals("Quebec")) language = "French";
else if (country.equals("China"))
   language = "Chinese";
```

★★ **R4.4** Find the errors in the following `if` statements.

a. `if (x && y == 0) { x = 1; y = 1; }`
b.
```
if (1 <= x <= 10)
   System.out.println(x);
```
c.
```
if (!s.equals("nickels") || !s.equals("pennies")
      || !s.equals("dimes") || !s.equals("quarters"))
   System.out.print("Input error!");
```
d.
```
if (input.equalsIgnoreCase("N") || "NO")
   return;
```

★ **R4.5** Explain the following terms, and give an example for each construct:

a. Expression
b. Condition
c. Statement
d. Simple statement
e. Compound statement
f. Block

★ **R4.6** Explain the difference between an `if` statement with multiple `else` branches and nested `if` statements. Give an example for each.

★ **R4.7** Give an example for an `if/else if/else` statement where the order of the tests does not matter. Give an example where the order of the tests matters.

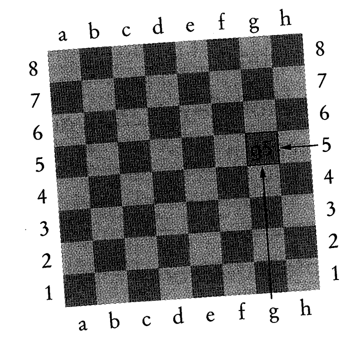
★ **R4.8** Of the following pairs of strings, which comes first in lexicographic order?

a. `"Tom", "Jerry"`
b. `"Tom", "Tomato"`
c. `"church", "Churchill"`
d. `"car manufacturer", "carburetor"`
e. `"Harry", "hairy"`
f. `"C++", " Car"`
g. `"Tom", "Tom"`
h. `"Car", "Carl"`
i. `"car", "bar"`
j. `"101", "11"`
k. `"1.01", "10.1"`

★ **R4.9** Complete the following truth table by finding the truth values of the Boolean expressions for all combinations of the Boolean inputs p, q, and r.

p	q	r	(p && q) || !r	!(p && (q || !r))
false	false	false		
false	false	false		
false	false	false		
. . .	. . .	. . .		
5 more combinations				
. . .				

★★ **R4.10** Each square on a chess board can be described by a letter and number, such as g5 in this example:



The following pseudocode describes an algorithm that determines whether a square with a given letter and number is dark (black) or light (white).

```
If the letter is an a, c, e, or g
    If the number is odd
        color = "black"
    Else
        color = "white"
Else
    If the number is even
        color = "black"
    Else
        color = "white"
```

Using the procedure in Productivity Hint 4.2 on page 155, trace this pseudocode with input g5.

★ **R4.11** Give a set of four test cases for the algorithm of Exercise R4.10 that covers all branches.

★★ **R4.12** In a scheduling program, we want to check whether two appointments overlap. For simplicity, appointments start at a full hour, and we use military time (with hours 0–24). The following pseudocode describes an algorithm that determines whether the appointment with start time **start1** and end time **end1** overlaps with the appointment with start time **start2** and end time **end2**.

```
If start1 > start2
    s = start1
Else
    s = start2
If end1 < end2
    e = endl
Else
    e = end2
If s < e
    The appointments overlap.
Else
    The appointments don't overlap.
```

Trace this algorithm with an appointment from 10–12 and one from 11–13, then with an appointment from 10–11 and one from 12–13.

★ **R4.13** Write pseudocode for a program that prompts the user for a month and day and prints out whether it is one of the following four holidays:

- New Year's Day (January 1)
- Independence Day (July 4)
- Veterans Day (November 11)
- Christmas Day (December 25)

★★★ **R4.14** True or false? *A* && *B* is the same as *B* && *A* for any Boolean conditions *A* and *B*.

★ **R4.15** Explain the difference between

```
s = 0;
if (x > 0) s++;
if (y > 0) s++;
```

and

```
s = 0;
if (x > 0) s++;
else if (y > 0) s++;
```

★★ **R4.16** Use de Morgan's law to simplify the following Boolean expressions.

a. `!(x > 0 && y > 0)`
b. `!(x != 0 || y != 0)`
c. `!(country.equals("US") && !state.equals("HI")`
`        && !state.equals("AK"))`
d. `!(x % 4 != 0 || !(x % 100 == 0 && x % 400 == 0))`

★★ **R4.17** Make up another Java code example that shows the dangling `else` problem, using the following statement: A student with a GPA of at least 1.5, but less than 2, is on probation; with less than 1.5, the student is failing.

★ **R4.18** Explain the difference between the `==` operator and the `equals` method when comparing strings.

★ **R4.19** Explain the difference between the tests

```
r == s
```

and

```
r.equals(s)
```

where both r and s are of type Rectangle.

★★ **R4.20** What is wrong with this test to see whether r is null? What happens when this code runs?

```
Rectangle r;
. . .
if (r.equals(null))
   r = new Rectangle(5, 10, 20, 30);
```

★ **R4.21** Explain how the lexicographic ordering of strings differs from the ordering of words in a dictionary or telephone book. *Hint:* Consider strings, such as IBM, wiley.com, Century 21, While-U-Wait, and 7-11.

★★★ **R4.22** Write Java code to test whether two objects of type Line2D.Double represent the same line when displayed on the graphics screen. *Do not* use a.equals(b).

```
Line2D.Double a;
Line2D.Double b;

if (your condition goes here)
   g2.drawString("They look the same!", x, y);
```

Hint: If p and q are points, then Line2D.Double(p, q) and Line2D.Double(q, p) look the same.

★ **R4.23** Explain why it is more difficult to compare floating-point numbers than integers. Write Java code to test whether an integer n equals 10 and whether a floating-point number x is approximately equal to 10.

★★ **R4.24** Consider the following test to see whether a point falls inside a rectangle.

```
Point2D.Double p = . . .
Rectangle r = . . .
boolean xInside = false;
if (r.getX() <= p.getX() && p.getX() <= r.getX() + r.getWidth())
   xInside = true;
boolean yInside = false;
if (r.getY() <= p.getY() && p.getY() <= r.getY() + r.getHeight())
   yInside = true;
if (xInside && yInside)
   g2.drawString("p is inside the rectangle.",
         p.getX(), p.getY());
```

Rewrite this code to eliminate the explicit true and false values, by setting xInside and yInside to the values of Boolean expressions.

★T **R4.25** Give a set of test cases for the earthquake program in Section 4.3.1. Ensure coverage of all branches.

★T **R4.26** Give an example of a boundary test case for the earthquake program in Section 4.3.1. What result do you expect?

Programming Exercises

★★ **P4.1** Write a program that prints all real solutions to the quadratic equation $ax^2 + bx + c = 0$. Read in a, b, c and use the quadratic formula. If the *discriminant* $b^2 - 4ac$ is negative, display a message stating that there are no real solutions.

Implement a class QuadraticEquation whose constructor receives the coefficients a, b, c of the quadratic equation. Supply methods getSolution1 and getSolution2 that get the solutions, using the quadratic formula, or 0 if no solution exists. The getSolution1 method should return the smaller of the two solutions.

Supply a method

```
boolean hasSolutions()
```

that returns false if the discriminant is negative.

★★ **P4.2** Write a program that takes user input describing a playing card in the following shorthand notation:

Notation	Meaning
A	Ace
2 ... 10	Card values
J	Jack
Q	Queen
K	King
D	Diamonds
H	Hearts
S	Spades
C	Clubs

Your program should print the full description of the card. For example,

```
Enter the card notation:
4S
Four of spades
```

Implement a class Card whose constructor takes the card notation string and whose getDescription method returns a description of the card. If the notation string is not in the correct format, the getDescription method should return the string "Unknown".

★★ **P4.3** Write a program that reads in three floating-point numbers and prints the three inputs in sorted order. For example:

```
Please enter three numbers:
4
9
2.5
The inputs in sorted order are:
2.5
4
9
```

★ **P4.4** Write a program that translates a letter grade into a number grade. Letter grades are A B C D F, possibly followed by + or -. Their numeric values are 4, 3, 2, 1, and 0. There is no F+ or F-. A + increases the numeric value by 0.3, a - decreases it by 0.3. However, an A+ has the value 4.0. All other inputs have value –1.

```
Enter a letter grade:
B-
Numeric value: 2.7.
```

Use a class Grade with a method getNumericGrade.

★ **P4.5** Write a program that translates a number into the closest letter grade. For example, the number 2.8 (which might have been the average of several grades) would be converted to B-. Break ties in favor of the better grade; for example, 2.85 should be a B. Any value ≥ 4.15 should be an A+.

Use a class Grade with a method getLetterGrade.

★ **P4.6** Write a program that reads in three strings and prints them in lexicographically sorted order:

```
Please enter three strings:
Tom
Dick
Harry
The inputs in sorted order are:
Dick
Harry
Tom
```

★★ **P4.7** Change the implementation of the getTax method in the TaxReturn class, by setting a variable rate1_limit, depending on the marital status. Then have a single formula that computes the tax, depending on the income and the limit. Verify that your results are identical to that of the TaxReturn class in this chapter.

★★★ **P4.8** The original U.S. income tax of 1913 was quite simple. The tax was

- 1 percent on the first $50,000.
- 2 percent on the amount over $50,000 up to $75,000.
- 3 percent on the amount over $75,000 up to $100,000.
- 4 percent on the amount over $100,000 up to $250,000.
- 5 percent on the amount over $250,000 up to $500,000.
- 6 percent on the amount over $500,000.

There was no separate schedule for single or married taxpayers. Write a program that computes the income tax according to this schedule.

★★ **P4.9** Write a program that prompts for the day and month of the user's birthday and then prints a horoscope. Make up fortunes for programmers, like this:

```
Please enter your birthday (month and day): 6 16
Gemini are experts at figuring out the behavior of complicated programs.
You feel where bugs are coming from and then stay one step ahead. Tonight,
your style wins approval from a tough critic.
```

Each fortune should contain the name of the astrological sign. (You will find the names and date ranges of the signs at a distressingly large number of sites on the Internet.)

★ **P4.10** When two points in time are compared, each given as hours (in military time, ranging from 0 and 23) and minutes, the following pseudocode determines which comes first.

```
If hour1 < hour2
   time1 comes first.
Else if hour1 and hour2 are the same
   If minute1 < minute2
      time1 comes first.
   Else if minute1 and minute2 are the same
      time1 and time2 are the same.
   Else
      time2 comes first.
Else
   time2 comes first.
```

Write a program that prompts the user for two points in time and prints the time that comes first, then the other time.

★ **P4.11** The following algorithm yields the season (Spring, Summer, Fall, or Winter) for a given month and day.

```
If month is 1, 2, or 3, season = "Winter"
Else if month is 4, 5, or 6, season = "Spring"
Else if month is 7, 8, or 9, season = "Summer"
Else if month is 10, 11, or 12, season = "Fall"
If month is divisible by 3 and day >= 21
   If season is "Winter", season = "Spring"
   Else if season is "Spring", season = "Summer"
   Else if season is "Summer", season = "Fall"
   Else season = "Winter"
```

Write a program that prompts the user for a month and day and then prints the season, as determined by this algorithm.

★ **P4.12** A year with 366 days is called a *leap year*. A year is a leap year if it is divisible by 4 (for example, 1980). However, since the introduction of the Gregorian calendar on October 15, 1582, a year is not a leap year if it is divisible by 100 (for example, 1900); however, it is a leap year if it is divisible by 400 (for example, 2000). Write a program that asks the user for a year and computes whether that year is a leap year. Implement a class Year with a predicate method boolean isLeapYear().

★ **P4.13** Write a program that asks the user to enter a month (1 = January, 2 = February, and so on) and then prints the number of days of the month. For February, print "28 days".

```
Enter a month (1-12):
5
31 days
```

Implement a class Month with a method int getDays(). Do not use a separate if or else statement for each month. Use Boolean operators.

★★★ **P4.14** Write a program that reads in two floating-point numbers and tests (a) whether they are the same when rounded to two decimal places and (b) whether they differ by less than 0.01.

Here are two sample runs.

```
Enter two floating-point numbers:
2.0
1.99998
They are the same when rounded to two decimal places.
They differ by less than 0.01.

Enter two floating-point numbers:
0.999
0.991
They are different when rounded to two decimal places.
They differ by less than 0.01.
```

★ **P4.15** Enhance the `BankAccount` class of Chapter 2 by

- Rejecting negative amounts in the `deposit` and `withdraw` methods
- Rejecting withdrawals that would result in a negative balance

★ **P4.16** Write a program that reads in the hourly wage of an employee. Then ask how many hours the employee worked in the past week. Be sure to accept fractional hours. Compute the pay. Any overtime work (over 40 hours per week) is paid at 150 percent of the regular wage. Solve this problem by implementing a class `Paycheck`.

★★ **P4.17** Write a unit conversion program that asks users to identify the unit from which they want to convert and the unit to which they want to convert. Legal units are *in*, *ft*, *mi*, *mm*, *cm*, *m*, and *km*. Declare two objects of a class `UnitConverter` that convert between meters and a given unit.

```
Convert from:
in
Convert to:
mm
Value:
10
10 in = 254 mm
```

★★★ **P4.18** A line in the plane can be specified in various ways:

- by giving a point (x, y) and a slope m
- by giving two points (x_1, y_1), (x_2, y_2)
- as an equation in slope-intercept form $y = mx + b$
- as an equation $x = a$ if the line is vertical

Implement a class `Line` with four constructors, corresponding to the four cases above. Implement methods

```
boolean intersects(Line other)
boolean equals(Line other)
boolean isParallel(Line other)
```

★★G **P4.19** Write a program that draws a circle with radius 100 and center (200, 200). Ask the user to specify the x- and y-coordinates of a point. Draw the point as a small circle. If the point lies inside the circle, color the small circle green. Otherwise, color it red. In your exercise, declare a class `Circle` and a method `boolean isInside(Point2D.Double p)`.

★★★G **P4.20** Write a graphics program that asks the user to specify the radii of two circles. The first circle has center (100, 200), and the second circle has center (200, 100). Draw the circles. If they intersect, then color both circles green. Otherwise, color them red. *Hint:* Compute the distance between the centers and compare it to the radii. Your program should draw nothing if the user enters a negative radius. In your exercise, declare a class `Circle` and a method `boolean intersects(Circle other)`.

Programming Projects

Project 4.1 Implement a *combination lock* class. A combination lock has a dial with 26 positions labeled A . . . Z. The dial needs to be set three times. If it is set to the correct combination, the lock can be opened. When the lock is closed again, the combination can be entered again. If a user sets the dial more than three times, the last three settings determine whether the lock can be opened. An important part of this exercise is to implement a suitable interface for the `CombinationLock` class.

Project 4.2 Get the instructions for last year's form 1040 from `http://www.irs.ustreas.gov`. Find the tax brackets that were used last year for all categories of taxpayers (single, married filing jointly, married filing separately, and head of household). Write a program that computes taxes following that schedule. Ignore deductions, exemptions, and credits. Simply apply the tax rate to the income.

Answers to Self-Check Questions

1. If the withdrawal amount equals the balance, the result should be a zero balance and no penalty.
2. Only the first assignment statement is part of the `if` statement. Use braces to group both assignment statements into a block statement.
3. (a) 0; (b) 1; (c) An exception occurs.
4. Syntactically incorrect: e, g, h. Logically questionable: a, d, f
5. Yes, if you also reverse the comparisons:

```
if (richter < 3.5)
   r = "Generally not felt by people";
else if (richter < 4.5)
   r = "Felt by many people, no destruction";
else if (richter < 6.0)
   r = "Damage to poorly constructed buildings";
. . .
```

6. The higher tax rate is only applied on the income in the higher bracket. Suppose you are single and make \$31,900. Should you try to get a \$200 raise? Absolutely: you get to keep 90 percent of the first \$100 and 75 percent of the next \$100.
7. When `x` is zero.
8. `if (!Character.isDigit(ch)) . . .`
9. Seven
10. An input of 0 should yield an output of `"Generally not felt by people"`. (If the output is `"Negative numbers are not allowed"`, there is an error in the program.)