APPENDIX B

Learning About Data Representation

In this appendix, you will:

- Work with numbering systems
- Represent numeric values
- Represent character values

Unless noted otherwise, all images are © 2014 Cengage Learning

Copyright 2013 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Understanding Numbering Systems

You can use devices such as computers, cell phones, microwave ovens, and automobiles without understanding how they work internally. Likewise, you can write many Java programs without understanding how the data items they use are represented internally. However, once you learn how data items are stored, you gain a deeper understanding of computer programming in general and Java in particular. You also can more easily troubleshoot some types of problems that arise in your programs.

The numbering system you know best is the **decimal numbering system**, which is based on 10 digits, 0 through 9. When you use the decimal system, no other symbols are available; if you want to express a value larger than 9, you must use multiple digits from the same pool of 10, placing them in columns. Decimal numbers are also called base 10 numbers.

When you use the decimal system, you analyze a multicolumn number by mentally assigning place values to each column. The value of the rightmost column is 1, the value of the next column to the left is 10, the next column's value is 100, and so on; you multiply the column value by 10 as you move to the left. There is no limit to the number of columns you can use; you simply add them to the left as you need to express higher values. For example, Figure B-1 shows how the value 305 is represented in the decimal system. You simply multiply the digit in each column by the value of the column, and then add the values together.

```
100s   10s   1s          Value is
                           3 × 100 = 300
 3      0     5            0 × 10  =   0
                           5 × 1   =   5
                                   -----
                                     305
```

Figure B-1 Representing 305 in the decimal system

The **binary numbering system** works in the same way as the decimal numbering system, except that it uses only two digits, *0* and *1*. When you use the binary system and you want to express a value greater than 1, you must use multiple columns because no single symbol represents any value other than 0 or 1. Instead of each new column to the left being 10 times greater than the previous column, each new binary column is only two times the value of the previous column. Binary numbers are called base 2 numbers.

For example, Figure B-2 shows how the numbers 9 and 305 are represented in the binary system. Notice that both the binary and decimal systems allow you to create numbers with 0 in one or more columns. As with the decimal system, the binary system has no limit to the number of columns—you can use as many as it takes to express a value.

Copyright 2013 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

8s	4s	2s	1s
1	0	0	1

Value is	
1 × 8	= 8
0 × 4	= 0
0 × 2	= 0
1 × 1	= 1
	9

256s	128s	64s	32s	16s	8s	4s	2s	1s
1	0	0	1	1	0	0	0	1

Value is	
1 × 256	= 256
0 × 128	= 0
0 × 64	= 0
1 × 32	= 32
1 × 16	= 16
0 × 8	= 0
0 × 4	= 0
0 × 2	= 0
1 × 1	= 1
	305

Figure B-2 Representing decimal values 9 and 305 in the binary system

Copyright 2013 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

A computer stores every piece of data it uses as a set of *0*s and *1*s. Each *0* or *1* is known as a **bit**, which is short for *binary digit*. Every computer uses *0*s and *1*s because all its values are stored as electronic signals that are either on or off. This two-state system is most easily represented using just two digits.

Representing Numeric Values

In Chapter 2, you learned that a floating-point number contains decimal positions. The term *floating-point* comes from the fact that the decimal point can be at any location in the stored value, allowing a much larger range of possible values to be stored in the same amount of memory. For example, assume that a computer could store only four digits and that the decimal point had to fall after the first two. The positive values that could be stored would then range from 00.00 through 99.99. However, if the decimal point could fall anywhere, the values could range from .0000 through 9999. Computers use more storage for each value, and store negative values as well, but the principle is the same.

Because of the binary nature of computers, representing floating-point numbers is imprecise. For example, suppose you want to represent the value 1/10 (0.10). You could try using each of the following techniques:

- If you use two bits to store the value, only four combinations are available (00, 01, 10, and 11), so they can only represent 0/4, 1/4, 2/4 (or 1/2), and 3/4. None of these is exactly 1/10, but 0/4 is the closest.
- Suppose you use three bits. This allows twice as many combinations, or eight, and the closest to 1/10 is 1/8. The approximation is closer than with two bits, but still not exact.
- Suppose you use four bits, which allows 16 combinations. The closest value to 1/10 is 2/16. This value is no closer to 1/10 than you could achieve with three bits.
- Suppose you use eight bits. Now, there are 256 bit combinations from 0/256 through 255/256. The value of 26/256, at 0.1015625, is closer than any of the other values so far, but it's still not exact.
- No matter how many bits you add to the representation, doubling the number of combinations each time, you can never express 0.1 exactly.

Although you cannot store 0.1 exactly, you can still display it. For example, the following two lines of code display 0.1 as expected:

```
double oneTenth = 0.1;
System.out.println(oneTenth);
```

When Java displays a floating-point number, it always displays at least one digit after the decimal. After that, it uses only as many digits as necessary to distinguish the number from the nearest floating-point value it can represent.

However, when you use 0.1 in an arithmetic statement, the imprecision becomes evident. Figure B-3 shows a simple program that declares two variables named `oneTenth` and `threeTenths`; the variables contain the values 0.1 and 0.3, respectively. Figure B-4 shows the

Copyright 2013 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

result of summing `oneTenth` three times and of comparing that sum to `threeTenths`. Because of floating-point imprecision, the first value is calculated to be slightly more than 0.3, so the comparison of `oneTenth + oneTenth + oneTenth` to `threeTenths` is `false`.

```
import java.util.Scanner;
public class FloatingPointTest
{
   public static void main(String[] args)
   {
      double oneTenth = 0.1;
      double threeTenths = 0.3;
      System.out.println(oneTenth + oneTenth + oneTenth);
      System.out.println(oneTenth + oneTenth + oneTenth ==
        threeTenths);
   }
}
```

Figure B-3 The `FloatingPointTest` class

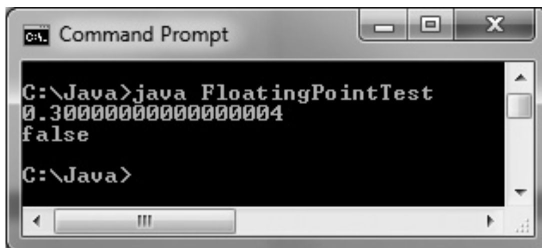

Figure B-4 Output of the `FloatingPointTest` program

For many purposes, you do not care about the small imprecisions generated by floating-point calculations, but sometimes they can make a difference. For example, several popular movies have used the idea that small amounts of extra money can be sliced off bank balances when compounding interest and then siphoned to a criminal's account. Many programmers recommend that you use the Java class `BigDecimal` when working with monetary or scientific values where precision is important. Additionally, be aware that when you test two floating-point values for equivalency, you might not get the expected results.

When precision is not an issue, but better-looking output is important, you can format the output to eliminate the small imprecisions that occur far to the right of the decimal point. Appendix C teaches you many techniques for formatting output to a desired number of decimal places.

Copyright 2013 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Representing Character Values

The characters used in Java are represented in **Unicode**, which is a 16-bit coding scheme for characters. For example, the letter *A* actually is stored in computer memory as a set of 16 zeros and ones as 0000 0000 0100 0001 (a space is inserted after each set of four digits for readability). Because 16-digit numbers are difficult to read, programmers often use a shorthand notation called the **hexadecimal numbering system**, or base 16. The hexadecimal system uses 16 values, 0 through 9 and A through F, to represent the decimal values 0 through 15. In hexadecimal shorthand, 0000 becomes 0, 0100 becomes 4, and 0001 becomes 1, so the letter *A* is represented in hexadecimal as 0041. You tell the compiler to treat the four-digit hexadecimal 0041 as a single character by preceding it with the \u escape sequence. Therefore, there are two ways to store the character *A*:

```
char letter = 'A';
char letter = '\u0041';
```

For more information about Unicode, go to *www.unicode.org*.

The second option, using hexadecimal, is obviously more difficult and confusing than the first method, so it is not recommended that you store letters of the alphabet using hexadecimal values. However, you can produce some interesting output using the Unicode format. For example, the sequence '\u0007' produces a bell-like noise if you send it to output. Letters from foreign alphabets that use characters instead of letters (Greek, Hebrew, Chinese, and so on) and other special symbols (foreign currency symbols, mathematical symbols, geometric shapes, and so on) are available using Unicode but not on a standard keyboard, so it may be important that you know how to use Unicode characters.

Two-digit, base 16 numbers can be converted to base 10 numbers by multiplying the left digit by 16 and adding the right digit. For example, hexadecimal 41 is 4 times 16 plus 1, or 65.

In the United States, the most widely used character set traditionally has been **ASCII** (American Standard Code for Information Interchange). The ASCII character set contains 128 characters. You can create any Unicode character by adding eight *0*s to the beginning of its ASCII character equivalent. This means that the decimal value of any ASCII character is the same as that of the corresponding Unicode character. For example, *B* has the value 66 in both character sets. The decimal values are important because they allow you to show nonprintable characters, such as a carriage return, in decimal codes. Also, the numeric values of the coding schemes are used when a computer sorts numbers and strings. When you sort characters in ascending order, for example, numbers are sorted first (because their Unicode values begin with decimal code 48), followed by capital letters (starting with decimal 65), and then lowercase letters (starting with decimal 97).

Copyright 2013 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Chapter 2 contains a list of Unicode values for some commonly used characters. For a complete list, see *www.unicode.org/charts*. There you will find Greek, Armenian, Hebrew, Tagalog, Cherokee, and a host of other character sets. Unicode also contains characters for mathematical symbols, geometric shapes, and other unusual characters. The ASCII character set is more limited than Unicode because it contains only letters and symbols used in the English language.

Key Terms

The **decimal numbering system** is based on 10 digits, 0 through 9, in which each column represents a value 10 times higher than the column to its right.

The **binary numbering system** is based on two digits, 0 and 1, in which each column represents a value two times higher than the column to its right.

A **bit** is each binary digit, 0 or 1, used to represent computerized values.

Unicode is a 16-bit coding scheme for representing characters.

The **hexadecimal numbering system** is based on 16 digits, 0 through F, in which each column represents a value 16 times higher than the column to its right.

ASCII (American Standard Code for Information Interchange) is a character set widely used to represent computer data.

Copyright 2013 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2013 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.