CHAPTER

2

USING DATA WITHIN A PROGRAM

In this chapter, you will:

- Use constants and variables
- Learn about the int data type
- Write arithmetic statements
- Use the Boolean data type
- Learn about floating-point data types
- Understand numeric type conversion
- Work with the char data type
- Learn about ASCII and Unicode

How are you doing with your first programs?" asked Lynn Greenbrier during a coffee break. "OK, I think," I replied with just a bit of doubt in my voice. "I sure wish I could do some calculations, though," I continued. "Writing code that only prints the output I coded using println() statements isn't exactly what I had in mind when I considered a job in programming." "Well then," Lynn replied, "let's start learning how Java uses different data types to perform arithmetic and other kinds of calculations."

Using Constants and Variables

You can categorize data as variable or constant. Data is **constant** when it cannot be changed after a program is compiled; data is **variable** when it might change. For example, if you include the statement `System.out.println(459);` in a Java program, the number 459 is a constant. Every time the program containing the constant 459 is executed, the value 459 will print. You can refer to the number 459 as a **literal constant** because its value is taken literally at each use.

Besides using literal constants, you can use symbolic constants, which you will learn about in Chapter 4.

On the other hand, you can set your data up as a variable. For example, if you create a variable named `ovenTemperature`, and include the statement `System.out.println(ovenTemperature);` within a Java program, then different values might display when the program is executed multiple times, depending on what value is stored in `ovenTemperature` each time the program is run.

Variables are named memory locations that your program can use to store values. The Java programming language provides for eight primitive types of data:

- `boolean`
- `float`
- `byte`
- `int`
- `char`
- `long`
- `double`
- `short`

The eight primitive data types are called **primitive types** because they are simple and uncomplicated. Primitive types also serve as the building blocks for more complex data types, called reference types. The objects you begin creating in Chapter 3 are examples of reference types.

Declaring Variables

You name variables using the same naming rules for legal class identifiers described in Chapter 1. Basically, variable names must start with a letter and cannot be any reserved

keyword. You must declare all variables you want to use in a program. A **variable declaration** includes the following:

- A data type that identifies the type of data that the variable will store
- An identifier that is the variable's name
- An optional assigned value, when you want a variable to contain an initial value
- An ending semicolon

Variable names usually begin with lowercase letters to distinguish them from class names. However, variable names can begin with either an uppercase or a lowercase letter.

For example, the variable declaration `int myAge = 25;` declares a variable of type int named myAge and assigns it an initial value of 25. This is a complete statement that ends in a semicolon. The equals sign (=) is the **assignment operator**. Any value to the right of the equals sign is assigned to the variable on the left of the equals sign. An assignment made when you declare a variable is an **initialization**; an assignment made later is simply an **assignment**. Thus, `int myAge = 25;` initializes myAge to 25, and a subsequent statement `myAge = 42;` might assign a new value to the variable. You should note that the expression `25 = myAge` is illegal.

The variable declaration `int myAge;` also declares a variable of type int named myAge, but no value is assigned at the time of creation.

You can declare multiple variables of the same type in separate statements on different lines. For example, the following statements declare two variables—the first variable is named myAge and its value is 25. The second variable is named yourAge and its value is 19.

```
int myAge = 25;
int yourAge = 19;
```

You also can declare two variables of the same type in a single statement by separating the variable declarations with a comma, as shown in the following statement:

```
int myAge = 25, yourAge = 19;
```

However, if you want to declare variables of different types, you must use a separate statement for each type. The following statements declare two variables of type int (myAge and yourAge) and two variables of type double (mySalary and yourSalary):

```
int myAge, yourAge;
double mySalary, yourSalary;
```

Learning about the int Data Type

In the Java programming language, you use variables of type int to store (or hold) **integers**, or whole numbers. An integer can hold any whole number value from –2,147,483,648 to 2,147,483,647. When you assign a value to an int variable, you do not type any commas; you type only digits and an optional plus or minus sign to indicate a positive or negative integer.

The legal integer values are -2^{31} through $2^{31}-1$. These are the highest and lowest values that you can store in four bytes of memory, which is the size of an int variable.

The types **byte**, **short**, and **long** are all variations of the integer type. You use byte or short if you know a variable will need to hold only small values so you can save space in memory. You use a long if you know you will be working with very large values. Table 2-1 shows the upper and lower value limits for each of these types. It is important to choose appropriate types for the variables you will use in a program. If you attempt to assign a value that is too large for the data type of the variable, the compiler will issue an error message and the program will not execute. If you choose a data type that is larger than you need, you waste memory. For example, a personnel program might use a byte variable for number of dependents (because a limit of 127 is more than enough), a short for hours worked in a month (because 127 isn't enough), and an integer for an annual salary (because even though a limit of 32,000 might be large enough for your salary, it isn't enough for the CEO).

If your program uses a literal constant integer, such as 932, the integer is an int by default. If you need to use a constant higher than 2,147,483,647, you must follow the number with the letter L to indicate long. For example, `long mosquitosInTheNorthWoods = 2444555888L;` stores a number that is greater than the maximum limit for the int type. You can type either an uppercase or lowercase L to indicate the long type, but the uppercase L is preferred to avoid confusion with the number one.

Table 2-1 Limits on integer values by type

Type	Minimum Value	Maximum Value	Size in Bytes
byte	–128	127	1
short	–32,768	32,767	2
int	–2,147,483,648	2,147,483,647	4
long	–9,223,372,036,854,775,808	9,223,372,036,854,775,807	8

Next you will write a program to declare and display numeric values.

To declare and display values in a program:

1. Open a new document in your text editor.
2. Create a class header and an opening and closing curly brace for a new class named **DemoVariables** by typing the following:

```
public class DemoVariables
{
}
```

3. Position the insertion point after the opening curly brace, press **[Enter]**, press **[Spacebar]** several times to indent the line, and then type the following main() method and its curly braces:

```
public static void main(String[] args)
{
}
```

4. Position the insertion point after the opening curly brace in the main() method, press **[Enter]**, press **[Spacebar]** several times to indent the line, and then type `int oneInt = 315;` to declare a variable of type int named oneInt with a value of 315.

You can declare variables at any point within a method prior to their first use. However, it is common practice to declare variables first, and place method calls second.

5. Press **[Enter]** at the end of the oneInt declaration statement, indent the line, and then type the following two output statements. The first statement uses the print() method to output "The int is " and leaves the insertion point on the same output line. The second statement uses the println() method to output the value of oneInt and advances the insertion point to a new line.

```
System.out.print("The int is ");
System.out.println(oneInt);
```

When your output contains a literal string such as "The int is ", you should type a space before the closing quotation mark so there is a space between the end of the literal string and the value that prints.

6. Save the file as **DemoVariables.java** in the Chapter.02 folder on your Student Disk.
7. Compile the file from the command line by typing `javac DemoVariables.java`. If necessary, correct any errors, save the file, and then compile again.
8. Execute the program from the command line by typing `java DemoVariables`. The output is shown in Figure 2-1.

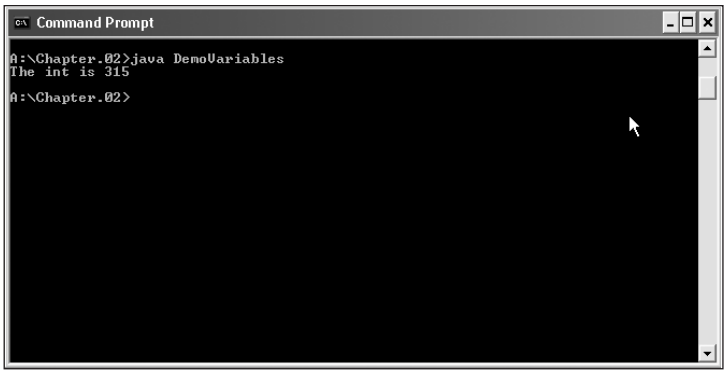

Figure 2-1 Output of the DemoVariables program

Even though you intend to add additional statements to the DemoVariables program, you compiled and executed the program at this point to make sure that it is working exactly as intended. Sometimes it is a good idea to write and compile your programs in steps, so you can identify any syntax or logical errors as you go, instead of waiting until you finish writing the entire program. Next you will declare two more variables in your program.

To declare two more variables in the program:

1. Return to the **DemoVariables.java** file in the text editor. Rename the class **DemoVariables2**.
2. Position the insertion point at the end of the line that contains the oneInt declaration, press **[Enter]**, and then type the following variable declarations on separate lines:

```
short oneShort = 23;
long oneLong = 1234567876543L;
```

3. Position the insertion point at the end of the line that contains the println() method that displays the oneInt value, press **[Enter]**, and then type the following statements to display the values of the two new variables:

```
System.out.print("The short is ");
System.out.println(oneShort);
System.out.print("The long is ");
System.out.println(oneLong);
```

4. Save the program using the filename **DemoVariables2.java**.
5. Compile the program by typing **`javac DemoVariables2.java`**. If necessary, correct any errors, save the file, and then compile again.
6. Execute the program by typing **`java DemoVariables2`**. The output is shown in Figure 2-2.

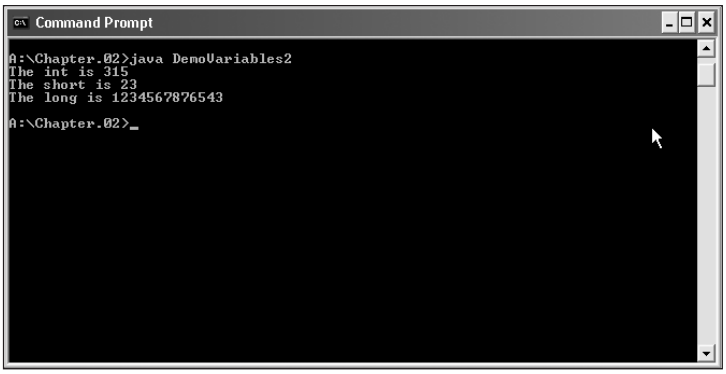

Figure 2-2 Output of the DemoVariables2 program

In the previous program, you used two print methods to print a compound phrase with the following code:

```
System.out.print("The long is ");
System.out.println(oneLong);
```

To reduce the amount of typing, you can use one method and combine the arguments with a plus sign using the following statement: `System.out.println("The long is " + oneLong);`. It doesn't matter which format you use—the result is the same, as you will see next.

To change the two print methods into a single statement:

1. Open the **DemoVariables2.java** text file, and rename the class **DemoVariables3**.
2. Use the mouse to select the two statements that print "The int is " and the value of oneInt, and then press **[Delete]** to delete them. In place of the deleted statements, type the following println() statement: `System.out.println("The int is " + oneInt);`.
3. Select the two statements that produce output for the short variable, press **[Delete]** to delete them, and then type the statement `System.out.println("The short is " + oneShort);`.
4. Finally, select the two statements that produce output for the long variables, delete them, and replace them with `System.out.println("The long is " + oneLong);`.
5. Save, compile, and test the program. The output is shown in Figure 2-3.

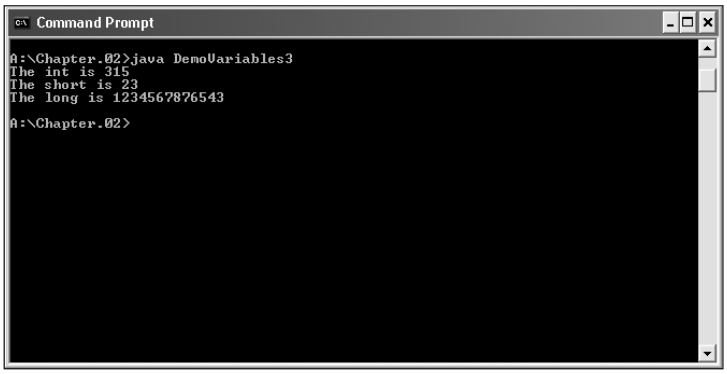

Figure 2-3 Output of the DemoVariables3 program

Writing Arithmetic Statements

Table 2-2 describes the five standard arithmetic operators for integers. You use the arithmetic operators to manipulate values in your programs.

You will learn about the shortcut arithmetic operators for the Java programming language in Chapter 5.

Table 2-2 Integer arithmetic operators

Operator	Description	Example
+	Addition	45 + 2, the result is 47
–	Subtraction	45 – 2, the result is 43
*	Multiplication	45 * 2, the result is 90
/	Division	45 / 2, the result is 22 (not 22.5)
%	Modulus (remainder)	45 % 2, the result is 1 (that is, 45 / 2 = 22 with a remainder of 1)

You do not need to perform a division operation before you can perform a modulus operation. A modulus operation can stand alone.

The operators / and % deserve special consideration. When you divide two integers, whether they are integer constants or integer variables, the result is an integer. In other words, any fractional part of the result is lost. For example, the result of 45 / 2 is 22, even though the result is 22.5 in a mathematical expression. When you use the modulus operator with two integers, the result is an integer with the value of the remainder after division

takes place—the result of 45 % 2 is 1 because 2 "goes into" 45 twenty-two times with a remainder of 1.

Next you will add some arithmetic statements to the DemoVariables3.java program.

To use arithmetic statements in a program:

1. Open the **DemoVariables3.java** file in your text editor, and change the class to **DemoVariables4**.
2. Position the insertion point at the end of the last line of the current variable declarations, press **[Enter]**, and then type the following declarations:

```
int value1 = 43, value2 = 10, sum, difference,
  product, quotient, modulus;
```

3. Position the insertion point after the statement that prints the oneLong variable, press **[Enter]**, and then type the following statements on separate lines:

```
sum = value1 + value2;
difference = value1 - value2;
product = value1 * value2;
quotient = value1 / value2;
modulus = value1 % value2;
```

4. Press **[Enter]**, and then type the following output statements:

```
System.out.println("Sum is " + sum);
System.out.println("Difference is " + difference);
System.out.println("Product is " + product);
System.out.println("Quotient is " + quotient);
System.out.println("Modulus is " + modulus);
```

5. Save the program as **DemoVariables4.java**.
6. Compile and run the program. Your output should look like Figure 2-4. Analyze the output and confirm that the arithmetic is correct.

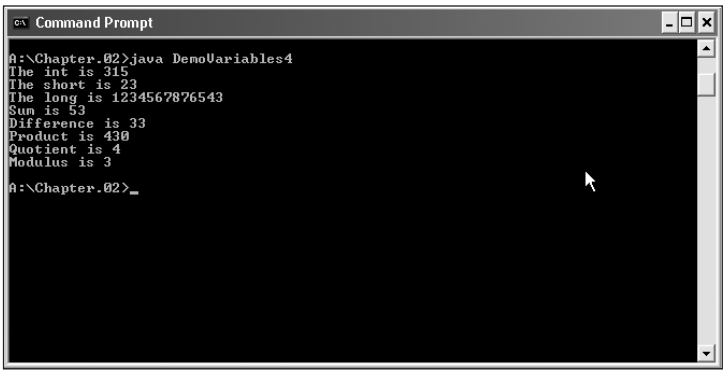

Figure 2-4 Output of the DemoVariables4 program

When you combine mathematical operations in a single statement, you must understand operator **precedence**, or the order in which parts of a mathematical expression are evaluated. Multiplication, division, and modulus always take place prior to addition or subtraction in an expression. For example, the expression `int result = 2 + 3 * 4;` results in 14, because the multiplication (3 * 4) occurs before adding 2. You can override normal operator precedence by putting the operation to perform first in parentheses. The statement `int result = (2 + 3) * 4;` results in 20, because the addition within the parentheses takes place first, and then that result (5) is multiplied by 4.

You will learn more about operator precedence in Chapter 5.

Using The Boolean Data Type

Boolean logic is based on true-or-false comparisons. Whereas an int variable can hold millions of different values (at different times), a **Boolean variable** can hold only one of two values—`true` or `false`. The following statements declare and assign appropriate values to Boolean variables:

```
boolean isItPayday = false;
boolean areYouBroke = true;
```

You also can assign values based on the result of comparisons to Boolean variables. The Java programming language supports six comparison operators. A **comparison operator** compares two items; an expression containing a comparison operator has a Boolean value. Table 2-3 describes the comparison operators.

You will learn about other Boolean operators in Chapter 5.

Table 2-3 Comparison operators

Operator	Description	`true` Example	`false` Example
<	Less than	3 < 8	8 < 3
>	Greater than	4 > 2	2 > 4
==	Equal to	7 == 7	3 == 9
<=	Less than or equal to	5 <= 5	8 <= 6
>=	Greater than or equal to	7 >= 3	1 >= 2
!=	Not equal to	5 != 6	3 != 3

When you use any of the operators that have two symbols (==, <=, >=, or !=), you cannot place any whitespace between the two symbols.

Legal, declaration statements might include the following statements, which compare two values directly:

```
boolean isSixBigger = (6 > 5);
  // Value stored would be true
boolean isSevenSmallerOrEqual = (7 <= 4);
  // Value stored would be false
```

Variable names are easily identified as Boolean if you use a form of "to be" (such as "is" or "are") as part of the variable name.

The Boolean expressions are more meaningful when variables (that have been assigned values) are used in the comparisons, as in the following examples. In the first statement, the hours variable is compared to a constant value of 40. If the hours variable is not greater than 40, then the expression evaluates to `false`. In the second statement, the income variable must be greater than 100000 for the expression to evaluate to true.

```
boolean overtime = (hours > 40);
boolean highTaxBracket = (income > 100000);
```

Next you will add two Boolean variables to the DemoVariables4.java file.

To add Boolean variables to a program:

1. Open the **DemoVariables4.java** file in your text editor and change the class to **DemoVariables5**.

2. Position the insertion point at the end of the line with the integer variable declarations, press **[Enter]**, and then type **`boolean isProgrammingFun = true, isProgrammingHard = false;`** on one line to add two new Boolean variables to the program.

   Next add some print statements to display the values.

3. Press **[Enter]**, and then type the following statements:

   ```
   System.out.println("The value of isProgrammingFun is "
     + isProgrammingFun);
   System.out.println("The value of isProgrammingHard is "
     + isProgrammingHard);
   ```

4. Save the file as **DemoVariables5.java**, compile it, and then test the program. The output appears in Figure 2-5.

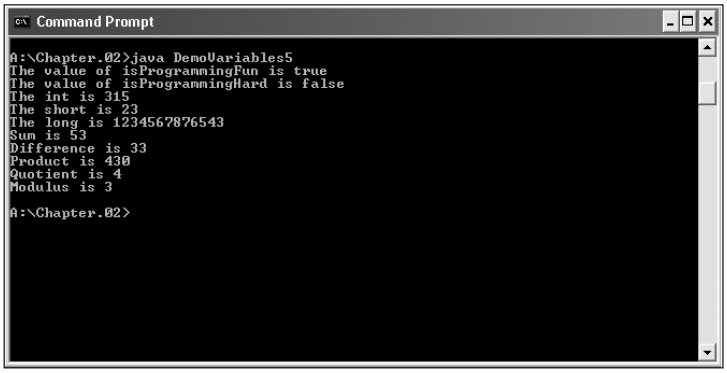

Figure 2-5 Output of the DemoVariables5 program

Learning about Floating-Point Data Types

A **floating-point** number contains decimal positions. The Java programming language supports two floating-point data types: float and double. A **float** data type can hold values up to six or seven significant digits of accuracy. A **double** data type can hold 14 or 15 significant digits of accuracy. The term **significant digits** refers to the mathematical accuracy of a value. For example, a float given the value 0.324616777 will display as 0.324617 because the value is only accurate to the sixth decimal position. Table 2-4 shows the minimum and maximum values for each data type.

A float given the value 324616777 will display as 3.24617e+008, which means approximately 3.24617 times 10 to the 8th power, or 324617000. The *e* stands for exponent; the format is called scientific notation. The large value contains only six significant digits.

Table 2-4 Limits on floating-point values

Type	Minimum	Maximum	Size in Bytes
Float	$-3.4 * 10^{38}$	$3.4 * 10^{38}$	4
Double	$-1.7 * 10^{308}$	$1.7 * 10^{308}$	8

A value written as $-3.4 * 10^{38}$ indicates that the value is –3.4 multiplied by 10 to the 38th power, or 10 with 38 trailing zeros—a very large number.

Just as an integer constant, such as 178, is an int by default, a floating-point number constant such as 18.23 is a double by default. To store a value explicitly as a float, you can type the letter *F* after the number, as in `float pocketChange = 4.87F;`. You can

type either a lowercase or an uppercase *F*. You also can type *D* (or *d*) after a floating-point value to indicate it is a double, but even without the *D*, the value will be stored as a double by default.

As with ints, you can perform the mathematical operations of addition, subtraction, multiplication, and division with floating-point numbers; however, you cannot perform modulus operations using floating-point values. (Floating-point division yields a floating-point result, so there is no remainder.)

Next you will add some floating-point variables to the DemoVariables5.java file and perform arithmetic with them.

To add floating-point variables to the program:

1. Open the **DemoVariables5.java** file in your text editor and change the class name to **DemoVariables6**.
2. Position the insertion point after the line that declares the Boolean variables, press **[Enter]**, and then type **`double doubNum1 = 2.3, doubNum2 = 14.8, doubResult;`** on one line to add some new floating-point variables.
3. Press **[Enter]**, and then type the following statements to perform arithmetic and produce output:

```
doubResult = doubNum1 + doubNum2;
System.out.println("The sum of the doubles is "
  + doubResult);
doubResult = doubNum1 * doubNum2;
System.out.println("The product of the doubles is "
  + doubResult);
```

4. Save the file as **DemoVariables6.java**, compile it, and then run the program. The output is shown in Figure 2-6.

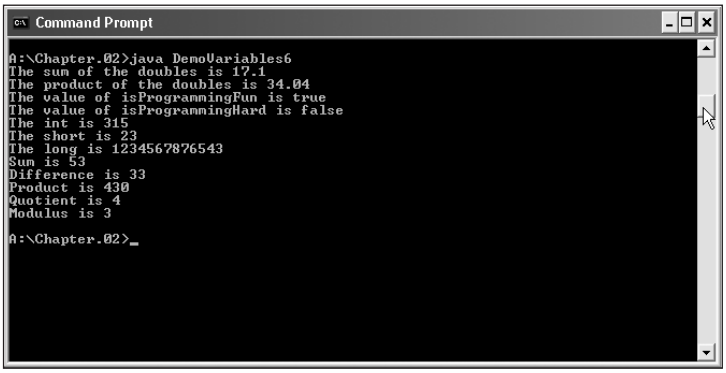

Figure 2-6 Output of the DemoVariables6 program

Understanding Numeric Type Conversion

When you are performing arithmetic with variables or constants of the same type, the result of the arithmetic retains the same type. For example, when you divide two integers, the result is an integer, and when you subtract two doubles, the result is a double. Often, however, you might want to perform mathematical operations on unlike types. In the following example you multiply an integer by a double:

```
int hoursWorked = 37;
double payRate = 6.73;
double grossPay = hoursWorked * payRate;
```

When you perform arithmetic operations with operands of unlike types, the Java programming language chooses a **unifying type** for the result. The Java programming language then **implicitly** (or automatically) converts nonconforming operands to the unifying type. The following list of operands ranks the order for establishing unifying types between two variables:

1. double
2. float
3. long
4. int
5. short
6. byte

An operand is simply any value used in an arithmetic or logical operation.

In other words, grossPay is the result of multiplication of an int and a double, so grossPay itself must be a double. Similarly, the addition of a short and an int results in an int.

You can **explicitly** (or purposely) override the unifying type imposed by the Java programming language by performing a type cast. **Type casting** involves placing the desired result type in parentheses, followed by the variable or constant to be cast. For example, two type casts are performed in the following code:

```
double bankBalance = 189.66;
float weeklyBudget = (float) bankBalance / 4;
  // weeklyBudget is 47.415, one-fourth of bankBalance
int dollars = (int) weeklyBudget;
  // dollars is 47, the integer part of weeklyBudget
```

It is easy to lose data when performing a cast. For example, the largest byte value is 127 and the largest int value is 2,147,483,647, so the following statements produce distorted results:

```
int anOkayInt = 200;
byte aBadByte = (byte)anOkayInt;
```

A byte is constructed from eight 1s and 0s, or binary digits. The first binary digit, or bit, holds a 0 or 1 to represent positive or negative. The remaining seven bits store the actual value. When the integer value 200 is stored in the byte variable, its large value consumes the eighth bit, turning it to a 1, and forcing the aBadByte variable to appear to hold the value –72, which is inaccurate and misleading.

The double value `bankBalance / 4` is converted to a float before it is stored in weeklyBudget, and the float value weeklyBudget is converted to an int before it is stored in dollars. When the float value is converted to an int, the decimal place values are lost.

Working with the char Data Type

You use the **char** data type to hold any single character. You place constant character values within single quotation marks because the computer stores characters and integers differently. For example, the statements `char aCharValue = '9';` and `int aNumValue = 9;` are legal. The statements `char aCharValue = 9;` and `int aNumValue = '9';` might produce undesirable results. If these variables are used in a println statement such as `System.out.println("aCharValue is" + aCharValue + "aNumValue is " + aNumValue);` the resulting output is a blank and the number 57, which are ASCII codes. Figure 2-7 shows ASCII decimal codes and character equivalents, which are covered later in this chapter. A number can be a character, but it must be enclosed in single quotation marks and declared as a char type. However, you cannot store an alphabetic letter in a numeric type. The following code shows how you can store any character string using the char data type:

```
char myInitial = 'J';
char percentSign = '%';
char numThatIsAChar = '9';
```

A variable of type char can hold only one character. To store a string of characters, such as a person's name, you must use a data structure called a **String**. Unlike single characters, which use single quotation marks, string constants are written between double quotation marks. For example, the expression that stores the name Audrey as a string in a variable named firstName is `string firstName = "Audrey";`.

You will learn more about Strings in Chapter 7.

You can store any character—including nonprinting characters such as a backspace or a tab—in a char variable. To store these characters, you must use an **escape sequence**, which always begins with a backslash. For example, the following code stores a backspace character and a tab character in the char variables aBackspaceChar and aTabChar:

```
char aBackspaceChar = '\b';
char aTabChar = '\t';
```

In the preceding code, the escape sequence indicates a unique value for the character, instead of the letters b or t. Table 2-5 describes some common escape sequences that are used in the Java programming language.

Table 2-5 Common escape sequences

Escape Sequence	Description
\b	Backspace
\t	Tab
\n	Newline or linefeed
\f	Form feed
\r	Carriage return
\"	Double quotation mark
\'	Single quotation mark
\\	Backslash

Learning about ASCII and Unicode

The characters used in the Java programming language 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 (the space inserted after each set of four digits is for readability). Because 16-digit numbers are difficult to read, programmers often use a shorthand notation called **hexadecimal**, or **base 16**. 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 *http://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 the hexadecimal method. However, there are some interesting values you can produce using the Unicode format. For example, the sequence '\u0007' is a bell that produces a 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 is important that you know how to use Unicode characters.

The most widely used character set is ASCII (American Standard Code for Information Interchange). There are 128 characters in the ASCII character set, which are shown in Figure 2-7 with their decimal code or numerical code and equivalent character representation. The first 32 characters and the last character are control characters and are nonprintable. These characters can be entered by holding down [Ctrl] and pressing a letter on the keyboard. For example, The Tab key or ^I(Ctrl I) produces a character 9, which produces a hard tab when pressed.

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	nul	32		64	@	96	`
1	soh ^A	33	!	65	A	97	a
2	stx ^B	34	"	66	B	98	b
3	etx ^C	35	#	67	C	99	c
4	eot ^D	36	$	68	D	100	d
5	enq ^E	37	%	69	E	101	e
6	ack ^F	38	&	70	F	102	f
7	bel ^G	39	'	71	G	103	g
8	bs ^H	40	(	72	H	104	h
9	ht ^I	41	)	73	I	105	i
10	lf ^J	42	*	74	J	106	j
11	vt ^K	43	+	75	K	107	k
12	ff ^L	44	,	76	L	108	l
13	cr ^M	45	-	77	M	109	m
14	so ^N	46	.	78	N	110	n
15	si ^O	47	/	79	O	111	o
16	dle ^P	48	0	80	P	112	p
17	dc1 ^Q	49	1	81	Q	113	q
18	dc2 ^R	50	2	82	R	114	r
19	dc3 ^S	51	3	83	S	115	s
20	dc4 ^T	52	4	84	T	116	t
21	nak ^U	53	5	85	U	117	u
22	syn ^V	54	6	86	V	118	v
23	etb ^W	55	7	87	W	119	w
24	can ^X	56	8	88	X	120	x
25	em ^Y	57	9	89	Y	121	y
26	sub ^Z	58	:	90	Z	122	z
27	esc	59	;	91	[	123	{
28	fs	60	<	92	\	124	|
29	gs	61	=	93	]	125	}
30	rs	62	>	94	^	126	~
31	us	63	?	95	_	127	del

Figure 2-7 ASCII Character Set

The relationship of ASCII and Unicode is that by adding eight zeros to any ASCII character gives the character's value in Unicode. The ASCII values are important because they allow you to show nonprintable characters, such as a carriage return, in decimal codes. Also, ASCII codes are often used when sorting numbers and strings. So when you need to sort characters in ascending order, numbers beginning with decimal 48 or 0 are sorted first, then capital letters starting with decimal 65 or A, and then lowercase letters starting with decimal code 97 or a.

Next you will add statements to your DemoVariables6.java file to use the \n and \t escape sequences.

To use escape sequences in a program:

1. Open the **DemoVariables6.java** file in your text editor and change the class name to **DemoVariables7**.
2. Position the insertion point after the last method line in the program, press **[Enter]**, and then type the following:

```
System.out.println("\nThis is on one line\nThis on
   another");
System.out.println("This shows\thow\ttabs\twork");
```

3. Save the file as **DemoVariables7.java**, compile, and then test the program. Your output should look like Figure 2-8.

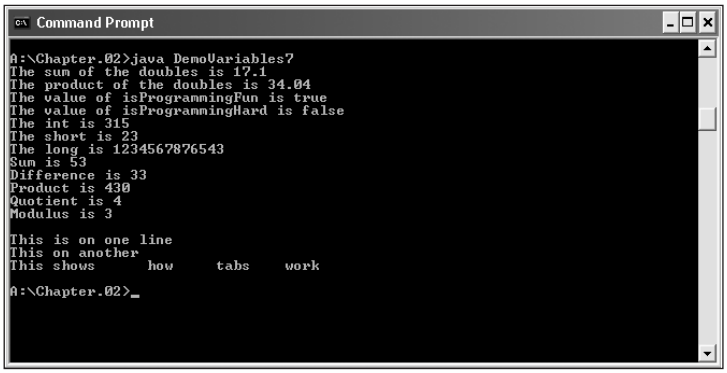

Figure 2-8 Output of the DemoVariables7 program demonstrating escape sequences

Chapter Summary

- Data is constant when it cannot be changed after a program is compiled; data is variable when it might change.
- Variables are named memory locations that your program can use to store values. You can name a variable using any legal identifier. A variable name must start with a

letter and cannot be a reserved keyword. You must declare all variables you want to use in a program. A variable declaration requires a type and a name; it can also include an assigned value.

- The Java programming language provides for eight primitive types of data: Boolean, byte, char, double, float, int, long, and short.
- You can declare multiple variables of the same type in separate statements or in a single statement, separated by commas.
- There are five standard arithmetic operators for integers: + – * / and %.
- Operator precedence is the order in which parts of a mathematical expression are evaluated. Multiplication, division, and modulus always take place prior to addition or subtraction in an expression. Right and left parentheses can be added within an expression when exceptions to this rule are required. When more than one pair of parentheses are added, the innermost expression surrounded by parentheses is evaluated first.
- A Boolean type variable can hold a `true` or `false` value.
- There are six comparison operators: > < == >= <= and !=.
- A floating-point number contains decimal positions. The Java programming language supports two floating-point data types: float and double.
- When you perform mathematical operations on unlike types, Java implicitly converts the variables to a unifying type. You can explicitly override the unifying type imposed by the Java programming language by performing a type cast.
- You use the char data type to hold any single character. You type constant character values in single quotation marks. You type String constants that store more than one character between double quotation marks. You can store some characters using an escape sequence, which always begins with a backslash.
- The characters used in Java programming are represented in the 16-bit Unicode scheme.

Review Questions

1. When data cannot be changed after a program is compiled, the data is _____.

   a. constant

   b. variable

   c. volatile

   d. mutable

2. Which of the following is not a primitive data type in the Java programming language?
   a. Boolean
   b. byte
   c. int
   d. sector
3. Which of the following elements is not required in a variable declaration?
   a. a type
   b. an identifier
   c. an assigned value
   d. a semicolon
4. The assignment operator in the Java programming language is _____.
   a. =
   b. ==
   c. :=
   d. ::
5. Which of the following values can you assign to a variable of type int?
   a. 0
   b. 98.6
   c. 'S'
   d. 5,000,000,000,000
6. Which of the following data types can store a value in the least amount of memory?
   a. short
   b. long
   c. int
   d. byte
7. The modulus operator _____.
   a. is represented by a forward slash
   b. provides the remainder of integer division
   c. provides the remainder of floating-point division
   d. Answers b. and c. are correct.
8. According to the rules of operator precedence, division always takes place prior to _____.
   a. multiplication
   b. modulus

c. subtraction

d. Answers a. and b. are correct.

9. A Boolean variable can hold _____.

a. any character

b. any whole number

c. any decimal number

d. the values `true` or `false`

10. The "equal to" comparison operator is _____.

a. =

b. ==

c. !=

d. !!

11. The value 137.68 can be held by a variable of type _____.

a. int

b. float

c. double

d. Two of the preceding answers are correct

12. When you perform arithmetic with values of diverse types, the Java programming language _____.

a. issues an error message

b. implicitly converts the values to a unifying type

c. requires you to explicitly convert the values to a unifying type

d. requires you to perform a cast

13. If you attempt to add a float, an int, and a byte, the result will be a(n) _____.

a. float

b. int

c. byte

d. error message

14. You use a _____ to explicitly override an implicit type.

a. mistake

b. type cast

c. format

d. type set

15. Which assignment is correct?
    a. `char aChar = 5;`
    b. `char aChar = "W";`
    c. `char aChar = '*';`
    d. Two of the preceding answers are correct
16. An escape sequence always begins with a(n) _____.
    a. 'e'
    b. forward slash
    c. backslash
    d. equals sign
17. The 16-bit coding scheme employed by the Java programming language is _____.
    a. Unicode
    b. ASCII
    c. EBCDIC
    d. hexadecimal

EXERCISES

1. What is the numeric value of each of the following expressions as evaluated by the Java programming language?
    a. 4 + 6 * 3
    b. 6 / 3 * 7
    c. 18 / 2 + 14 / 2
    d. 16 / 2
    e. 17 / 2
    f. 28 / 5
    g. 16 % 2
    h. 17 % 2
    i. 28 % 5
    j. 28 % 5 * 3 + 1
    k. (2 + 3) * 4
    l. 20 / (4 + 1)

2. What is the value of each of the following Boolean expressions?
   a. 4 > 1
   b. 5 <= 18
   c. 43 >= 43
   d. 2 == 3
   e. 2 + 5 == 7
   f. 3 + 8 <= 10
   g. 3 != 9
   h. 13 != 13
   i. –4 != 4
   j. 2 + 5 * 3 == 21
3. Which of the following expressions are illegal? For the legal expressions, what is the numeric value of each, as evaluated by the Java programming language?
   a. 2.3 * 1.2
   b. 5.67 – 2
   c. 25.0 / 5.0
   d. 7.0 % 3.0
   e. 8 % 2.0
4. Choose the best data type for each of the following so that no memory storage is wasted. Give an example of a typical value that would be held by the variable, and explain why you chose the type you did.
   a. your age
   b. the U.S. national debt
   c. your shoe size
   d. your middle initial
5. Use a text editor to write a Java program that declares variables to represent the length and width of a room in feet. Use Room as the class name. Assign appropriate values to the variables—for example, length = 15 and width = 25. Compute and display the floor space of the room in square feet (area = length * width). Display more than just a value as output; also display explanatory text with the value—for example, `The floor space is 375 square feet.`. Save the program as **Room.java** in the Chapter.02 folder on your Student Disk.
6. Use a text editor to write a Java program that declares variables to represent the length and width of a room in feet, and the price of carpeting per square foot in dollars and cents. Use Carpet as the class name. Assign appropriate values to the variables. Compute and display, with explanatory text, the cost of carpeting the room. Save the program as **Carpet.java** in the Chapter.02 folder on your Student Disk.

7. Write a program that declares variables to represent the length and width of a room in feet, and the price of carpeting per square yard in dollars and cents. Use Yards as the class name. Assign the value 25 to the length variable and the value 42 to the width variable. Compute and display the cost of carpeting the room. (*Hint*: There are nine square feet in one square yard.) Save the program as **Yards.java** in the Chapter.02 folder on your Student Disk.
8. Write a program that declares a minutes variable that represents minutes worked on a job, and assign a value. Use Time as the class name. Display the value in hours and minutes. For example, 197 minutes becomes 3 hours and 17 minutes. Save the program as **Time.java** in the Chapter.02 folder on your Student Disk.
9. Write a program that declares variables to hold your three initials. Display the three initials with a period following each one, as in J.M.F. Save the program as **Initials.java** in the Chapter.02 folder on your Student Disk.
10. Write a program that contains variables that hold your tuition fee and your book fee. Display the sum of the variables. Save the program as **Fees.java** in the Chapter.02 folder on your Student Disk.
11. Write a program that contains variables that hold your hourly rate of pay and number of hours that you worked. Display your gross pay, your withholding tax, which is 15 percent of your gross pay, and your net pay (gross pay – withholding). Save the program as **Payroll.java** in the Chapter.02 folder on your Student Disk.
12. a. Write a program that calculates and displays the conversion of $57 into dollar-bill form—20's, 10's, 5's, and 1's. Create a separate method to do the calculation and the display. Pass 57 as a variable to this method. Save the program as **Dollars.java** in the Chapter.02 folder on your Student Disk.
    b. In the Dollars.java program, alter the value of the variable that holds the amount of money. Run the program and confirm that the amount of each denomination calculates correctly.
13. Write a program that calculates and displays the amount of money you would have if you invested $1,000 at 5 percent interest for one year. Use the formula: Future Amount = Principal * Rate * Time. Save the program as **Interest.java** in the Chapter.02 folder on your Student Disk.
14. Write a program that illustrates the use of casting. Start with an integer, float, and double, and perform casts on the integer, float, and double. Print the results of each cast. Save the program as **Types.java** in the Chapter.02 folder on your Student Disk.
15. Write a program that displays FirstName, LastName, Address, and Phone on one line of output, and your first name, last name, address, and phone number on the second line. Make sure that your data lines up with the headings. Save the program as **Escape.java** in the Chapter.02 folder on your Student Disk.

16. Write a program to convert Fahrenheit temperature to Centigrade. Use the normal human body temperature of 98.6 degrees Fahrenheit, as the test case. Use the formula Centigrade = 5/9 (Fahrenheit -32). Save the program as **FahrenheitToCentigrade.java** in the Chapter.02 folder on your Student Disk.
17. Write a program that will output the following table of inventory items:

Item Num	Item Name	Units	On Hand
B1242	Bolt	Each	1000
N1242	Nut	Each	1200
W1242	Washer	Each	1150
N2323	Nails	Lbs	2250

Save the program as **Inventory.java** in the Chapter.02 folder on your Student Disk.

18. Each of the following files in the Chapter.02 folder on your Student Disk has syntax and/or logical errors. In each case, determine the problem and fix the program. After you correct the errors, save each file using the same filename preceded with Fix. For example, DebugTwo1.java will become FixDebugTwo1.java.
    a. DebugTwo1.java
    b. DebugTwo2.java
    c. DebugTwo3.java
    d. DebugTwo4.java

CASE PROJECT

Travel Tickets Company sells tickets for airlines, tours, and other travel-related services. Because long ticket numbers have often been entered incorrectly by agents, Travel Tickets has asked you to code a program that will indicate if a ticket number entry is invalid. The program also should prompt the agent to check and reenter the correct ticket number. Ticket numbers are 10 to 12 characters long. Ticket numbers are designed so that if you drop the last digit of the number, then divide the number by 7, the remainder of the division will be identical to the last dropped digit. This process is illustrated in the following example:

Step 1	Ticket number	12344321566
Step 2	Remove last digit, leaving	1234432156
Step 3	Divide remaining number by 7.	1234432156 divided by 7
Step 4	Remainder of 6 matches the digit dropped from the ticket.	
Step 5	Because the last digit matches the remainder of the division, the ticket number is valid.	

Although you cannot write the entire application, write a program that allows the declaration of a variable to hold the ticket number 1234432156 (first 10 digits of the 11-digit ticket number 12344321566), remove the last digit, divide the number by 7, and print the result of the division, which should be the digit 6 in this example for the ticket to be valid. Declare a second variable to hold the ticket number 2323454567 (first 10 digits of the 11-digit ticket number 23234545678), divide the number by 7, and print the result of the division. Is either of the entered ticket numbers valid by Travel Tickets rules? Save the case as **TicketNumber.java** in the Chapter.02 folder on your Student Disk.