Chapter 2 Introduction to Java Applications

Java™ How to Program, 9/e

OBJECTIVES

In this chapter you'll learn:

- To write simple Java applications.
- To use input and output statements.
- Java's primitive types.
- Basic memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write decision-making statements.
- To use relational and equality operators.

2.1 Introduction
2.2 Your First Program in Java: Printing a Line of Text
2.3 Modifying Your First Java Program
2.4 Displaying Text with `printf`
2.5 Another Application: Adding Integers
2.6 Memory Concepts
2.7 Arithmetic
2.8 Decision Making: Equality and Relational Operators
2.9 Wrap-Up

2.1 Introduction

- Java application programming
- Use tools from the JDK to compile and run programs.
- Videos at `www.deitel.com/books/jhtp9/`
  - Help you get started with Eclipse and NetBeans integrated development environments.

2.2 Our First Program in Java: Printing a Line of Text

- Java application
  - A computer program that executes when you use the java command to launch the Java Virtual Machine (JVM).
- Sample program in Fig. 2.1 displays a line of text.

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.println( "Welcome to Java Programming!" );
10    } // end method main
11 } // end class Welcome1
```

```
Welcome to Java Programming!
```

Fig. 2.1 | Text-printing program.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Comments

```
// Fig. 2.1: Welcome1.java
```

  - // indicates that the line is a comment.
  - Used to document programs and improve their readability.
  - Compiler ignores comments.
  - A comment that begins with // is an end-of-line comment—it terminates at the end of the line on which it appears.

- Traditional comment, can be spread over several lines as in

```
/* This is a traditional comment. It
   can be split over multiple lines */
```

  - This type of comment begins with /* and ends with */.
  - All text between the delimiters is ignored by the compiler.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Javadoc comments
  - Delimited by /** and */.
  - All text between the Javadoc comment delimiters is ignored by the compiler.
  - Enable you to embed program documentation directly in your programs.
  - The javadoc utility program (Appendix M) reads Javadoc comments and uses them to prepare your program's documentation in HTML format.

Common Programming Error 2.1

Forgetting one of the delimiters of a traditional or Java-doc comment is a syntax error. A ***syntax error*** *occurs when the compiler encounters code that violates Java's language rules (i.e., its syntax). These rules are similar to a natural language's grammar rules specifying sentence structure. Syntax errors are also called* ***compiler errors****,* ***compile-time errors*** *or* ***compilation errors****, because the compiler detects them during the compilation phase. The compiler responds by issuing an error message and preventing your program from compiling.*

Good Programming Practice 2.1

Some organizations require that every program begin with a comment that states the purpose of the program and the author, date and time when the program was last modified.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Blank lines and space characters
  - Make programs easier to read.
  - Blank lines, spaces and tabs are known as white space (or whitespace).
  - White space is ignored by the compiler.

Good Programming Practice 2.2

Use blank lines and spaces to enhance program readability.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Class declaration

```
public class Welcome1
```

  - Every Java program consists of at least one class that you define.
  - **class** keyword introduces a class declaration and is immediately followed by the class name.
  - Keywords (Appendix C) are reserved for use by Java and are always spelled with all lowercase letters.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Class names
  - By convention, begin with a capital letter and capitalize the first letter of each word they include (e.g., `SampleClassName`).
  - A class name is an identifier—a series of characters consisting of letters, digits, underscores (`_`) and dollar signs (`$`) that does not begin with a digit and does not contain spaces.
  - Java is case sensitive—uppercase and lowercase letters are distinct—so `a1` and `A1` are different (but both valid) identifiers.

Common Programming Error 2.2

A `public` *class must be placed in a file that has the same name as the class (in terms of both spelling and capitalization) plus the* `.java` *extension; otherwise, a compilation error occurs. For example,* `public` *class* `Welcome` *must be placed in a file named* `Welcome.java`.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Braces
  - A left brace, {, begins the body of every class declaration.
  - A corresponding right brace, }, must end each class declaration.
  - Code between braces should be indented.
  - This indentation is one of the spacing conventions mentioned earlier.

Error-Prevention Tip 2.1

When you type an opening left brace, {, immediately type the closing right brace, }, then reposition the cursor between the braces and indent to begin typing the body. This practice helps prevent errors due to missing braces. Many IDEs insert the braces for you.

Common Programming Error 2.3

It's a syntax error if braces do not occur in matching pairs.

Good Programming Practice 2.3

Indent the entire body of each class declaration one "level" between the left brace and the right brace that delimit the body of the class. We recommend using three spaces to form a level of indent. This format emphasizes the class declaration's structure and makes it easier to read.

Good Programming Practice 2.4

Many IDEs insert indentation for you in all the right places. The Tab *key may also be used to indent code, but tab stops vary among text editors. Most IDEs allow you to configure tabs such that a specified number of spaces is inserted each time you press the* Tab *key.*

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Declaring the `main` Method

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

  - Starting point of every Java application.
  - Parentheses after the identifier `main` indicate that it's a program building block called a method.
  - Java class declarations normally contain one or more methods.
  - `main` must be defined as shown; otherwise, the JVM will not execute the application.
  - Methods perform tasks and can return information when they complete their tasks.
  - Keyword **`void`** indicates that this method will not return any information.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Body of the method declaration
  - Enclosed in left and right braces.
- Statement

```
System.out.println("Welcome to Java Programming!");
```

  - Instructs the computer to perform an action
    - Print the string of characters contained between the double quotation marks.
  - A string is sometimes called a character string or a string literal.
  - White-space characters in strings are not ignored by the compiler.
  - Strings cannot span multiple lines of code.

Good Programming Practice 2.5

Indent the entire body of each method declaration one "level" between the braces that define the body of the method. This makes the structure of the method stand out and makes the method declaration easier to read.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- **System.out** object
  - Standard output object.
  - Allows Java applications to display strings in the command window from which the Java application executes.
- **System.out.println** method
  - Displays (or prints) a line of text in the command window.
  - The string in the parentheses the argument to the method.
  - Positions the output cursor at the beginning of the next line in the command window.
- Most statements end with a semicolon.

Error-Prevention Tip 2.2

When learning how to program, sometimes it's helpful to "break" a working program so you can familiarize yourself with the compiler's syntax-error messages. These messages do not always state the exact problem in the code. When you encounter an error message, it will give you an idea of what caused the error. [Try removing a semicolon or brace from the program of Fig. 2.1, then recompile the program to see the error messages generated by the omission.]

Error-Prevention Tip 2.3

When the compiler reports a syntax error, it may not be on the line that the error message indicates. First, check the line for which the error was reported. If you don't find an error on that line,, check several preceding lines.

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- Compiling and Executing Your First Java Application
  - Open a command window and change to the directory where the program is stored.
  - Many operating systems use the command `cd` to change directories.
  - To compile the program, type

```
javac Welcome1.java
```

  - If the program contains no syntax errors, preceding command creates a `.class` file (known as the class file) containing the platform-independent Java bytecodes that represent the application.
  - When we use the `java` command to execute the application on a given platform, these bytecodes will be translated by the JVM into instructions that are understood by the underlying operating system.

Error-Prevention Tip 2.4

When attempting to compile a program, if you receive a message such as "`bad command or filename`*," "*`javac: command not found`*" or "*`'javac' is not recognized as an internal or external command, operable program or batch file`*," then your Java software installation was not completed properly. If you're using the JDK, this indicates that the system's* PATH *environment variable was not set properly. Please carefully review the installation instructions in the Before You Begin section of this book. On some systems, after correcting the* PATH*, you may need to reboot your computer or open a new command window for these settings to take effect.*

Error-Prevention Tip 2.5

Each syntax-error message contains the file name and line number where the error occurred. For example, `Welcome1.java:6` *indicates that an error occurred at line 6 in* `Welcome1.java`*. The rest of the message provides information about the syntax error.*

Error-Prevention Tip 2.6

The compiler error message "`class Welcome1 is public, should be declared in a file named Welcome1.java`*" indicates that the file name does not match the name of the* `public` *class in the file or that you typed the class name incorrectly when compiling the class.*

2.2 Our First Program in Java: Printing a Line of Text (Cont.)

- To execute the program, type `java Welcome1`.
- Launches the JVM, which loads the `.class` file for class `Welcome1`.
- Note that the `.class` file-name extension is omitted from the preceding command; otherwise, the JVM will not execute the program.
- The JVM calls method `main` to execute the program.

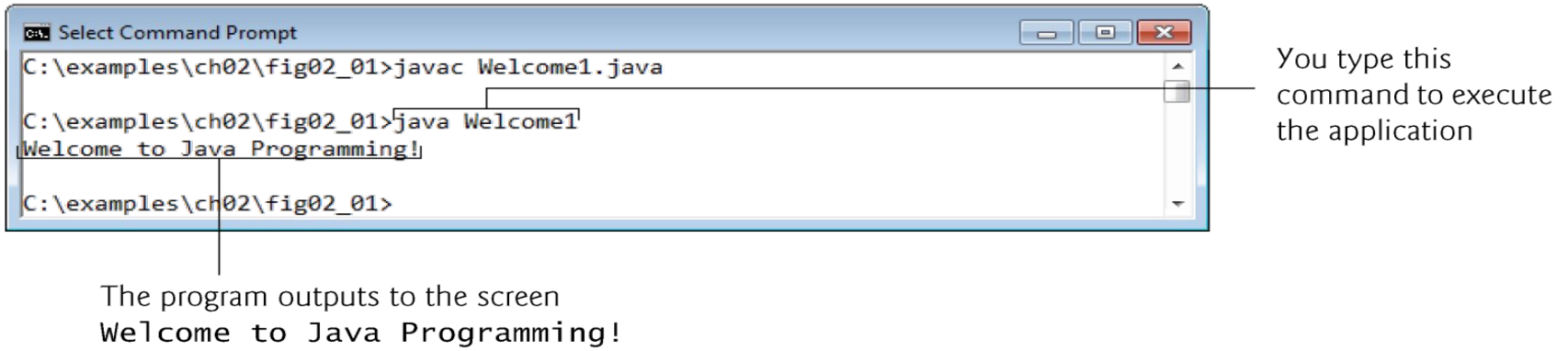

Fig. 2.2 | Executing `Welcome1` from the **Command Prompt**.

Error-Prevention Tip 2.7

When attempting to run a Java program, if you receive a message such as "`Exception in thread "main" java.lang.NoClassDefFoundError: Welcome1`*," your* `CLASSPATH` *environment variable has not been set properly. Please carefully review the installation instructions in the Before You Begin section of this book. On some systems, you may need to reboot your computer or open a new command window after configuring the* `CLASSPATH`*.*

2.3 Modifying Your First Java Program

- Class `Welcome2`, shown in Fig. 2.3, uses two statements to produce the same output as that shown in Fig. 2.1.
- New and key features in each code listing are highlighted.
- `System.out`'s method `print` displays a string.
- Unlike `println`, `print` does not position the output cursor at the beginning of the next line in the command window.
  - The next character the program displays will appear immediately after the last character that `print` displays.

```
1  // Fig. 2.3: Welcome2.java
2  // Printing a line of text with multiple statements.
3
4  public class Welcome2
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.print( "Welcome to " );
10       System.out.println( "Java Programming!" );
11    } // end method main
12 } // end class Welcome2
```

Prints `Welcome to` and leaves cursor on same line

Prints `Java Programming!` starting where the cursor was positioned previously, then outputs a newline character

```
Welcome to Java Programming!
```

Fig. 2.3 | Printing a line of text with multiple statements.

2.3 Modifying Your First Java Program (Cont.)

- Newline characters indicate to `System.out`'s `print` and `println` methods when to position the output cursor at the beginning of the next line in the command window.
- Newline characters are white-space characters.
- The backslash (\) is called an escape character.
  - Indicates a "special character"
- Backslash is combined with the next character to form an escape sequence.
- The escape sequence `\n` represents the newline character.
- Complete list of escape sequences

```
java.sun.com/docs/books/jls/third_edition/html/
  lexical.html#3.10.6.
```

```
1  // Fig. 2.4: Welcome3.java
2  // Printing multiple lines of text with a single statement.
3
4  public class Welcome3
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.println( "Welcome\nto\nJava\nProgramming!" );
10    } // end method main
11 } // end class Welcome3
```

Each \n moves the output cursor to the next line, where output continues

```
Welcome
to
Java
Programming!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do *not* advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`.

Fig. 2.5 | Some common escape sequences.

2.4 Displaying Text with `printf`

- `System.out.printf` method
  - `f` means "formatted"
  - displays formatted data
- Multiple method arguments are placed in a comma-separated list.
- Java allows large statements to be split over many lines.
  - Cannot split a statement in the middle of an identifier or string.
- Method `printf`'s first argument is a format string
  - May consist of fixed text and format specifiers.
  - Fixed text is output as it would be by `print` or `println`.
  - Each format specifier is a placeholder for a value and specifies the type of data to output.
- Format specifiers begin with a percent sign (%) and are followed by a character that represents the data type.
- Format specifier `%s` is a placeholder for a string.

Good Programming Practice 2.6

Place a space after each comma (,) in an argument list to make programs more readable.

```
1  // Fig. 2.6: Welcome4.java
2  // Displaying multiple lines with method System.out.printf.
3
4  public class Welcome4
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.printf( "%s\n%s\n",
10          "Welcome to", "Java Programming!" );
11    } // end method main
12 } // end class Welcome4
```

Each %s is a placeholder for a String that comes later in the argument list

Statements can be split over multiple lines.

```
Welcome to
Java Programming!
```

Fig. 2.6 | Displaying multiple lines with method `System.out.printf`.

Common Programming Error 2.4

Splitting a statement in the middle of an identifier or a string is a syntax error.

2.5 Another Application: Adding Integers

- Integers
  - Whole numbers, like –22, 7, 0 and 1024)
- Programs remember numbers and other data in the computer's memory and access that data through program elements called variables.
- The program of Fig. 2.7 demonstrates these concepts.

```
1  // Fig. 2.7: Addition.java
2  // Addition program that displays the sum of two numbers.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class Addition
6  {
7     // main method begins execution of Java application
8     public static void main( String[] args )
9     {
10       // create a Scanner to obtain input from the command window
11       Scanner input = new Scanner( System.in );
12
13       int number1; // first number to add
14       int number2; // second number to add
15       int sum; // sum of number1 and number2
16
17       System.out.print( "Enter first integer: " ); // prompt
18       number1 = input.nextInt(); // read first number from user
19
20       System.out.print( "Enter second integer: " ); // prompt
21       number2 = input.nextInt(); // read second number from user
22
23       sum = number1 + number2; // add numbers, then store total in sum
```

Imports class `Scanner` for use in this program

Creates `Scanner` for reading data from the user

Variables that are declared but not initialized

Reads an `int` value from the user

Reads another `int` value from the user

Sums the values of `number1` and `number2`

Fig. 2.7 | Addition program that displays the sum of two numbers. (Part 1 of 2.)

```
24
25          System.out.printf( "Sum is %d\n", sum ); // display sum
26       } // end method main
27    } // end class Addition
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Fig. 2.7 | Addition program that displays the sum of two numbers. (Part 2 of 2.)

2.5 Another Application: Adding Integers (Cont.)

- **import** declaration
  - Helps the compiler locate a class that is used in this program.
  - Rich set of predefined classes that you can reuse rather than "reinventing the wheel."
  - Classes are grouped into packages—named groups of related classes—and are collectively referred to as the Java class library, or the Java Application Programming Interface (Java API).
  - You use `import` declarations to identify the predefined classes used in a Java program.

Common Programming Error 2.5

All `import` *declarations must appear before the first class declaration in the file. Placing an* `import` *declaration inside or after a class declaration is a syntax error.*

Error-Prevention Tip 2.8

Forgetting to include an `import` *declaration for a class used in your program typically results in a compilation error containing a message such as "*`cannot find symbol`*." When this occurs, check that you provided the proper* `import` *declarations and that the names in them are correct, including proper capitalization.*

2.5 Another Application: Adding Integers (Cont.)

- Variable declaration statement

```
Scanner input = new Scanner( System.in );
```

  - Specifies the name (`input`) and type (`Scanner`) of a variable that is used in this program.

- Variable
  - A location in the computer's memory where a value can be stored for use later in a program.
  - Must be declared with a name and a type before they can be used.
  - A variable's name enables the program to access the value of the variable in memory.
  - The name can be any valid identifier.
  - A variable's type specifies what kind of information is stored at that location in memory.

2.5 Another Application: Adding Integers (Cont.)

- **Scanner**
  - Enables a program to read data for use in a program.
  - Data can come from many sources, such as the user at the keyboard or a file on disk.
  - Before using a `Scanner`, you must create it and specify the source of the data.
- The equals sign (=) in a declaration indicates that the variable should be initialized (i.e., prepared for use in the program) with the result of the expression to the right of the equals sign.
- The `new` keyword creates an object.
- Standard input object, `System.in`, enables applications to read bytes of information typed by the user.
- `Scanner` object translates these bytes into types that can be used in a program.

2.5 Another Application: Adding Integers (Cont.)

- Variable declaration statements

```
int number1; // first number to add
int number2; // second number to add
int sum; // sum of number1 and number2
```

  declare that variables `number1`, `number2` and `sum` hold data of type `int`

  - They can hold integer.
  - Range of values for an `int` is –2,147,483,648 to +2,147,483,647.
  - Actual `int` values may not contain commas.

- Several variables of the same type may be declared in one declaration with the variable names separated by commas.

Good Programming Practice 2.7

Declare each variable on a separate line. This format allows a descriptive comment to be inserted next to each declaration.

Good Programming Practice 2.8

Choosing meaningful variable names helps a program to be self-documenting (i.e., one can understand the program simply by reading it rather than by reading manuals or viewing an excessive number of comments).

Good Programming Practice 2.9

By convention, variable-name identifiers begin with a lowercase letter, and every word in the name after the first word begins with a capital letter. For example, variable-name identifier `firstNumber` *starts its second word,* `Number`*, with a capital* `N`*.*

2.5 Another Application: Adding Integers (Cont.)

- Prompt
  - Output statement that directs the user to take a specific action.
- `System` is a class.
  - Part of package **java.lang**.
  - Class `System` is not imported with an `import` declaration at the beginning of the program.

Software Engineering Observation 2.1

By default, package `java.lang` *is imported in every Java program; thus, classes in* `java.lang` *are the only ones in the Java API that do not require an* `import` *declaration.*

2.5 Another Application: Adding Integers (Cont.)

- `Scanner` method `nextInt`

```
number1 = input.nextInt(); // read first number from user
```

  - Obtains an integer from the user at the keyboard.
  - Program waits for the user to type the number and press the Enter key to submit the number to the program.
- The result of the call to method `nextInt` is placed in variable `number1` by using the assignment operator, `=`.
  - "`number1` gets the value of `input.nextInt()`."
  - Operator `=` is called a binary operator—it has two operands.
  - Everything to the right of the assignment operator, `=`, is always evaluated before the assignment is performed.

Good Programming Practice 2.10

Placing spaces on either side of a binary operator makes the program more readable.

2.5 Another Application: Adding Integers (Cont.)

- Arithmetic

```
sum = number1 + number2; // add numbers
```

  - Assignment statement that calculates the sum of the variables `number1` and `number2` then assigns the result to variable `sum` by using the assignment operator, `=`.
  - "`sum` gets the value of `number1 + number2`."
  - In general, calculations are performed in assignment statements.
  - Portions of statements that contain calculations are called expressions.
  - An expression is any portion of a statement that has a value associated with it.

2.5 Another Application: Adding Integers (Cont.)

- Integer formatted output

```
System.out.printf( "Sum is %d\n", sum );
```

  - Format specifier %d is a placeholder for an `int` value
  - The letter d stands for "decimal integer."

2.6 Memory Concepts

- Variables
  - Every variable has a name, a type, a size (in bytes) and a value.
  - When a new value is placed into a variable, the new value replaces the previous value (if any)
  - The previous value is lost.

Fig. 2.8 | Memory location showing the name and value of variable `number1`.

Fig. 2.9 | Memory locations after storing values for `number1` and `number2`.

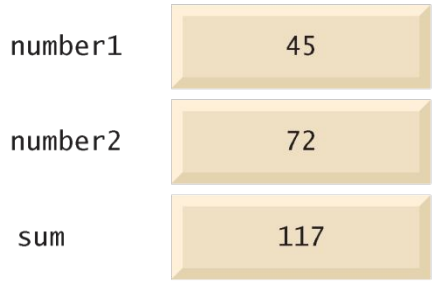

Fig. 2.10 | Memory locations after storing the sum of `number1` and `number2`.

2.7 Arithmetic

- Arithmetic operators are summarized in Fig. 2.11.
- The asterisk (*) indicates multiplication
- The percent sign (%) is the remainder operator
- The arithmetic operators are binary operators because they each operate on two operands.
- Integer division yields an integer quotient.
  - Any fractional part in integer division is simply discarded (i.e., truncated)—no rounding occurs.
- The remainder operator, %, yields the remainder after division.

Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.11 | Arithmetic operators.

2.7 Arithmetic (Cont.)

- Arithmetic expressions in Java must be written in straight-line form to facilitate entering programs into the computer.
- Expressions such as "`a` divided by `b`" must be written as `a / b`, so that all constants, variables and operators appear in a straight line.
- Parentheses are used to group terms in expressions in the same manner as in algebraic expressions.
- If an expression contains nested parentheses, the expression in the innermost set of parentheses is evaluated first.

2.7 Arithmetic (Cont.)

- Rules of operator precedence
  - Multiplication, division and remainder operations are applied first.
  - If an expression contains several such operations, they are applied from left to right.
  - Multiplication, division and remainder operators have the same level of precedence.
  - Addition and subtraction operations are applied next.
  - If an expression contains several such operations, the operators are applied from left to right.
  - Addition and subtraction operators have the same level of precedence.
- When we say that operators are applied from left to right, we are referring to their associativity.
- Some operators associate from right to left.
- Complete precedence chart is included in Appendix A.

Operator(s)	Operation(s)	Order of evaluation (precedence)
*	Multiplication	Evaluated first. If there are several operators of this type, they're evaluated from left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated next. If there are several operators of this type, they're evaluated from left to right.
-	Subtraction	
=	Assignment	Evaluated last.

Fig. 2.12 | Precedence of arithmetic operators.

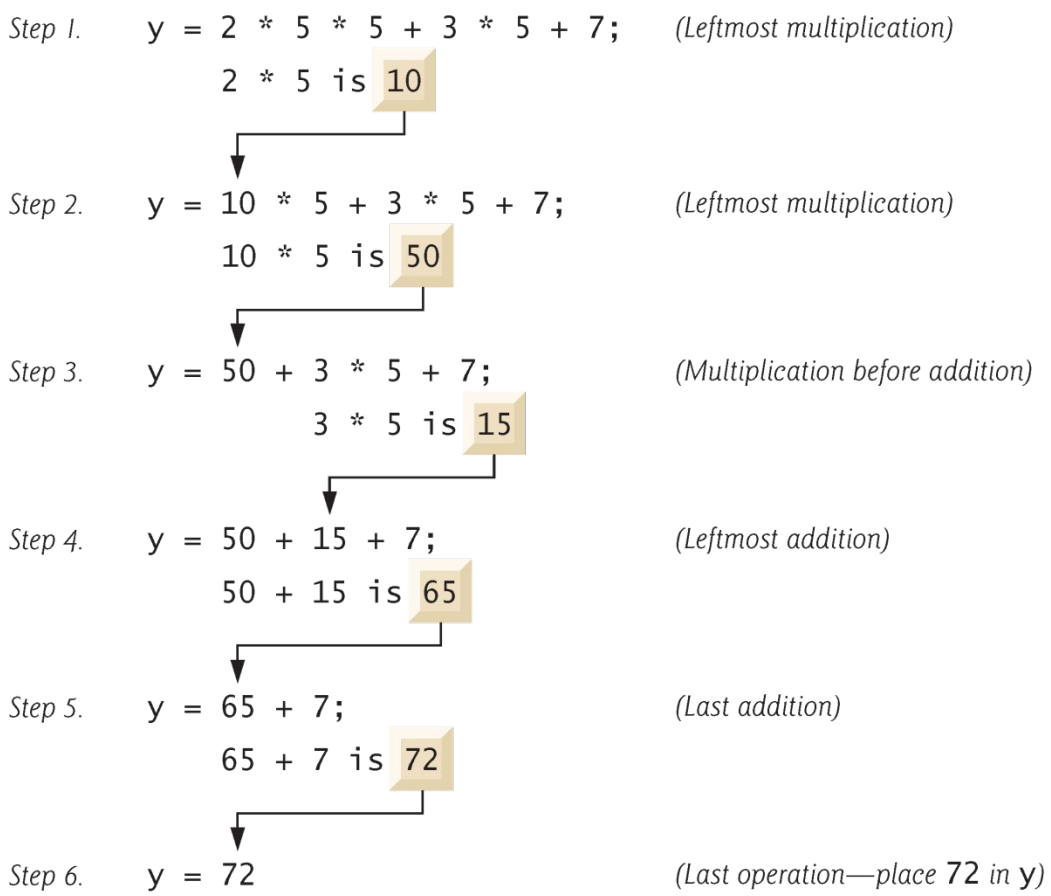

Fig. 2.13 | Order in which a second-degree polynomial is evaluated.

2.7 Arithmetic (Cont.)

- As in algebra, it's acceptable to place redundant parentheses (unnecessary parentheses) in an ex-pression to make the expression clearer.

2.8 Decision Making: Equality and Relational Operators

- Condition
  - An expression that can be true or false.
- if selection statement
  - Allows a program to make a decision based on a condition's value.
- Equality operators (== and !=)
- Relational operators (>, <, >= and <=)
- Both equality operators have the same level of precedence, which is lower than that of the relational operators.
- The equality operators associate from left to right.
- The relational operators all have the same level of precedence and also associate from left to right.

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.14 | Equality and relational operators.

```
1  // Fig. 2.15: Comparison.java
2  // Compare integers using if statements, relational operators
3  // and equality operators.
4  import java.util.Scanner; // program uses class Scanner
5
6  public class Comparison
7  {
8     // main method begins execution of Java application
9     public static void main( String[] args )
10    {
11       // create Scanner to obtain input from command line
12       Scanner input = new Scanner( System.in );
13
14       int number1; // first number to compare
15       int number2; // second number to compare
16
17       System.out.print( "Enter first integer: " ); // prompt
18       number1 = input.nextInt(); // read first number from user
19
20       System.out.print( "Enter second integer: " ); // prompt
21       number2 = input.nextInt(); // read second number from user
22
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 1 of 3.)

```
23          if ( number1 == number2 )
24             System.out.printf( "%d == %d\n", number1, number2 );
25
26          if ( number1 != number2 )
27             System.out.printf( "%d != %d\n", number1, number2 );
28
29          if ( number1 < number2 )
30             System.out.printf( "%d < %d\n", number1, number2 );
31
32          if ( number1 > number2 )
33             System.out.printf( "%d > %d\n", number1, number2 );
34
35          if ( number1 <= number2 )
36             System.out.printf( "%d <= %d\n", number1, number2 );
37
38          if ( number1 >= number2 )
39             System.out.printf( "%d >= %d\n", number1, number2 );
40       } // end method main
41    } // end class Comparison
```

Output statement executes only if the numbers are equal

Output statement executes only if the numbers are not equal

Output statement executes only if `number1` is less than `number2`

Output statement executes only if `number1` is greater than `number2`

Output statement executes only if `number1` is less than or equal to `number2`

Output statement executes only if `number1` is greater than or equal to `number2`

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 2 of 3.)

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 3 of 3.)

2.8 Decision Making: Equality and Relational Operators (Cont.)

- An `if` statement always begins with keyword `if`, followed by a condition in parentheses.
  - Expects one statement in its body, but may contain multiple statements if they are enclosed in a set of braces (`{}`).
  - The indentation of the body statement is not required, but it improves the program's readability by emphasizing that statements are part of the body.
- Note that there is no semicolon (`;`) at the end of the first line of each `if` statement.
  - Such a semicolon would result in a logic error at execution time.
  - Treated as the empty statement—semicolon by itself.

Common Programming Error 2.6

Confusing the equality operator, ==, with the assignment operator, =, can cause a logic error or a syntax error. The equality operator should be read as "is equal to" and the assignment operator as "gets" or "gets the value of." To avoid confusion, some people read the equality operator as "double equals" or "equals equals."

Good Programming Practice 2.11

Placing only one statement per line in a program enhances program readability.

Common Programming Error 2.7

Placing a semicolon immediately after the right parenthesis of the condition in an `if` *statement is normally a logic error.*

Error-Prevention Tip 2.9

A lengthy statement can be spread over several lines. If a single statement must be split across lines, choose breaking points that make sense, such as after a comma in a comma-separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines until the end of the statement.

Good Programming Practice 2.12

When writing expressions containing many operators, refer to the operator precedence chart (Appendix A) . Confirm that the operations in the expression are performed in the order you expect. If, in a complex expression, you're uncertain about the order of evaluation, use parentheses to force the order, exactly as you'd do in algebraic expressions.

Operators				Associativity	Type
*	/	%		left to right	multiplicative
+	-			left to right	additive
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

Fig. 2.16 | Precedence and associativity of operators discussed.