Chapter 2
Introduction to Java Applications; Input/Output and Operators

Java How to Program, 11/e, Global Edition

Questions? E-mail paul.deitel@deitel.com

OBJECTIVES

In this chapter you'll:

- Write simple Java applications.
- Use input and output statements.
- Learn about Java's primitive types.
- Understand basic memory concepts.
- Use arithmetic operators.
- Learn the precedence of arithmetic operators.
- Write decision-making statements.
- Use relational and equality operators.

OUTLINE

OUTLINE (cont.)

2.5 Another Application: Adding Integers

- 2.5.1 `import` Declarations
- 2.5.2 Declaring and Creating a `Scanner` to Obtain User Input from the Keyboard
- 2.5.3 Prompting the User for Input
- 2.5.4 Declaring a Variable to Store an Integer and Obtaining an Integer from the Keyboard
- 2.5.5 Obtaining a Second Integer
- 2.5.6 Using Variables in a Calculation
- 2.5.7 Displaying the Calculation Result
- 2.5.8 Java API Documentation
- 2.5.9 Declaring and Initializing Variables in Separate Statements

OUTLINE (cont.)

2.1 Introduction

- Java application programming
- Use tools from the JDK to compile and run programs.

2.2 Your 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     // main method begins execution of Java application
6     public static void main(String[] args) {
7        System.out.println("Welcome to Java Programming!");
8     } // end method main
9  } // end class Welcome1
```

```
Welcome to Java Programming!
```

Fig. 2.1 | Text-printing program.

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

Commenting Your Programs

- 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 (online Appendix G) reads Javadoc comments and uses them to prepare program documentation in HTML5 format.

Common Programming Error 2.1

Forgetting one of the delimiters of a traditional or Javadoc 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 natural-language grammar rules specifying sentence structure, such as those in English, French, Spanish, etc. Syntax errors are also called **compiler errors**, **compile-time errors** or **compilation errors**, because the compiler detects them when compiling the program. When a syntax error is encountered, the compiler issues an error message. You must eliminate all compilation errors before your program will compile properly.

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

Using Blank Lines

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

Good Programming Practice 2.2

Use white space to enhance program readability.

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

Declaring a class

- 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 Your First Program in Java: Printing a Line of Text (Cont.)

Filename for a `public` Class

- A `public` class must be placed in a file that has a filename of the form *ClassName*`.java`, so class `Welcome1` is stored in the file `Welcome1.java`.

Common Programming Error 2.2

A compilation error occurs if a `public` class's file-name is not exactly the same name as the class (in terms of both spelling and capitalization) followed by the `.java` extension.

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

Class Names and Identifiers

- 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.

Underscore (_) in Java 9

- As of Java 9, you can no longer use an underscore (_) by itself as an identifier.

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

Class Body

- A left brace, {, begins the body of every class declaration.
- A corresponding right brace, }, must end each class declaration.

Good Programming Practice 2.3

By convention, every word in a class-name identifier begins with an uppercase letter. For example, the class-name identifier `DollarAmount` starts its first word, `Dollar`, with an uppercase `D` and its second word, `Amount`, with an uppercase `A`. This naming convention is known as **camel case**, because the uppercase letters stand out like a camel's humps.

Good Programming Practice 2.4

Indent the entire body of each class declaration one "level" between the braces that delimit the class's. This format emphasizes the class declaration's structure and makes it easier to read. We use three spaces to form a level of indent—many programmers prefer two or four spaces. Whatever you choose, use it consistently.

Good Programming Practice 2.5

IDEs typically indent code for you. The Tab key may also be used to indent code. You can configure each IDE to specify the number of spaces inserted when you press Tab.

Common Programming Error 2.3

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

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 do this for you.

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

Declaring a 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.

Good Programming Practice 2.6

Indent the entire body of each method declaration one "level" between the braces that define the method's body. This emphasizes the method's structure and makes it easier to read.

2.2 Your 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
    - Display the characters contained between the double quotation marks.
  - Together, the quotation marks and the characters between them are a string—also known as 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.

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

- `System.out` object
  - Standard output object.
  - Allows a Java application to display information in the command window from which it 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.

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

Compiling 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 compilation 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.

Common Programming Error 2.4

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

Error-Prevention Tip 2.2

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

Executing the `Welcome1` Application

- To execute this program in a command window, change to the directory containing `Welcome1.java`
  - `C:\examples\ch02\fig02_01` on Microsoft Windows or
  - `~/Documents/examples/ch02/fig02_01` on Linux/macOS.
- Next, type `java Welcome1`.
- This launches the JVM, which loads the `Welcome1.class` file.
- The command *omits* the `.class` file-name extension; otherwise, the JVM will *not* execute the program.
- The JVM calls class `Welcome1`'s `main` method.

Error-Prevention Tip 2.3

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`.

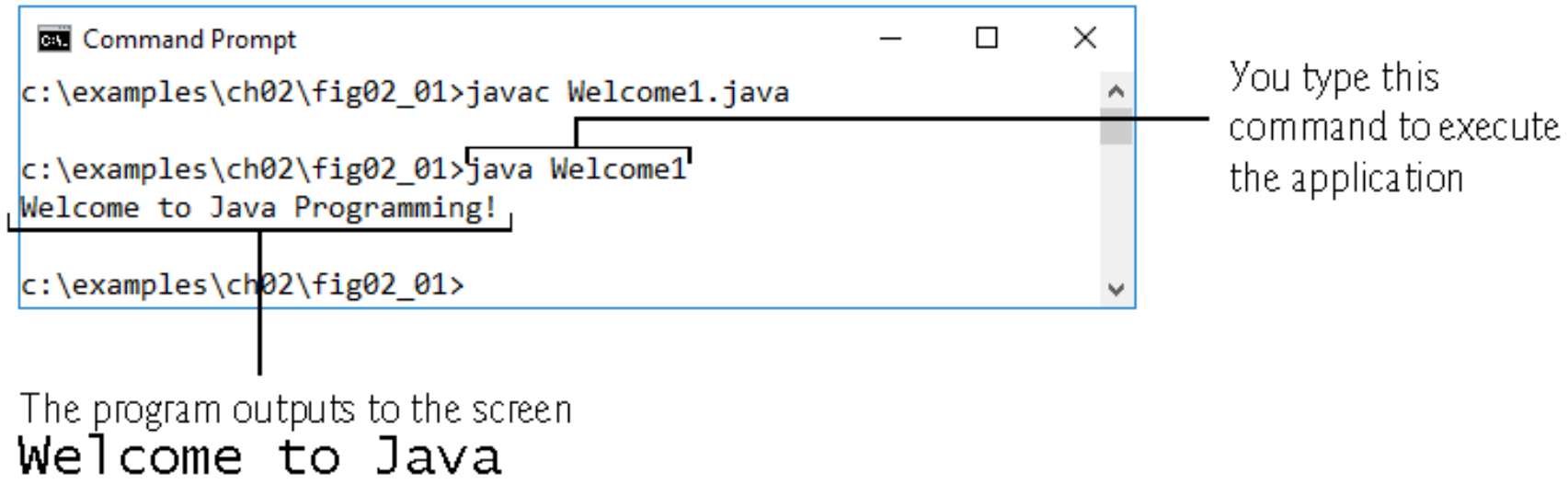

Fig. 2.2 | Executing `Welcome1` from the **Command Prompt**.

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     // main method begins execution of Java application
6     public static void main(String[] args) {
7        System.out.print("Welcome to ");
8        System.out.println("Java Programming!");
9     } // end method main
10 } // end class Welcome2
```

```
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 whitespace characters.
- The backslash (\) is called an escape character.
  - Indicates a "special character"
- Backslash is combined with the next character to form an escape sequence—`\n` represents the newline character.
- Complete list of escape sequences

  `http://docs.oracle.com/javase/specs/jls/se7/html/jls-3.html#jls-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     // main method begins execution of Java application
6     public static void main(String[] args) {
7        System.out.println("Welcome\nto\nJava\nProgramming!");
8     } // end method main
9  } // end class Welcome3
```

```
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.
- Calling a method is also referred to as invoking a method.
- 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.

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

```
Welcome to
Java Programming!
```

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

Good Programming Practice 2.7

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

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

Fig. 2.7 | Addition program that inputs two numbers then, displays their sum. (Part 1 of 2.)

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

Fig. 2.7 | Addition program that inputs two numbers then, displays their sum. (Part 2 of 2.)

2.5 Another Application: Adding Integers (cont.)

- import 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.

Common Programming Error 2.6

Forgetting to include an `import` declaration for a class that must be imported 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 data typed by the user.
- `Scanner` object translates these bytes into types that can be used in a program.

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 associated documentation or creating and viewing an excessive number of comments).

Good Programming Practice 2.9

By convention, variable-name identifiers use the camel-case naming convention with a lowercase first letter—for example, `firstNumber`.

2.5.5 Prompting the User for Input

- Prompt
  - Output statement that directs the user to take a specific action.
- Class `System`
  - 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.)

- Variable declaration statement

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

  declares that variables `number1` holds data of type `int`
  - Range of values for an `int` is –2,147,483,648 to +2,147,483,647.
  - The `int` values you use in a program may not contain commas.
- For readability, you can place underscores in numbers
  - 60_000_000 represents the `int` value 60,000,000

2.5 Another Application: Adding Integers (cont.)

- `Scanner` method `nextInt`
  - 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`
  - The = indicates that `int` variable `number1` should be initialized in its declaration with the result of `input.nextInt()`

Good Programming Practice 2.10

Place spaces on either side of a binary operator for readability.

2.5 Another Application: Adding Integers (Cont.)

- Arithmetic

```
int sum = number1 + number2; // add numbers then store total in sum
```

  - Calculates the sum of the variables `number1` and `number2` then assigns the result to variable `sum`.
  - Addition operator is a binary operator, because it has two operands—`number1` and `number2`
  - 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, so this process is said to be *destructive*.

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

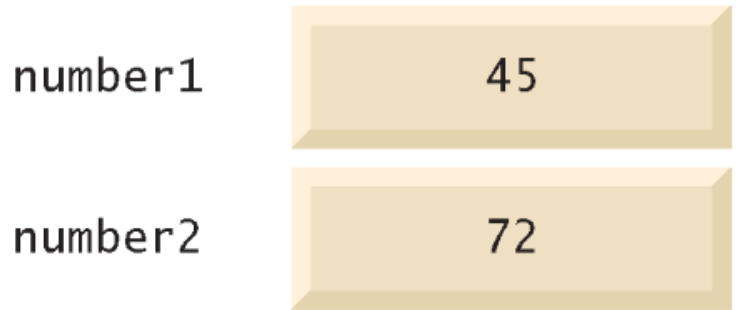

Fig. 2.9 | Memory locations after storing values for `number1` and `number2`.

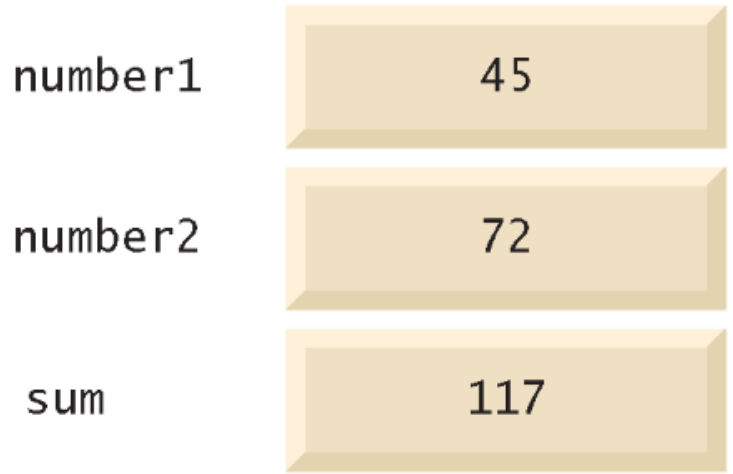

Fig. 2.10 | Memory locations after storing the sum of `number1` and `number2`.

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 *truncated* (i.e., *discarded*)—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.

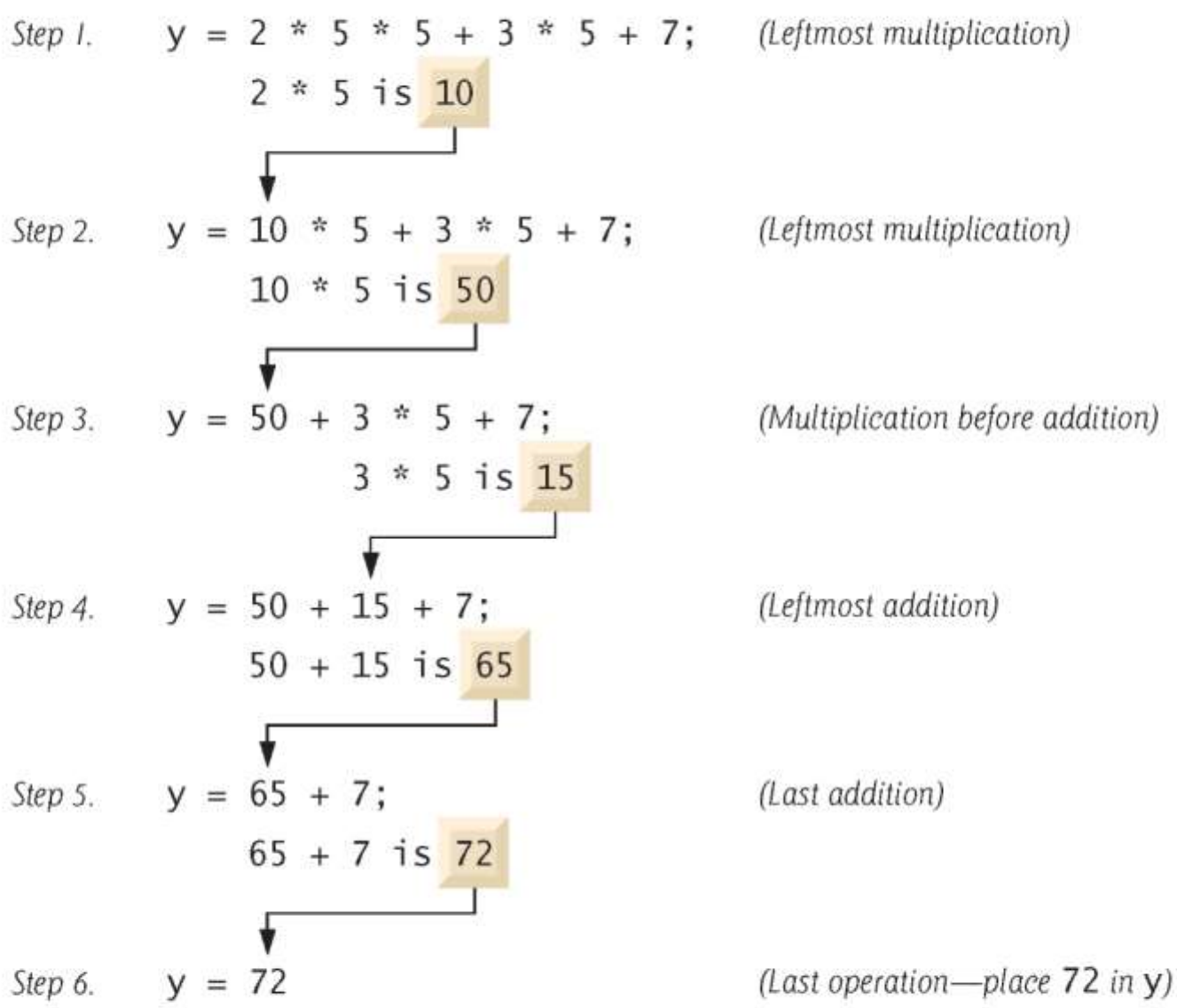

Fig. 2.13 | Order in which a second-degree polynomial is evaluated.

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*.

Algebraic 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     // main method begins execution of Java application
8     public static void main(String[] args) {
9        // create Scanner to obtain input from command line
10       Scanner input = new Scanner(System.in);
11
12       System.out.print("Enter first integer: "); // prompt
13       int number1 = input.nextInt(); // read first number from user
14
15       System.out.print("Enter second integer: "); // prompt
16       int number2 = input.nextInt(); // read second number from user
17
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 1 of 4.)

```
18          if (number1 == number2)
19             System.out.printf("%d == %d%n", number1, number2);
20          }
21
22          if (number1 != number2) {
23             System.out.printf("%d != %d%n", number1, number2);
24          }
25
26          if (number1 < number2) {
27             System.out.printf("%d < %d%n", number1, number2);
28          }
29
30          if (number1 > number2) {
31             System.out.printf("%d > %d%n", number1, number2);
32          }
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 2 of 4.)

```
33
34          if (number1 <= number2) {
35             System.out.printf("%d <= %d%n", number1, number2);
36          }
37
38          if (number1 >= number2) {
39             System.out.printf("%d >= %d%n", number1, number2);
40          }
41       } // end method main
42    } // end class Comparison
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 3 of 4.)

```
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 4 of 4.)

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

- An `if` statement always begins with keyword `if`, followed by a condition in parentheses.
  - We've enclosed each body statement in a pair of braces, { }, creating what's called a compound statement or a block
  - 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

Good Programming Practice 2.11

Indent the statement(s) in the body of an `if` statement to enhance readability. IDEs typically do this for you, allowing you to specify the indent size.

Error-Prevention Tip 2.4

You don't need to use braces, `{ }`, around single-statement bodies, but you must include the braces around multiple-statement bodies. You'll see later that forgetting to enclose multiple-statement bodies in braces leads to errors. To avoid errors, as a rule, always enclose an `if` statement's body statement(s) in braces.

Common Programming Error 2.7

Placing a semicolon immediately after the right parenthesis after the condition in an `if` statement is often a logic error (although not a syntax error). The semicolon causes the body of the `if` statement to be empty, so the `if` statement performs no action, regardless of whether or not its condition is true. Worse yet, the original body statement of the `if` statement always executes, often causing the program to produce incorrect results.

Error-Prevention Tip 2.5

A lengthy statement can be spread over several lines. If a single statement must be split across lines, choose natural breaking points, 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.

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.

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.