

***Java Technology***

Semeter 1B, Academic Year 2020-2021

*Name of project:* **Mini Game Application**

*Course code:* **ITE1219E.201B04E**

*Instructors in charge:***TS. Nguyễn Hà Giang**

*Member List perform:*

Nguyễn Trà Thanh Trí 185050070

Trần Thị Thanh Trang 185050071

Nguyễn Vinh Hiển 185050531

Trần Thanh Linh 185050512

***Task Plan***

* Nguyễn Trà Thanh Trí
  + Modify the code
  + Code game pokemon
  + File.txt
* Trần Thị Thanh Trang
  + Code game puzzle
  + Make powerpoint slide
* Nguyễn Vinh Hiển
  + Interface design for the game
  + Code game snake of prey
* Trần Thanh Linh
  + The Theory section A
  + Code game Guess Number

Table of Contents

[Part A: The Theory section 4](#_Toc61866676)

[2.1 Introduction 4](#_Toc61866677)

[2.2 Your First Program in Java: Printing a Line of Text 4](#_Toc61866678)

[Commenting Your Programs 4](#_Toc61866679)

[Using Blank Lines 5](#_Toc61866680)

[Declaring a Class 5](#_Toc61866681)

[Class Names and Identifiers 5](#_Toc61866682)

[Declaring a Method 6](#_Toc61866683)

[Performing Output with System.out.println 6](#_Toc61866684)

[Using End-of-Line Comments on Right Braces for Readability 7](#_Toc61866685)

[Compiling and Executing Your First Java Application 7](#_Toc61866686)

[2.3 Modifying Your First Java Program 8](#_Toc61866687)

[Displaying a Single Line of Text with Multiple Statements 8](#_Toc61866688)

[Displaying Multiple Lines of Text with a Single Statement 9](#_Toc61866689)

[2.4 Displaying Text with printf 10](#_Toc61866690)

[2.5 Another Application: Adding Integers 11](#_Toc61866691)

[Import Declarations 12](#_Toc61866692)

[Declaring Class Addition 12](#_Toc61866693)

[Declaring and Creating a Scanner to Obtain User Input from the Keyboard 13](#_Toc61866694)

[Declaring Variables to Store Integers 13](#_Toc61866695)

[Prompting the User for Input 13](#_Toc61866696)

[Obtaining an int as Input from the User 14](#_Toc61866697)

[Prompting for and Inputting a Second int 14](#_Toc61866698)

[Using Variables in a Calculation 14](#_Toc61866699)

[Displaying the Result of the Calculation 14](#_Toc61866700)

[Java API Documentation 14](#_Toc61866701)

[2.6 Memory Concepts 15](#_Toc61866702)

[2.7 Arithmetic 15](#_Toc61866703)

[Arithmetic Expressions in Straight-Line Form 16](#_Toc61866704)

[Parentheses for Grouping Subexpressions 16](#_Toc61866705)

[Rules of Operator Precedence 16](#_Toc61866706)

[Sample Algebraic and Java Expressions 17](#_Toc61866707)

[Evaluation of a Second-Degree Polynomial 17](#_Toc61866708)

[2.8 Decision Making: Equality and Relational Operators 18](#_Toc61866709)

[Part B: Java Swing Application: 21](#_Toc61866710)

# Part A: The Theory section

|  |
| --- |
| **2.1** Introduction **2.6** Memory Concepts  **2.2** Your First Program in Java: Printing a **2.7** Arithmetic  Line of Text **2.8** Decision Making: Equality and  Relational Operators  **2.3** Modifying Your First Java Program  **2.4** Displaying Text with printf  **2.5** Another Application: Adding Integers  *Summary | Self-Review Exercises | Answers to Self-Review Exercises | Exercises | Making a Difference* |

## 2.1 Introduction

This chapter introduces Java application programming. We begin with examples of programs that display messages on the screen. You’ll learn how to instruct the computer to perform arithmetic calculations and save their results for later use.  
The last example demonstrates how to make decisions. This chapter uses tools from the JDK to compile and run programs.  
Dive Into videos at www.deitel.com/books/jhtp9/ to help you get started with the popular Eclipse and NetBeans integrated development environments.

## 2.2 Your First Program in Java: Printing a Line of Text

A Java application is a computer program that executes when you use the java command to launch the Java Virtual Machine . Later in this section we’ll discuss how to compile and run a Java application. Figure 2.1 shows the program followed by a box that displays its output. The program includes line numbers.

This example illustrates several important Java features.

|  |  |  |
| --- | --- | --- |
| **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.

### Commenting Your Programs

We insert comments to document programs and improve their readability. The Java compiler ignores comments, so they do not cause the computer to perform any action when the program is run. By convention, we begin every program with a comment indicating the figure number and file name.

The comment in line 1 begins with //, indicating that it is an end-of-line comment—it terminates at the end of the line on which the // appears. Java also has traditional comments, which can be spread over several lines as in These begin and end with delimiters, /\* and \*/. The compiler ignores all text between the delimiters. Java incorporated traditional comments and end-of-line comments from the C and C++ programming languages, respectively.

The compiler ignores all text between the delimiters. Javadoc comments enable you to embed program documentation directly in your programs. Such comments are the preferred Java documenting format in industry. The java doc utility program reads Javadoc comments and uses them to prepare your program’s documentation in HTML format.

// Fig. 2.1: Welcome1.java

// Text-printing program.

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

|  |
| --- |
| **Common Programming Error 2.1**  Forgetting one of the delimiters of a traditional or Javadoc comment is a syntax error. These rules are similar to a natural language’s grammar rules specifying sentence structure. 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. |

### Using Blank Lines

Line 3 is a blank line. Blank lines, space characters and tabs make programs easier to read. Together, they’re known as white space (or whitespace). The compiler ignores white space.

|  |
| --- |
| **Good Programming Practice 2.2**  Use blank lines and spaces to enhance program readability  *.* |

### Declaring a Class

Line 4 begins a class declaration for class Welcome1. Every Java program consists of at least one class that you define. The class keyword introduces a class declaration and is immediately followed by the class name . The complete list of keywords is shown in Appendix C.

public class Welcome1

### Class Names and Identifiers

By convention, class names begin with a capital letter and capitalize the first letter of each word they include . 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. Some valid identifiers are Welcome1, $value, \_value, m\_input Field1 and button7. Normally, an identifier that does not begin with a capital letter is not a class name.

Java is case sensitiv uppercase and lowercase letters are distinct so value and Value are different identifiers. For our application, the file name is Welcome1.java. You’ll learn more about public and non-public classes in Chapter 8. A left brace , {, begins the body of every class declaration.

|  |
| --- |
| **Common Programming Error 2.2**  A public class must be placed in a file that has the same name as the class plus the .java extension; otherwise, a compilation error occurs. For example, public class Welcome must be placed in a file named Welcome.java |
| **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**  Many IDEs insert indentation for you in all the right places. Most IDEs allow you to configure tabs such that a specified number of spaces is inserted each time you press the Tab key. |
| **Good Programming Practice 2.4**  Many IDEs insert indentation for you in all the right places. Most IDEs allow you to configure tabs such that a specified number of spaces is inserted each time you press the Tab key. |

### Declaring a Method

Line 6 is an end-of-line comment indicating the purpose of lines 7–10 of the program. Line 7 is the starting point of every Java application. For a Java application, one of the methods must be called main and must be defined as shown in line 7; otherwise, the Java Virtual Machine will not execute the application. Methods perform tasks and can return information when they complete their tasks. For now, simply mimicmain’s first line in your Java applications. In line 7, the String args in parentheses is a required part of the method main’s declaration—we discuss this in Chapter 7.

// main method begins execution of Java application

public static void main( String[] args )

|  |
| --- |
| **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 |

### Performing Output with System.out.println

Line 9 instructs the computer to perform an action namely, to 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. In recent versions of Microsoft Windows, the command window is the Command Prompt. In UNIX/Linux/Mac OS X, the command window is called a terminal window or a shell. Many programmers call it simply the command line. This is similar to what happens when you press the Enter key while typing in a text editor the cursor appears at the beginning of the next line in the document.

|  |
| --- |
| **Error-Prevention Tip 2.2**  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. |
| **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 line. |

### Using End-of-Line Comments on Right Braces for Readability

We include an end-of-line comment after a closing brace that ends a method declaration and after a closing brace that ends a class declaration. For example, line 10 indicates the closing brace of method main, and line 11 indicates the closing brace of class Welcome1. Each comment indicates the method or class that the right brace terminates.

} // end method main

} // end class Welcome1

### Compiling and Executing Your First Java Application

We’re now ready to compile and execute our program. We assume you’re using the Java Development Kit’s command-line tools, not an IDE. Our Java Resource Centers at www.deitel.com/ResourceCenters.html provide links to tutorials that help you get started with several popular Java development tools, including NetBeans™, Eclipse™ and others. We’ve also posted NetBeans and Eclipse videos at www.deitel.com/books/jhtp9/ to help you get started using these popular IDEs.  
Welcome1.class containing the platform-independent Java bytecodes that represent our application.

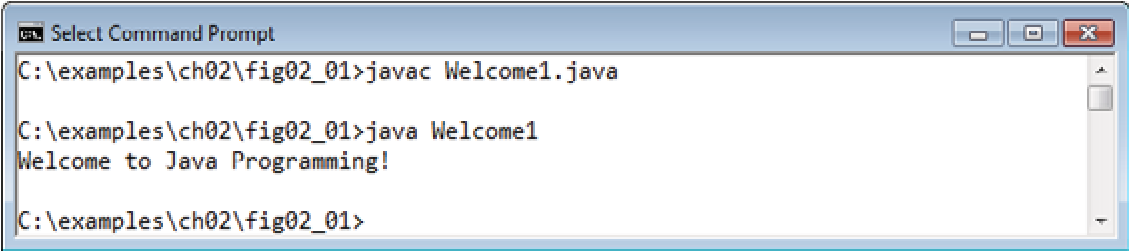
cd c:\examples\ch02\fig02\_01

cd ~/examples/ch02/fig02\_01

javac Welcome1.java

|  |
| --- |
| **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. |
| **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. |

Figure 2.2 shows the program of Fig. 2.1 executing in a Microsoft Windows 7 Command Prompt window. To execute the program, type java Welcome1. This command launches the JVM, which loads the .class file for class Welcome1. The command omits the .class file-name extension; otherwise, the JVM will not execute the program. The JVM calls method main. Next, the statement at line 9 of main displays «Welcome to Java Programming!»



You type this

command to execute

the application

The program outputs to the screen

Welcome to Java Programming!

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

|  |
| --- |
| **Error-Prevention Tip 2.7**  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**

In this section, we modify the example in Fig. 2.1 to print text on one line by using multiple statements and to print text on several lines by using a single statement.

### Displaying a Single Line of Text with Multiple Statements

Welcome to Java Programming! can be displayed several ways. Class Welcome2, shown in Fig. 2.3, uses two statements to produce the output shown in Fig. 2.1

1. // Fig. 2.3: Welcome2.java
2. // Printing a line of text with multiple statements. **3**
3. public class Welcome2
4. {
5. // main method begins execution of Java application
6. public static void main( String[] args )
7. {

|  |
| --- |
| System.out.print( "Welcome to " );  System.out.println( "Java Programming!" ); |

**9**

**10**

1. } // end method main
2. } // end class Welcome2

Welcome to Java Programming!

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

The program is similar to Fig. 2.1, so we discuss only the changes here. Line 4 begins the Welcome2 class declaration. Lines 9–10 of method main display one line of text. Each print or println statement resumes displaying characters from where the last print or println statement stopped displaying characters.

// Printing a line of text with multiple statements.

System.out.print( "Welcome to " );

System.out.println( "Java Programming!" );

### Displaying Multiple Lines of Text with a Single Statement

A single statement can display multiple lines by using newline characters, which 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. The program in Fig. Most of the program is identical to those in Fig.2.4

1. // Fig. 2.4: Welcome3.java
2. // Printing multiple lines of text with a single statement. **3**
3. public class Welcome3
4. {
5. // main method begins execution of Java application
6. public static void main( String[] args )
7. {

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| \n |  | \n |  | \n |

1. System.out.println( "Welcome to Java Programming!" );
2. } // end method main
3. } // end class Welcome3

|  |
| --- |
| Welcome to Java Programming! |

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

Line 2 is a comment stating the program’s purpose. Line 4 begins the Welcome3 class declaration.  
Line 9 displays four separate lines of text in the command window. Note, however, that the paired characters and n do not appear on the screen. The backslash is an escape character. which has special meaning to System.out’s print and println methods. When a newline character appears in a string being output with System.out, the newline character causes the screen’s output cursor to move to the beginning of the next line in the command window.

// Printing multiple lines of text with a single statement.

System.out.println( "Welcome\nto\nJava\nProgramming!" );

|  |  |
| --- | --- |
| 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**

The System.out.printf method displays formatted data. Figure 2.6 uses this method to output the strings «Welcome to» and «Java Programming!». Lines 9–10 call method System.out.printf to display the program’s output. When a method requires multiple arguments, they’re placed in a comma-separated list.



**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. public class Welcome4
4. {
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!" ); |

**9**

**10**

1. }
2. // end method main
3. } // end class Welcome4

Welcome to

Java Programming!

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

Lines 9–10 represent only one statement. Java allows large statements to be split over many lines. We indent line 10 to indicate that it’s a continuation of line 9. Each format specifier is a placeholder for a value and specifies the type of data to output. Format specifiers also may include optional formatting information. Format specifiers begin witha percent sign followed by a character that represents the data type.

Common Programming Error 2.4

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



At the first format specifier’s position, printf substitutes the value of the first argument after the format string. At each subsequent format specifier’s position, printf substitutes the value of the next argument. So this example substitutes «Welcome to» for the first %s and «Java Programming!» for the second %s.

## 2.5 **Another Application: Adding Integers**

This program must keep track of the numbers supplied by the user for the calculation later in the program. 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

1. // Fig. 2.7: Addition.java
2. // Addition program that displays the sum of two numbers.

|  |
| --- |
| import java.util.Scanner; // program uses class Scanner |

**4**

1. public class Addition
2. {
3. // main method begins execution of Java application
4. public static void main( String[] args )
5. {

|  |
| --- |
| // create a Scanner to obtain input from the command window  Scanner input = new Scanner( System.in ); |

**10**

**11**

**12**

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

**13**

**14**

**15**

**16**

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

**17** System.out.print( "Enter first integer: " ); // prompt

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

**18**

**19**

|  |
| --- |
| System.out.print("Entersecondinteger: ") ;  number2 = input.nextInt(); // read second number from user |

**20**

**21**

**22**

|  |
| --- |
| sum = number1 + number2; // add numbers, then store total in sum |

**23**

**24**

|  |
| --- |
| System.out.printf( "Sum is %d\n", sum ); // display sum |

**25**

1. } // end method main
2. } // 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.)

### Import Declarations

Lines 1–2 state the figure number, file name and purpose of the program. A great strength of Java is its rich set of predefined classes that you can reuse rather than «reinventing the wheel.» These classes are grouped into packages—named groups of related classes—and are collectively referred to as the Java class library, or theJava Application Programming Interface. Line 3 is an import declaration that helps the compiler locate a class that’s used in this program.

// Fig. 2.7: Addition.java

// Addition program that displays the sum of two numbers.

|  |
| --- |
| **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. |

### Declaring Class Addition

The left brace marks the beginning of method main’s body, and the corresponding right brace marks its end. Method main is indented one level in the body of class Addition, and the code in the body of main is indented another level for readability.

public class Addition

### Declaring and Creating a Scanner to Obtain User Input from the Keyboard

A variable is a location in the computer’s memory where a value can be stored for use later in a program. A variable’s name enables the program to access the value of the variable in memory. A variable’s name can be any valid identifier. A variable’s type specifies what kind of information is stored at that location in memory.

Line 11 is a variable declaration statement that specifies the name and type of a variable that’s used in this program. A Scanner enables a program to read data for use in a program. The Scanner translates these bytes into types that can be used in a program.

Scanner input = new Scanner( System.in );

### Declaring Variables to Store Integers

The variable declaration statements in lines 13–15 declare that variables number1, number2 and sum hold data of type int they can hold integer values . These variables are not yet initialized. Real numbers contain decimal points, such as 3.4, 0.0 and –11.19. Variables of type char represent individual characters, such as an uppercase letter, a digit , a special character or an escape sequence . Appendix D summarizes the characteristics of the eight primitive types .Several variables of the same type may be declared in a single declaration with the variable names separated by commas.

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

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

|  |
| --- |
| **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. |

### Prompting the User for Input

Line 17 uses System. So, System is a class. Class System is part of package java. Notice that class System is not imported with an import declaration at the beginning of the program.

System.out.print( "Enter first integer: " ); // prompt

|  |
| --- |
| **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. |

### Obtaining an int as Input from the User

Line 18 uses Scanner object input’s next Int method to obtain an integer from the user at the keyboard. At this point the program waits for the user to type the number and press the Enter key to submit the number to the program. If not , a runtime logic error will occur and the program will terminate. Chapter 11, Exception Handling: A Deeper Look, discusses how to make your programs more robust by enabling them to handle such errors.

In line 18 ,we place the result of the call to method next Int invariable number1 by using the assignment operator, =. The statement is read as «number1 gets the value of input.next Int.» Operator = is called a binary operator, because it has two operands—number1 and the result of the method call input.next Int.

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

|  |
| --- |
| **Good Programming Practice 2.10**  Placing spaces on either side of a binary operator makes the program more readable. |

### Prompting for and Inputting a Second int

Line 20

System.out.print( "Enter second integer: " ); // prompt

prompts the user to input the second integer. Line 21

number2 = input.nextInt(); // read second number from user

reads the second integer and assigns it to variable number2.

### Using Variables in a Calculation

Line 23 is an assignment statement that calculates the sum of the variables number1 and number2 then assigns the result to variable sum by using the assignment operator, =. In the preceding statement, the addition operator is a binary operator—its two operands are the variables number1 and number2. In fact, an expression is any portion of a statement that has a value associated with it.

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

### Displaying the Result of the Calculation

The format specifier %d is a placeholder for an int value —the letter d stands for «decimal integer .»The remaining characters in the format string are all fixed text .So ,method printf displays «Sum is », followed by the value of sum and a newline. Calculations can also be performed inside printf statements. We could have combined the statements at lines 23 and 25 into the statement The parentheses around the expression number1 + number2 are not required—they’re included to emphasize that the value of the entire expression is output in the position of the %d format specifier.

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

System.out.printf( "Sum is %d\n", ( number1 + number2 ) );

### Java API Documentation

A web-based version of this documentation can be found at. You can download it from Appendix E shows how to use this documentation.

download.oracle.com/javase/6/docs/api/

[www.oracle.com/technetwork/java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

## 2.6 **Memory Concepts**

Variable names such as number1, number2 and sum actually correspond to locations in the computer’s memory. Every variable has a name ,a type ,a size and a value. The computer places that integer value into location number1 , replacing the previous value in that location. The computer places that integer value into location number2.  
The memory now appears as shown in Fig. After sum has been calculated, memory appears as shown in Fig. The values of number1 and number2 appear exactly as they did before they were used in the calculation of sum. These values were used, but not destroyed, as the computer performed the calculation.

When a value is read from a memory location, the process is nondestructive.

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

45

number1

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

number2 = input.nextInt(); // read second number from user

45

72

number1

number2

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

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

45

72

117

number1

number2

sum

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

## 2.7 **Arithmetic**

Most programs perform arithmetic calculations. The arithmetic operators are summarized in Fig. 2.11. Note the use of various special symbols not used in algebra. The arithmetic operators in Fig. 2.11 are binary operators, because each operates on two operands. For example, the expression f+7contains the binary operator + and the two operands f and 7. Any fractional part in integer division is simply discarded no rounding occurs. Java provides the remainder operator, %, which yields the remainder after division. The expression x % y yields the remainder after x is divided by y. Thus,7%4 yields 3, and 17 % 5 yields 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Java operation | Operator |  | Algebraic expression | Java expression |
| Addition | + |  | *f* + 7 | f + 7 |
| Subtraction | – |  | *p* – *c* | p - c |
| Multiplication | \* |  | *bm* | b \* m |
| Division  Remainder | /  % |  | *x*  *x* /*y* or *--y* or *x* ÷ *y r* mod *s* | x / y r % s |

**Fig. 2.11** | Arithmetic operators.

### Arithmetic Expressions in Straight-Line Form

Arithmetic expressions in Java must be written in straight-line form to facilitate entering programs into the computer. Thus, 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. The following algebraic notation is generally not acceptable to compilers:

### Parentheses for Grouping Subexpressions

Parentheses are used to group terms in Java expressions in the same manner as in algebraic expressions. For example, to multiply a times the quantity b + c, we write. If an expression contains nested parentheses, such as the expression in the innermost set of parentheses is evaluated first.

a \* ( b + c )

( ( a + b ) \* c )

### Rules of Operator Precedence

Multiplication, division and remainder operations are applied first. If an expression contains several such operations, they ’reapplied from left to right. Multiplication, division and remainder operators have the same level of precedence.  
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. These rules enable Java to apply operators in the correct order. Some operators associate from right to left. Figure 2.12 summarizes these rules of operator precedence.

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

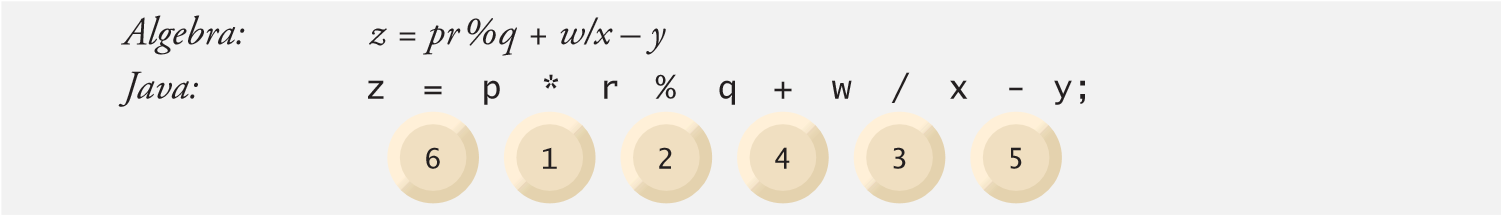
**Fig. 2.12** | Precedence of arithmetic operators.

### Sample Algebraic and Java Expressions

Now let’s consider several expressions in light of the rules of operator precedence. Each example lists an algebraic expression and its Java equivalent. The following is an example of an arithmetic mean of five terms:  
If the parentheses are erroneously omitted, we obtain a + b + c + d + e /5, which evaluates as.

Here’s an example of the equation of a straight line:  
No parentheses are required. The multiplication operator is applied first because multiplication has a higher precedence than addition. The circled numbers under the statement indicate the order in which Java applies the operators. The \*, % and / operations are evaluated first in left-to-right order because they have higher precedence than + and -.

|  |  |  |
| --- | --- | --- |
| *Algebra:*  *Java:* | | *a* + *b* + *c* + *d* + *e m* = -------------------------------------  5  m = ( a + b + c + d + e ) / 5; |
| *e a* + *b* + *c* + *d* + -  5 | | |
| *Algebra: Java:* | *y* = *mx* + *b* y = m \* x + b; | |



### Evaluation of a Second-Degree Polynomial

The multiplication operations are evaluated first in left-to-right order, because they have higher precedence than addition. The addition operations are evaluated next from left to right. Suppose that a, b, c and x are initialized as follows: a=2, b=3, c=7and x=5. Figure 2.13 illustrates the order in which the operators are applied. You can use redundant parentheses to make an expression clearer. For example, the preceding statement might be parenthesized as follows:



*(*

*Leftmost multiplication*

*)*

*)*

*(*

*Leftmost multiplication*

*)*

*(*

*Multiplication before addition*

*)*

*Leftmost addition*

*(*

*)*

*Last addition*

*(*

*Last operation—place*

*(*

72

*in*

y

*)*

*Step 1.*

y=2\*5\*5+3\*5+7;

is

10

2\*5

*Step 2.*

y=10\*5+3\*5+7;

50

is

10\*5

*Step 3.*

y=50+3\*5+7;

3\*5

is

15

*Step 4.*

y=50+15+7;

is

50 + 15

65

*Step 5.*

y=65+7;

72

65+7

is

*Step 6.*

y=72

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

y = ( a \* x

\* x ) + ( b \* x ) + c;

## 2.8 Decision Making: Equality and Relational Operators

This section introduces Java’s if selection statement, which allows a program to make a decision based on a condition’s value. If the condition in an if statement is true, the body of the if statement executes.  
Conditions in if statements can be formed by using the equality operators and relational operators summarized in Fig.

Both equality operators have the same level of precedence, which is lower than that of the relational operators. The relational operators all have the same level of precedence and also associate from left to right.

Figure 2.15 uses six if statements to compare two integers input by the user. If the condition in any of these if statements is true, the statement associated with that if statement executes ; otherwise ,the statement is skipped. We use a Scanner to input the integers from the user and store them in variables number1 and number2. The program compares the numbers and displays the results of the comparisons that are true.

|  |  |  |  |
| --- | --- | --- | --- |
| 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**

* 1. public class Comparison
  2. {
  3. // main method begins execution of Java application
  4. public static void main( String[] args )
  5. {
  6. // create Scanner to obtain input from command line
  7. Scanner input = new Scanner( System.in );

**13**

1. int number1; // first number to compare
2. int number2; // second number to compare

**16**

1. System.out.print( "Enter first integer: " ); // prompt
2. number1 = input.nextInt(); // read first number from user

**19**

1. System.out.print( "Enter second integer: " ); // prompt
2. number2 = input.nextInt(); // read second number from user

**22**

|  |
| --- |
| if ( number1 == number2 )  System.out.printf( "%d == %d\n", number1, number2 ); |

**23**

**24**

**25**

|  |
| --- |
| if ( number1 != number2 )  System.out.printf( "%d != %d\n", number1, number2 ); |

**26**

**27**

**28**

|  |
| --- |
| if ( number1 < number2 )  System.out.printf( "%d < %d\n", number1, number2 ); |

**29**

**30**

**31**

**Fig. 2.15** | Compare integers using if statements, relational operators and equality operators.

(Part 1 of 2.)

|  |
| --- |
| if ( number1 > number2 )  System.out.printf( "%d > %d\n", number1, number2 ); |

**32**

**33**

**34**

|  |
| --- |
| if ( number1 <= number2 )  System.out.printf( "%d <= %d\n", number1, number2 ); |

**35**

**36**

**37**

|  |
| --- |
| if ( number1 >= number2 )  System.out.printf( "%d >= %d\n", number1, number2 ); |

**38**

**39**

**40** } // end method main

**41**

}

// end class Comparison

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 2 of2)

The declaration of class Comparison begins at line 6.  
The class’s main method begins the execution of the program. Line 12 declares Scanner variable input and assigns it a Scanner that inputs data from the standard input . Lines 14–15 declare the int variables used to store the values input from the user. Lines 17–18 prompt the user to enter the first integer and input the value, respectively.The input value is stored in variable number1.  
Lines 23–24 compare the values of number1 and number2 to determine whether they’re equal. An if statement always begins with keyword if, followed by a condition in parentheses. An if statement expects one statement in its body, but may contain multiple statements if they’re enclosed in a set of braces. The indentation of the body statement shown here is not required, but it improves the program’s readability by emphasizing that the statement in line 24 is part of the if statement that begins at line 23. If the condition in one or more of the if statements is true, the corresponding body statement executes.

There’s 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. For example, would actually be interpreted by Java as where the semicolon on the line by itself—called the empty statement—is the statement to execute if the condition in the if statement is true. When the empty statement executes, no task is performed.

public class Comparison

Scanner input = new Scanner( System.in );

int number1; // first number to compare int number2; // second number to compare

System.out.print( "Enter first integer: " ); // prompt number1 = input.nextInt(); // read first number from user

System.out.print( "Enter second integer: " ); // prompt number2 = input.nextInt(); // read second number from user

|  |
| --- |
| if ( number1 == number2 )  System.out.printf( "%d == %d\n", number1, number2 ); |

|  |
| --- |
| **Common Programming Error 2.6**  Conf using the equality operator, ==, with the assignment operator, =,can causea logic error or a syntax error. The equality operator should be read as «is equalto »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.

|  |
| --- |
| if ( number1 == number2 ); // logic error  System.out.printf( "%d == %d\n", number1, number2 ); |
| **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.  **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. |

# Part B: Java Swing Application:

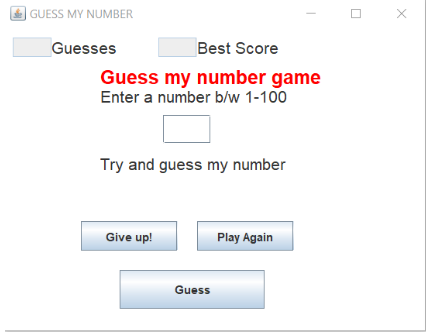
# Login interface:

# The game's main interface:

## Game guess the number:

### Mission of game:

### Interface of the game:



## Game snake of prey:

### Mission of game:

### Interface of the game:



## Game pokemon:

### Mission of game:

### Interface of the game:

## Game puzzle:

### Mission of game:

### Interface of the game:

# Associated with Table:

# 