Solution Manual for Java Programming 7th Edition Farrell 1285081951 9781285081953

Chapter 2

Using Data

[redacted]

Instructor's Manual Table of Contents

Overview

Chapter 2 introduces the eight primitive data types in the Java language. Students will learn to work with integer, floating-point, Boolean, and character values. Arithmetic and comparison operators are introduced. Finally, students will learn to create input and confirm dialog boxes using the `JOptionPane` class.

Objectives

- Declare and use constants and variables
- Use integer data types
- Use the `boolean` data type
- Use floating-point data types
- Use the `char` data type
- Use the `Scanner` class to accept keyboard input
- Use the `JOptionPane` class to accept GUI input
- Perform arithmetic
- Understand type conversion

Teaching Tips

Declaring and Using Constants and Variables

1. Define **variables** and **constants**. Explain the difference between variables and constants. Using Table 2-1, explain the concept of data types and introduce the eight primitive data types. Suggest uses for the primitive types.

2. Define a **primitive type** and what it means in Java.

3. If your students have worked with a database system or another programming language, compare these data types with those found elsewhere. Emphasize the similarity between concepts.

4. Define a **reference type** as a Java class. In Chapter 3, students will create classes out of primitive types and other reference types.

Declaring Variables

1. Describe the components of a **variable declaration**. Note that it is common practice to use **camel casing**, which is when variable names begin with a lowercase letter and any subsequent words within the variable name are capitalized.

2. Demonstrate how to create several different variables of differing types. If possible, demonstrate using your Java compiler.

Teaching Tip	Discuss the importance of choosing meaningful names for variables.

3. Define the **initialization** of variables. Provide several examples of variable initializations. Emphasize that the variable must be on the left side of the **assignment operator**. Briefly discuss the **associativity** of operators.

4. Spend time discussing what Java does when it encounters an **uninitialized variable**. Define the concept of a **garbage value**. If possible, demonstrate using your Java compiler.

5. Point out the single line with multiple declarations on page 54. Emphasize that while this is legal code, it should be avoided to ensure program readability.

Declaring Named Constants

1. Define a **named constant**. Explain how to create a named constant using the **`final`** keyword. Note that it is common practice to use all uppercase letters with constants. Rather than using camel casing to differentiate words in a constant, suggest using an underscore between words.

2. Demonstrate how to create several constants. If possible, demonstrate this using your compiler. Refer to the examples on page 55.

3. Define a **blank `final`**. Demonstrate how to create a blank `final`, and discuss when this type of constant might be appropriate.

4. Identify the benefits of using named constants over literal values.

5. Define a **magic number**. Demonstrate the difficulty of working with magic numbers in large programs. Page 55 lists several reasons to use constants instead of magic numbers.

The Scope of Variables and Constants

1. Define **scope** as the area in which a data item is visible to a program and in which you can refer to it using its simple identifier.

2. Explain that a variable or constant is in scope from the point it is declared until the end of the **block of code** in which the declaration lies.

3. An excellent analogy is the classroom. Items written on your board are not visible in the room next door. Also, students named Tim in your class are quite different from those named Tim next door.

Concatenating Strings to Variables and Constants

1. Define **concatenation**. Discuss the shaded code in Figure 2-1 on page 56.

2. Explain concatenation as an **operation**. Compare it to a math operation. This is probably the first time your students have encountered operations outside of a math or science context.

3. Figure 2-3 shows concatenation used in the `JOptionPane.showMessageDialog` method. Point out the **null `String`** as a simple way to display numeric output anywhere string elements are expected.

Pitfall: Forgetting That a Variable Holds One Value at a Time

1. Mention that each constant can hold only one value for the duration of a program.

2. Explain how to correctly swap the values of two variables. Refer to page 58 for the sample code to swap variable contents.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 59.

You Do It

1. Students should follow the steps in the book on pages 59–62 to create a Java application that declares and uses a variable.

Learning About Integer Data Types

1. Mathematically define **integers** and whole numbers. It is likely that your students have forgotten what numbers these represent.

2. Describe the **`int`**, **`byte`**, **`short`**, and **`long`** data types. Using Table 2-2, explain the storage capacity of each type. Spend a little time discussing why programmers must care about the storage capacity.

3. Demonstrate what happens if a math expression results in a number outside of the range of a data type. For example, consider that the code `byte dogAge = (byte) (42 * 7);` results in the variable `dogAge` holding 38. The value comes from subtracting 256 from the "real" answer of 294.

Quick Quiz 1

1. A data item is constant when it cannot be changed while a program is running. A data item is ____ when it might change.
   Answer: variable

2. An item's _____ describes the type of data that can be stored there, how much memory the item occupies, and what types of operations can be performed on the data. Answer: data type

3. True or False: A variable declaration is a statement that reserves a named memory location.
   Answer: True

4. The ____ types are all variations of the integer type. Answer: `byte`, `short`, and `long`

5. The + sign in the following expression refers to the ____ operation.
   ```
   System.out.println("My age: " + ageVar);
   ```
   Answer: concatenation

Two Truths and a Lie

1. Discuss the two truths and a lie on page 63.

You Do It

1. Students should follow the steps in the book on pages 64–67 to create a Java application that declares and uses a variable.

Using the `boolean` Data Type

1. Introduce the concept of a **`boolean` variable**, which can have one of two values: `true` or `false`.

2. Using Table 2-3, describe the **relational operators** available in Java. Note that the result of each comparison is a `boolean` value.

3. Discuss that these concepts will be very important in later chapters.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 69.

Teaching Tip	The Boolean type is named after George Boole, an English mathematician.

Learning About Floating-Point Data Types

1. Define **floating-point** numbers. Describe the type of numbers that floating-point values represent.

2. Using Table 2-4, introduce the two floating-point data types: **`double`** and **`float`**. Make sure that students understand the concept of **significant digits**. Reiterate the concept of precision. `Double` variables are more precise than `float` variables.

3. Demonstrate how to create several floating-point types. As shown on page 70, discuss why you need to type the letter *F* after the number in `float` declarations and instantiations.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 70.

Using the `char` Data Type

1. Explain the use of the **`char`** data type to hold a single character. A constant character value is placed between single quotation marks.

2. Describe the Unicode system as holding all symbols for all languages on the planet. Unicode helps Java be useful around the world. Some Unicode values are listed in Table 2-5; the entire table can be found at Unicode.org.

3. Demonstrate how to store Unicode values in the `char` data type. For example, this line will store the fraction ½ in the `char` variable `half`: `char half = '\u00BD';`.

4. Introduce the built-in Java type **`String`**.

5. Describe the purpose of an **escape sequence**. Using Table 2-6, describe common escape sequences. Discuss the differences between Figures 2-14 and 2-15.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 74.

You Do It

1. Students should follow the steps in the book on page 75 to create a Java application that declares and uses a `char` variable.

Using the `Scanner` Class to Accept Keyboard Input

1. Define the `Scanner` class. Discuss why it is much better than traditional character-bycharacter input.

2. Demonstrate how to use the `Scanner` class to capture keyboard input from the **standard input device** (keyboard) represented by `System.in`. Reiterate the importance of the **prompt**. Selected methods of the `Scanner` class are listed in Table 2-7 on page 77.

3. Review the `GetUserInfo` class in Figure 2-17 and the program output in Figure 2-18 on page 78. Discuss the importance of **echoing the input**.

4. Demonstrate what happens if the user types a string into a `nextInt`() prompt. If desired, you can demonstrate how to correctly input data into `String`s, and then convert the `String`s to the proper data type. This is covered a little later in the chapter.

Teaching Tip	Students can learn more about the `Scanner` class with following documentation: http://java.sun.com/javase/6/docs/api/java/util/Scanner.html.

Pitfall: Using `nextLine()` Following One of the Other `Scanner` Input Methods

1. Illustrate the problems that may occur when using the `nextLine()` method after one of the other `Scanner` class input methods. Use the code samples in Figures 2-19 and 2-21 to aid the discussion. Make sure that students are familiar with the concept of the **keyboard buffer**.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 82.

You Do It

1. Students should follow the steps in the book on pages 82–85 to create a Java application that accepts keyboard input.

Using the `JOptionPane` Class to Accept GUI Input

1. Remind students about using the `JOptionPane` class to create dialog boxes. Introduce an **input dialog box** and a **confirm dialog box**.

Using Input Dialog Boxes

1. Show how to use the **`showInputDialog()` method** of `JOptionPane`. Review the code in Figure 2-26, which produces the output shown in Figures 2-27 and 2-28.

2. Using the code above Figure 2-29, demonstrate how the input boxes can be modified with different titles and icons.

3. Describe how to convert a `String` into a primitive class using the **type-wrapper classes**: `Integer`, `Float`, and `Double`. Figure 2-30 illustrates how to convert a `String` class into `double` and `int` variables.

Teaching Tip	Define the term **parse**. Its literal meaning is to break an object into component parts. It can be roughly defined as reading the contents of an object.

Using Confirm Dialog Boxes

1. Explain how to use the **`showConfirmDialog()` method** of `JOptionPane`. Review the `AirlineDialog` class in Figure 2-32.

2. Using the code above Figure 2-35, demonstrate how confirm dialog boxes can be modified with different titles and icons.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 91.

Performing Arithmetic

1. Using Table 2-8, show that Java provides all of the **standard arithmetic operators**. Remind students that the rules of operator precedence apply in a program just as they do in math.

2. Define **operand** and **binary operators**. Identify them in a simple math expression.

3. Differentiate between **integer division** and **floating-point division**. Use examples for each. Make sure that students understand that in integer division, any fractional portion of a division result will be lost when both operators are of an integer data type.

4. Define the **modulus** or **remainder operator**. Provide numerous examples of this operator's uses. Students often have a difficult time grasping the concept of modulus. You will need to discuss this operator often in class.

Teaching Tip	Students may not be as familiar with the modulus operator, %, as with other arithmetic operators.

Associativity and Precedence

1. Remind students about the traditional order of operations acronym, PEMDAS, which they may have learned in grade school. Spell it out for them: "Please Excuse My Dear Aunt Sally," or "Parenthesis, Exponents, Multiplication or Division, and Addition or Subtraction." Remind students that math expressions are evaluated from left to right both in Java and in pure math.

2. Define **operator precedence** and refer to Table 2-9. Point out that operator precedence aligns nicely with PEMDAS. Using your Java environment, demonstrate how operator precedence works using your Java environment.

Writing Arithmetic Statements Efficiently

1. Use examples to explain how to avoid unnecessary repetition of arithmetic statements. Point out the examples on page 94. Have students identify the `grossPay` variable.

Pitfall: Not Understanding Imprecision in Floating-Point Numbers

1. Mention that integer values are exact, but floating-point numbers frequently are only approximations.

2. Explain that imprecision leads to several problems, including:
   a. Floating-point output might not look like what you expect or want.
   b. Comparisons with floating-point numbers might not be what you expect or want.

3. Using your Java environment, provide examples of the imprecision in floating-point numbers.

Teaching Tip	To get precise floating-point values, you need to use `java.math.BigDecimal`.

Teaching Tip	Students may not be able to reproduce the output shown in Figure 2-37.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 96.

You Do It

1. Students should follow the steps in the book on pages 96–98 to create a Java application that uses arithmetic operators.

Quick Quiz 2

1. A relational operator compares two items; an expression containing a comparison operator has a(n) ____ value.
   Answer: `boolean`

2. A(n) ____ data type can hold floating-point values of up to six or seven significant digits of accuracy. Answer: `float`

3. You use arithmetic ____ to perform calculations with values in your programs. Answer: operators

4. When you combine mathematical operations in a single statement, you must understand ____, or the rules for the order in which parts of a mathematical expression are evaluated.
   Answer: operator precedence

5. The ____ operator returns the remainder of integer division.
   Answer: `modulus` or %

Understanding Type Conversion

1. Describe the concept of **type conversion**. Discuss why this is an important concept.

Automatic Type Conversion

1. Define a **unifying type**. Using Figure 2-41, explain how Java promotes variables to a unifying type by selecting the largest data type in the expression.

Teaching Tip	Ask students to write a program that illustrates the use of unifying types and type casting.

Explicit Type Conversions

1. Remind students of the *F* placed after numbers to convert a `double` into `float`.

2. Define **type casting**. Demonstrate how to create an **explicit conversion** using the **cast operator**. Be sure to provide an example demonstrating why this is important. A good example is dividing 1 and 2, expecting .5 but getting 0.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 102.

You Do It

1. Students should follow the steps in the book on pages 102–104 to create a Java application that uses unifying types and casting.

Don't Do It

1. Review this section, discussing each point with the class.

Quick Quiz 3

1. True or False: The cast type is the type to which all operands in an expression are converted so that they are compatible with each other. Answer: False

2. ____ casting forces a value of one data type to be used as a value of another type. Answer: Type

3. True or False: A character that is a digit is represented in computer memory differently than a numeric value represented by the same digit.
   Answer: True

4. A(n) ____ dialog box asks a question and provides a text field in which the user can enter a response.
   Answer: input

Class Discussion Topics

1. Why do you think it is important to have a variety of different data types for integers and floating-point numbers?

2. Why might it be necessary to perform type casting?

3. How could the `JOptionPane` be used for a video game?

Additional Projects

1. Create a Java application that performs two arithmetic and two comparison operations on the same set of variables. Print the results to the console.

2. Create a Java application that prompts the user for two values using input dialog boxes and then displays the sum of the values using a message dialog box.

3. Find out how Android handles the `JOptionPane`.

Additional Resources

1. Primitive Data Types:
   http://download.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

2. More on `JOptionPane`:
   http://download.oracle.com/javase/tutorial/uiswing/components/dialog.html

3. Summary of Operators:
   http://download.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html

4. Operators: http://download.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

5. Conversions and Promotions:
   http://docs.oracle.com/javase/specs/jls/se5.0/html/conversions.html

Key Terms

- **Assignment:** the act of providing a value for a variable.
- **Assignment operator:** the equal sign (=). Any value to the right of the equal sign is assigned to the variable on the left of the equal sign.
- **Associativity:** refers to the order in which operands are used with operators.
- **Binary operators:** require two operands.
- **Blank `final`:** a `final` variable that has not yet been assigned a value.
- **Block of code:** the code contained within a set of curly braces.
- **`boolean` variable:** can hold only one of two values: `true` or `false`.
- **`byte`:** a data type that holds very small integers, from –128 to 127.
- **Camel casing:** a style in which an identifier begins with a lowercase letter and subsequent words within the identifier are capitalized.
- **Cast operator:** performs an explicit-type conversion. It is created by placing the desired result type in parentheses before the expression to be converted.
- **`char`:** a data type used to hold any single character.
- **Comparison operator:** another name for a relational operator.
- **Concatenated:** combine a value with another value.
- **Confirm dialog box:** displays the options Yes, No, and Cancel.
- **Constant:** a data item that cannot be changed during the execution of an application.
- **Consumed:** retrieve and discard an entry without using it.
- **Data type:** describes the type of data that can be stored there, how much memory the item occupies, and what types of operations can be performed on the data.
- **`double`:** a data type that can hold a floating-point value of up to 14 or 15 significant digits of accuracy.
- **Double-precision floating-point number:** stored in a double.
- **Echoing the input:** repeat the user's entry as output so that the user can visually confirm the entry's accuracy.
- **Escape sequence:** begins with a backslash followed by a character; the pair represents a single character.
- **Explicit conversion:** the data-type transformation caused by using a cast operator.
- **`final`:** the keyword that precedes named constants.
- **`float`:** a data type that can hold a floating-point value of up to six or seven significant digits of accuracy.
- **Floating-point:** a number that contains decimal positions.
- **Floating-point division:** the operation in which two values are divided and either or both are floating-point values.
- **Garbage value:** the unknown value stored in an uninitialized variable.
- **Implicit conversion:** the automatic transformation of one data type to another.
- **Initialization:** an assignment made when you declare a variable.

- **Input dialog box:** asks a question and provides a text field in which the user can enter a response.
- **`int`:** the data type used to store integers.
- **Integer:** a whole number without decimal places.
- **Integer division:** the operation in which one integer value is divided by another; the result contains no fractional part.
- **Keyboard buffer:** a small area of memory where keystrokes are stored before they are retrieved into a program.
- **Literal constant:** a value that is taken literally at each use.
- **`long`:** a data type that holds very large integers, from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- **Lvalue:** an expression that can appear on the left side of an assignment statement.
- **Magic number:** a value that does not have immediate, intuitive meaning, or a number that cannot be explained without additional knowledge. Unnamed constants are magic numbers.
- **Modulus operator:** the remainder operator; also called **mod**.
- **Named constant:** a memory location whose declaration is preceded by the keyword `final` and whose value cannot change during program execution.
- **Null `String`:** an empty `String` created by typing a set of quotation marks with nothing between them.
- **Numeric constant:** a number whose value is taken literally at each use.
- **Operand:** a value used in an arithmetic statement.
- **Operator precedence:** the rules for the order in which parts of a mathematical expression are evaluated.
- **Parse:** to break into component parts.
- **Primitive type:** a simple data type. Java's primitive types are `byte`, `short`, `int`, `long`, `float`, `double`, `char`, and `boolean`.
- **Promotions:** implicit conversions.
- **Prompt:** a message that requests and describes user input.
- **Reference types:** complex data types that are constructed from primitive types.
- **Relational operator:** compares two items. An expression that contains a relational operator has a Boolean value.
- **Remainder operator:** the percent sign. When it is used with two integers, the result is an integer with the value of the remainder after division takes place.
- **Rvalue:** an expression that can appear only on the right side of an assignment statement.
- **Scientific notation:** a display format that more conveniently expresses large or small numeric values. A multidigit number is converted to a single-digit number and multiplied by 10 to a power.
- **Scope:** the area in which a variable is visible to a program.
- **`short`:** a data type that holds small integers, from –32,768 to 32,767.
- **`showConfirmDialog()` method:** in the `JOptionPane` class; used to create a confirm dialog box.

- **`showInputDialog()` method:** used to create an input dialog box.
- **Significant digits:** refers to the mathematical accuracy of a value.
- **Single-precision floating-point number:** stored in a `float`.
- **Standard arithmetic operators:** used to perform calculations with values in your applications.
- **Standard input device:** normally is the keyboard.
- **`String`:** a built-in Java class that provides you with the means for storing and manipulating character strings.
- **Strongly typed language:** a language in which all variables must be declared before they can be used.
- **Symbolic constant:** a named constant.
- **Token:** a unit of data. The `Scanner` class separates input into tokens.
- **Type casting:** forces a value of one data type to be used as a value of another type.
- **Type conversion:** the process of converting one data type to another.
- **Type-ahead buffer:** the keyboard buffer.
- **Type-wrapper classes:** classes contained in the `java.lang` package; include methods that can process primitive-type values.
- **Unary cast operator:** a more complete name for the cast operator that performs explicit conversions.
- **Unary operator:** uses only one operand.
- **Unicode:** A character encoding scheme that encompasses all symbols in all languages on Earth.
- **Unifying type:** a single data type to which all operands in an expression are converted.
- **Uninitialized variable:** a variable that has not been assigned a value.
- **Unnamed constant:** a number or string that is not held in a variable.
- **Variable:** a named memory location that you can use to store a value.
- **Variable declaration:** a statement that reserves a named memory location.