JAVA™ PROGRAMMING

Chapter 2: Using Data

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

Objectives (cont'd.)

- Use the `JOptionPane` class to accept GUI input
- Perform arithmetic
- Understand type conversion

Declaring and Using Constants and Variables

- **Constant**
  - Cannot be changed while program is running
- **Literal constant**
  - Value taken literally at each use
- **Numeric constant**
  - As opposed to a literal constant
- **Unnamed constant**
  - No identifier is associated with it

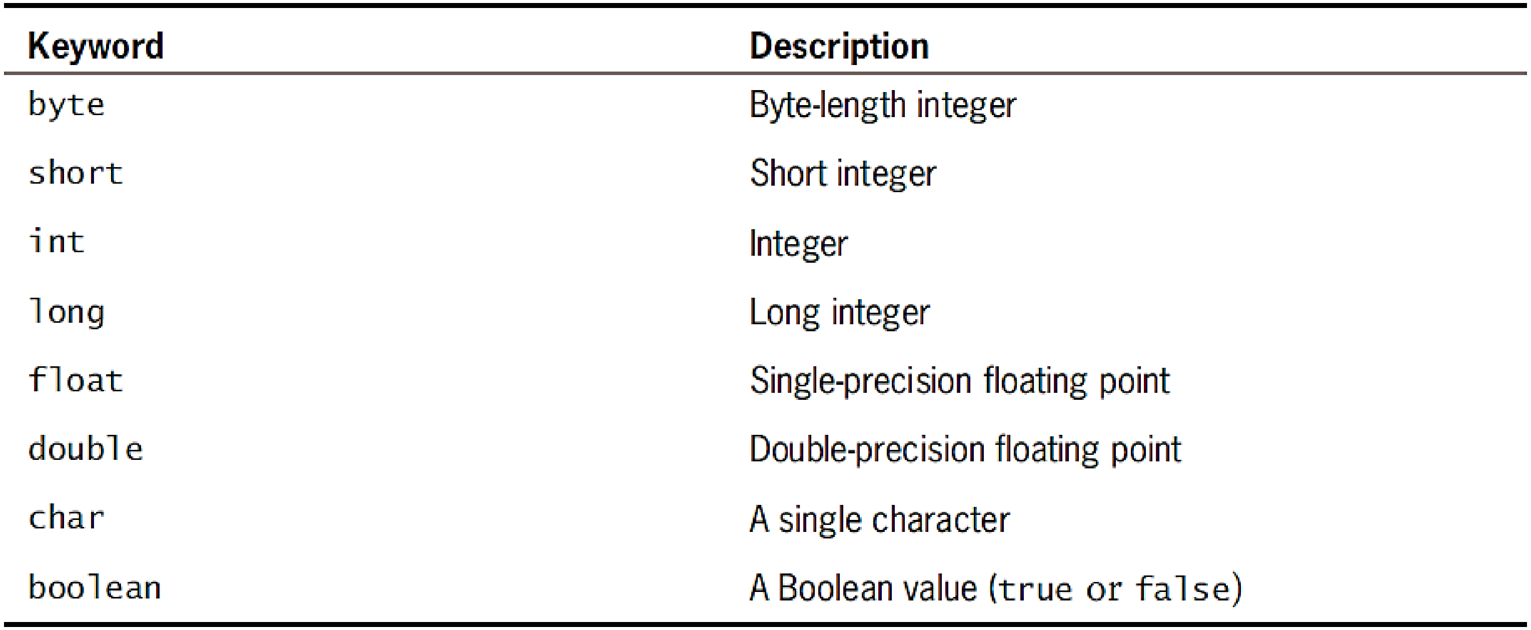
Declaring and Using Constants and Variables (cont'd.)

- **Variable**
  - A named memory location
  - Used to store a value
  - Can hold only one value at a time
  - Its value can change
- **Data type**
  - A type of data that can be stored
  - How much memory an item occupies
  - What types of operations can be performed on data

Declaring and Using Constants and Variables (cont'd.)

- **Primitive type**
  - A simple data type
- **Reference types**
  - More complex data types

Declaring and Using Constants and Variables (cont'd.)



Keyword	Description
`byte`	Byte-length integer
`short`	Short integer
`int`	Integer
`long`	Long integer
`float`	Single-precision floating point
`double`	Double-precision floating point
`char`	A single character
`boolean`	A Boolean value (`true` or `false`)

Table 2-1 Java primitive data types

Declaring Variables

- Name variables
  - Use naming rules for legal class identifiers
- **Variable declaration**
  - A statement that reserves a named memory location
  - Includes:
    - Data type
    - Identifier
    - Optional assignment operator and assigned value
    - Ending semicolon

Declaring Variables (cont’d.)

- **Assignment operator**
  - The equal sign (=)
  - The value to the right is assigned to the variable on the left
- **Initialization**
  - An assignment made when declaring a variable
- **Assignment**
  - An assignment made after a variable is declared
- **Associativity**
  - The order in which operands are used with operators

Declaring Variables (cont'd.)

- Declare multiple variables of the same type in separate statements on different lines

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

- When declaring variables of different types, you must use a separate statement for each type

Declaring Named Constants

- A **named constant**:
  - Should not change during program execution
  - Has a data type, name, and value
  - Has a data type preceded by the keyword **`final`**
  - Can be assigned a value only once
  - Conventionally is given identifiers using all uppercase letters

Declaring Named Constants (cont'd.)

- Reasons for using named constants:
  - Make programs easier to read and understand
  - Enable you to change a value at one location within a program
  - Reduce typographical errors
  - Stand out as separate from variables

The Scope of Variables and Constants

- **Scope**
  - The area in which a data item is visible to a program, and in which you can refer to it using its simple identifier
- A variable or constant is in scope from the point it is declared
  - Until the end of the **block of code** where the declaration lies

Concatenating Strings to Variables and Constants

- `print()` or `println()` statement
  - Use alone or in combination with a `String`
- **Concatenated**
  - A numeric variable is concatenated to a `String` using the plus sign
  - The entire expression becomes a `String`
- The `println()` method can accept a number or `String`

Concatenating Strings to Variables and Constants (cont'd.)

- Use a dialog box to display values

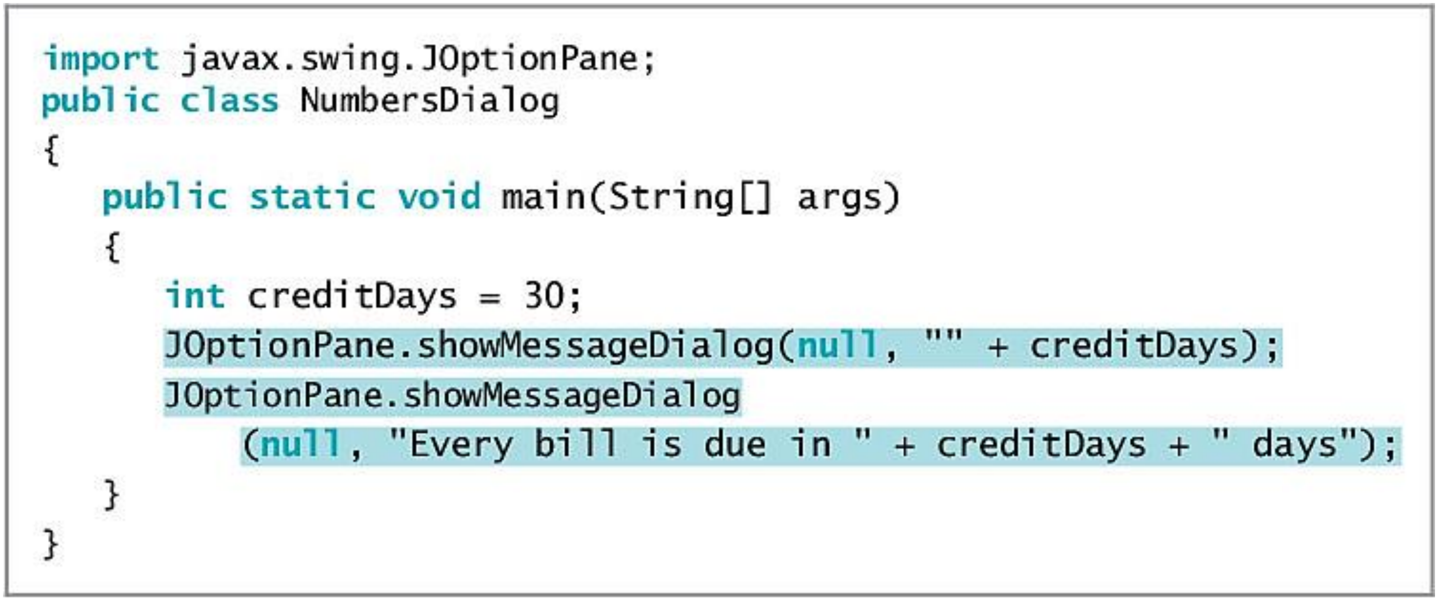
  `JOptionPane.showMessageDialog()`

  – Does not accept a single numeric variable

- **Null `String`**

  – An empty string: `""`

Concatenating Strings to Variables and Constants (cont'd.)



```
import javax.swing.JOptionPane;
public class NumbersDialog
{
   public static void main(String[] args)
   {
      int creditDays = 30;
      JOptionPane.showMessageDialog(null, "" + creditDays);
      JOptionPane.showMessageDialog
         (null, "Every bill is due in " + creditDays + " days");
   }
}
```

Figure 2-3 NumbersDialog class

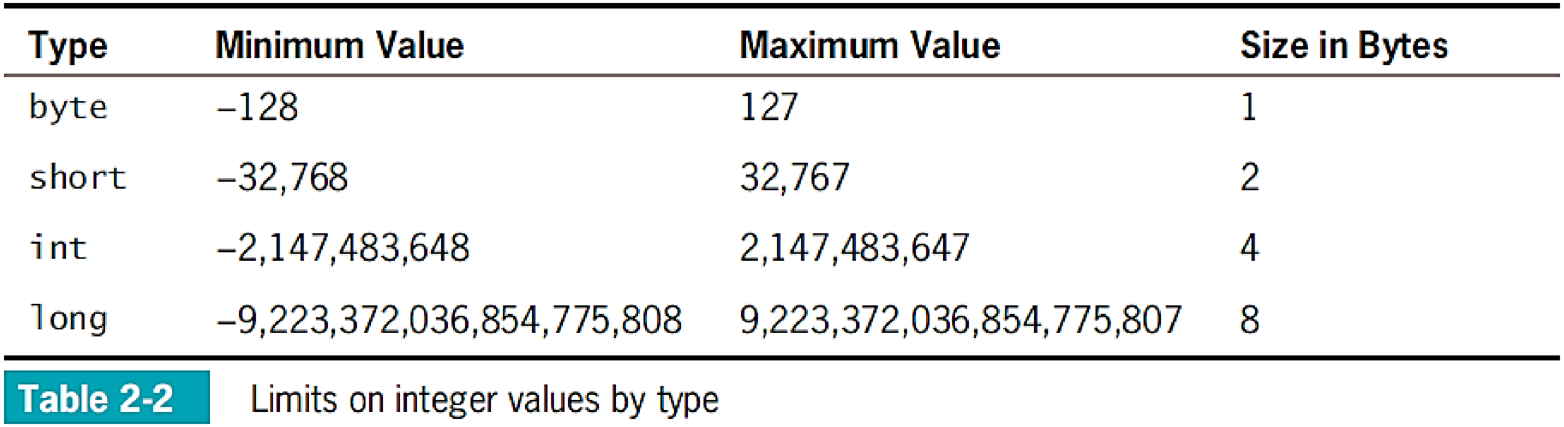
Pitfall: Forgetting That a Variable Holds One Value at a Time

- Each constant can hold only one value for the duration of the program
- Switch values of two variables
  - Use a third variable

Learning About Integer Data Types

- **`int`** data type
  - Stores an **integer**, or whole number
  - Value from –2,147,483,648 to +2,147,483,647
- Variations of the integer type
  - **`byte`**
  - **`short`**
  - **`long`**
- Choose appropriate types for variables

Learning About Integer Data Types (cont'd.)



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

Table 2-2 Limits on integer values by type

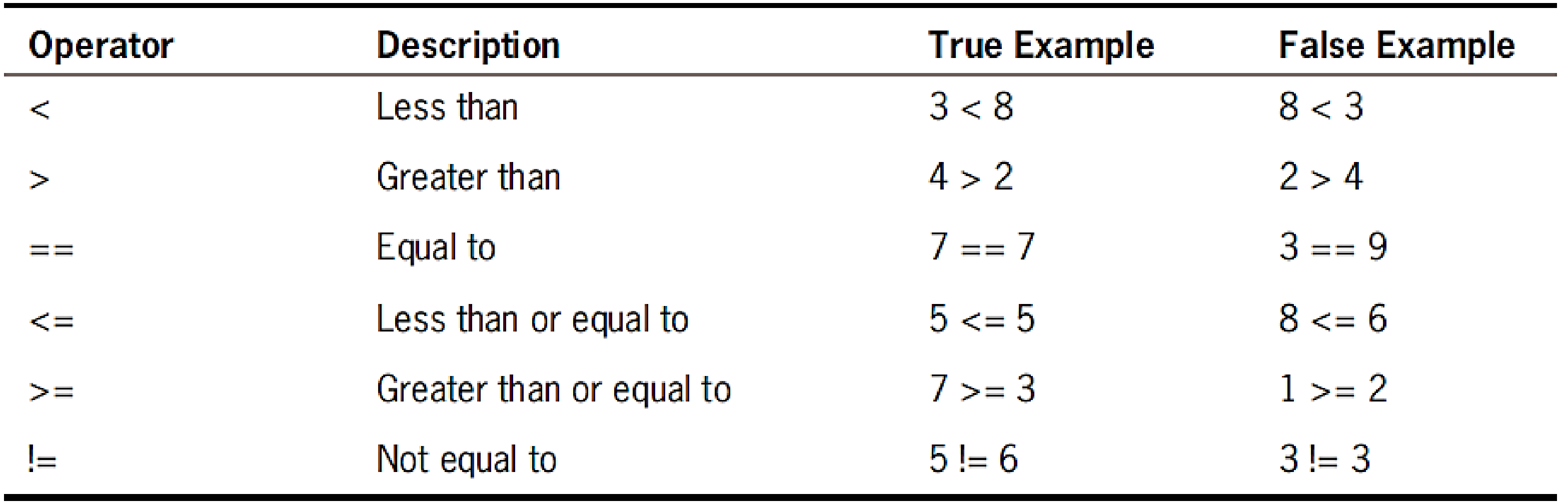
Using the `boolean` Data Type

- Boolean logic
  - Based on true-or-false comparisons
- **`boolean` variable**
  - Can hold only one of two values
  - `true` or `false`

```
boolean isItPayday = false;
```

- **Relational operator (comparison operator)**
  - Compares two items

Using the boolean Data Type (cont'd.)



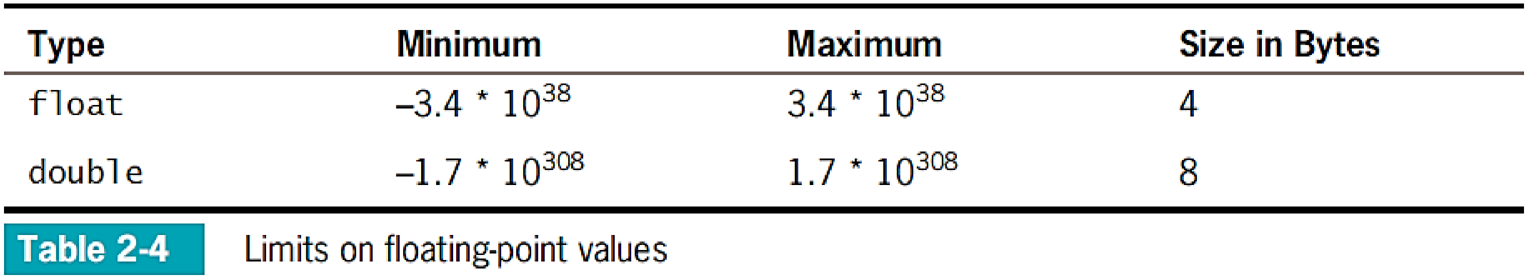
Operator	Description	True Example	False Example
<	Less than	3 < 8	8 < 3
>	Greater than	4 > 2	2 > 4
==	Equal to	7 == 7	3 == 9
<=	Less than or equal to	5 <= 5	8 <= 6
>=	Greater than or equal to	7 >= 3	1 >= 2
!=	Not equal to	5 != 6	3 != 3

Table 2-3 Relational operators

Learning About Floating-Point Data Types

- **Floating-point** number
  - Contains decimal positions
- Floating-point data types
  - `float`
  - `double`
- **Significant digits**
  - Refers to mathematical accuracy

Learning About Floating-Point Data Types (cont'd.)



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

Table 2-4 Limits on floating-point values

Using the `char` Data Type

- **`char`** data type
  - Holds any single character
- Place constant character values within single quotation marks

```
char myMiddleInitial = 'M';
```

- **`String`**
  - A built-in class
  - Stores and manipulates character strings
  - `String` constants are written between double quotation marks

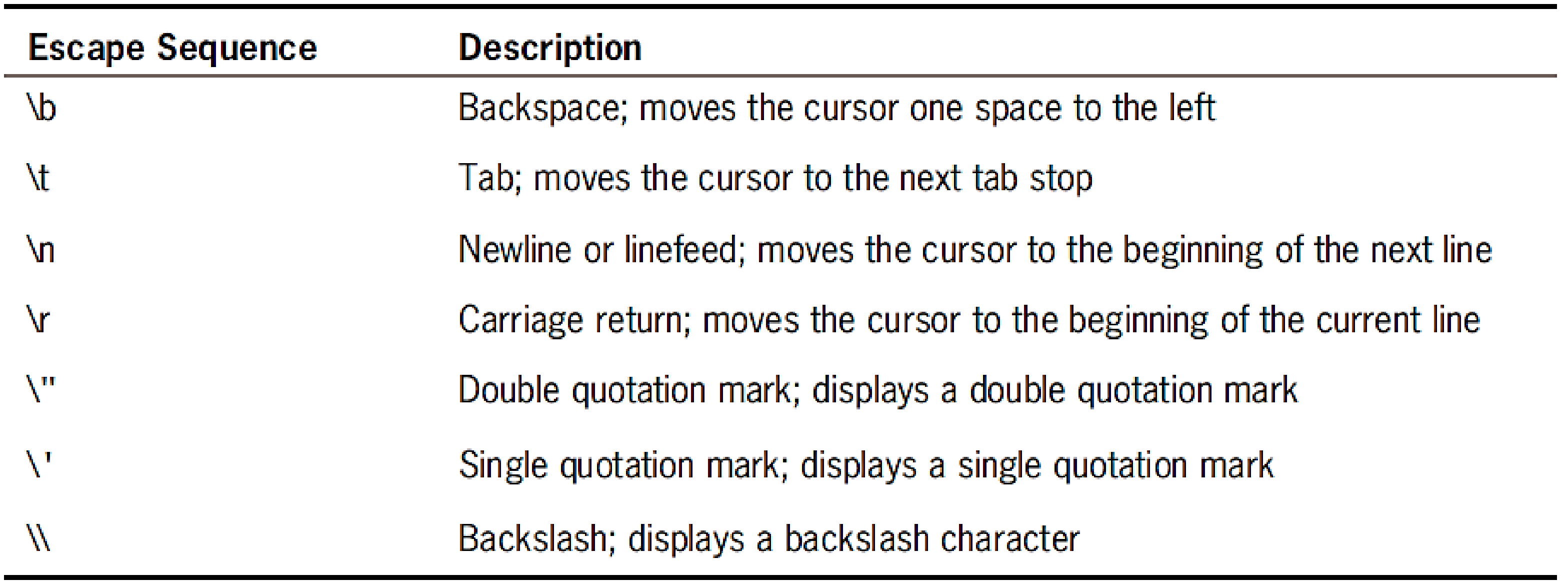
Using the `char` Data Type (cont’d.)

- **Escape sequence**
  - Begins with a backslash followed by a character
  - Represents a single nonprinting character

```
char aNewLine = '\n';
```

- To produce console output on multiple lines in the command window, use one of these options:
  - Use the newline escape sequence
  - Use the `println()` method multiple times

Using the `char` Data Type (cont'd.)



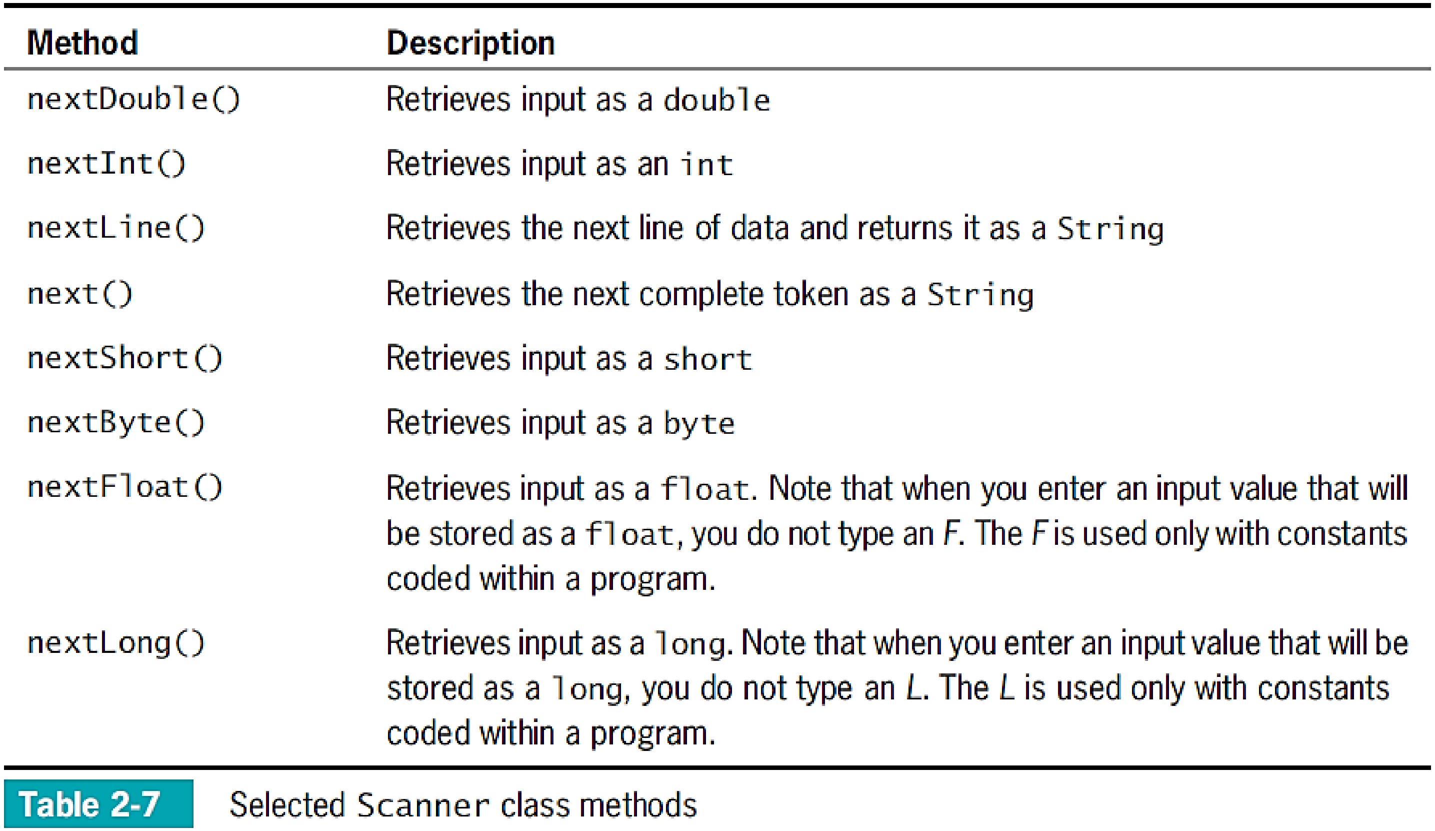
Escape Sequence	Description
\b	Backspace; moves the cursor one space to the left
\t	Tab; moves the cursor to the next tab stop
\n	Newline or linefeed; moves the cursor to the beginning of the next line
\r	Carriage return; moves the cursor to the beginning of the current line
\"	Double quotation mark; displays a double quotation mark
\'	Single quotation mark; displays a single quotation mark
\\\\	Backslash; displays a backslash character

Table 2-6 Common escape sequences

Using the `Scanner` Class to Accept Keyboard Input

- `System.in` object
  - **Standard input device**
  - Normally the keyboard
  - Access using the `Scanner` class
- `Scanner` object
  - Breaks input into units called **tokens**

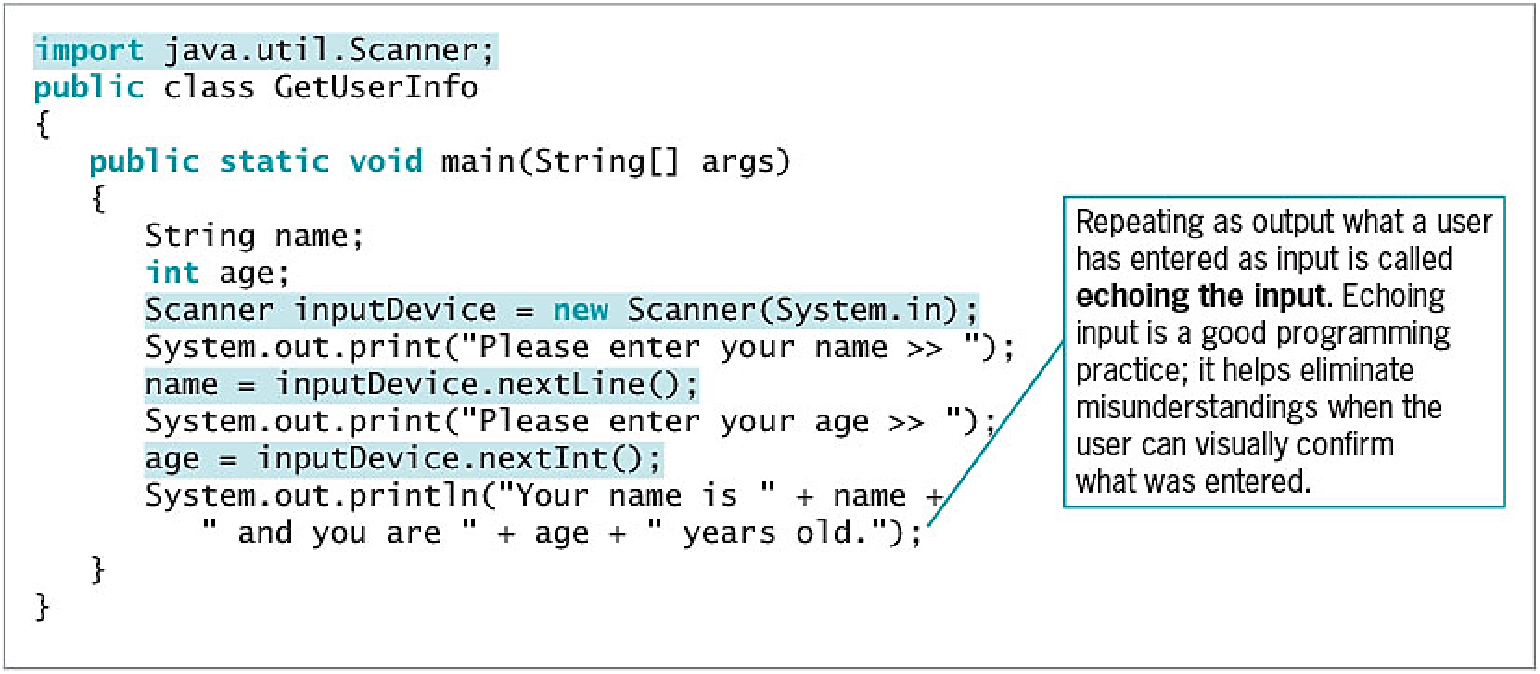
Using the `Scanner` Class to Accept Keyboard Input (cont'd.)



Method	Description
`nextDouble()`	Retrieves input as a `double`
`nextInt()`	Retrieves input as an `int`
`nextLine()`	Retrieves the next line of data and returns it as a `String`
`next()`	Retrieves the next complete token as a `String`
`nextShort()`	Retrieves input as a `short`
`nextByte()`	Retrieves input as a `byte`
`nextFloat()`	Retrieves input as a `float`. Note that when you enter an input value that will be stored as a `float`, you do not type an *F*. The *F* is used only with constants coded within a program.
`nextLong()`	Retrieves input as a `long`. Note that when you enter an input value that will be stored as a `long`, you do not type an *L*. The *L* is used only with constants coded within a program.

Table 2-7 Selected `Scanner` class methods

Using the Scanner Class to Accept Keyboard Input (cont'd.)



```
import java.util.Scanner;
public class GetUserInfo
{
   public static void main(String[] args)
   {
      String name;
      int age;
      Scanner inputDevice = new Scanner(System.in);
      System.out.print("Please enter your name >> ");
      name = inputDevice.nextLine();
      System.out.print("Please enter your age >> ");
      age = inputDevice.nextInt();
      System.out.println("Your name is " + name +
         " and you are " + age + " years old.");
   }
}
```

Repeating as output what a user has entered as input is called **echoing the input**. Echoing input is a good programming practice; it helps eliminate misunderstandings when the user can visually confirm what was entered.

Figure 2-17 The GetUserInfo class

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

- There is a problem when using one numeric `Scanner` class retrieval method or `next()` method before using the `nextLine()` method
- **Keyboard buffer**
  - Location in memory that stores all keystrokes, including Enter
- To avoid issues, add an extra `nextLine()` method call to retrieve the abandoned Enter key character after numeric or `next()` inputs

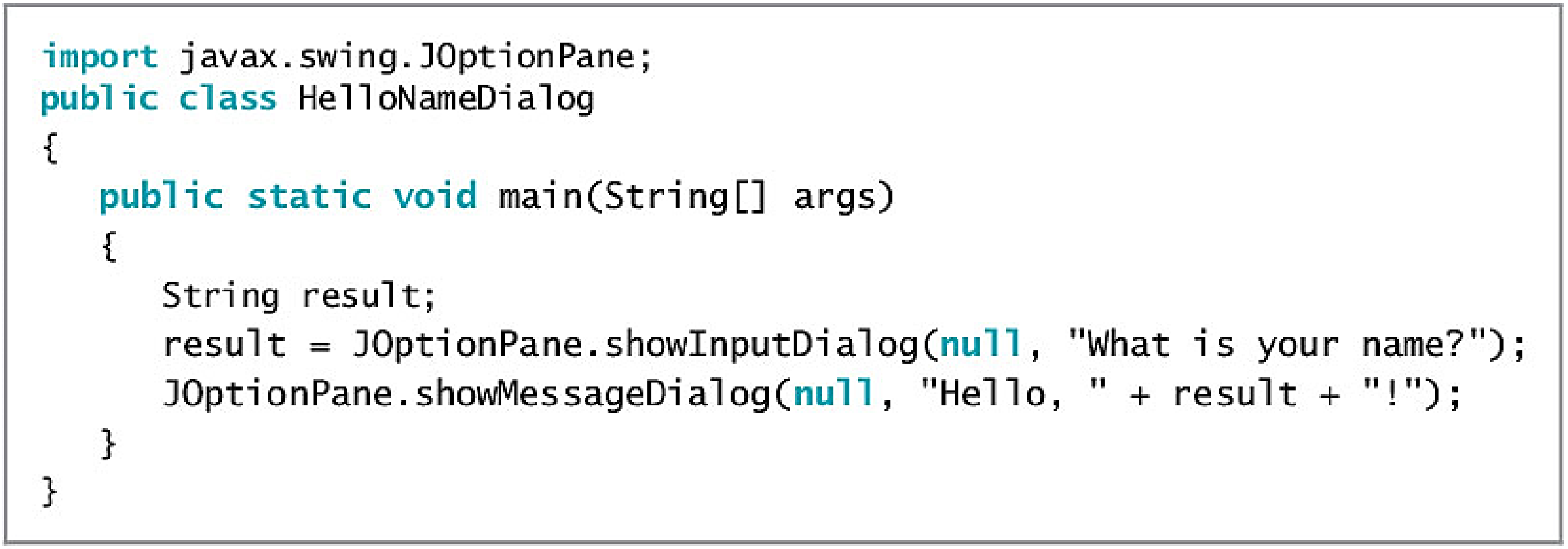
Using the `JOptionPane` Class to Accept GUI Input

- Dialog boxes used to accept user input:
  - Input dialog box
  - Confirm dialog box

Using Input Dialog Boxes

- **Input dialog box**
  - Asks a question
  - Provides a text field in which the user can enter a response
- **`showInputDialog()` method**
  - Six overloaded versions
  - Returns a `String` representing a user's response
- **Prompt**
  - A message requesting user input

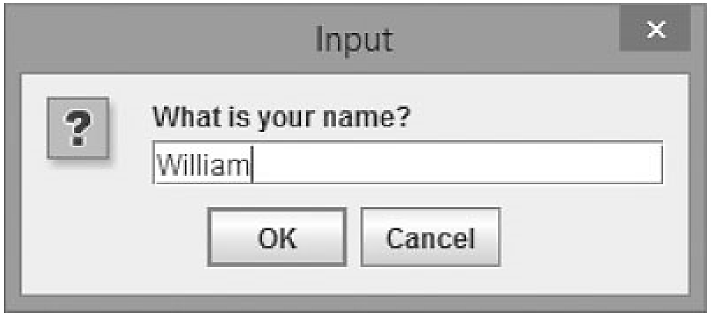
Using Input Dialog Boxes (cont'd.)



```
import javax.swing.JOptionPane;
public class HelloNameDialog
{
   public static void main(String[] args)
   {
      String result;
      result = JOptionPane.showInputDialog(null, "What is your name?");
      JOptionPane.showMessageDialog(null, "Hello, " + result + "!");
   }
}
```

Figure 2-26 The HelloNameDialog class

Using Input Dialog Boxes (cont'd.)



What is your name?

William

Figure 2-27 Input dialog box of the `HelloNameDialog` application

Using Input Dialog Boxes (cont'd.)

- `showInputDialog()`
  - One version requires four arguments:
    - Parent component
    - Message
    - Title
    - Type of dialog box
- Convert `String` to `int` or `double`
  - Use methods from the built-in Java classes `Integer` and `Double`

Using Input Dialog Boxes (cont'd.)

- **Type-wrapper classes**
  - Each primitive type has a corresponding class contained in the `java.lang` package
  - Include methods to process primitive type values

```
Integer.parseInt()
Double.parseDouble()
```

Using Confirm Dialog Boxes

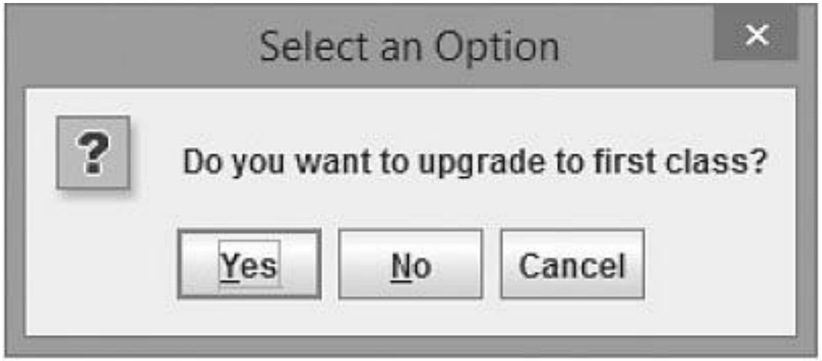
- **Confirm dialog box**
  - Displays the options Yes, No, and Cancel
- **`showConfirmDialog()` method** in `JOptionPane` class
  - Four overloaded versions are available
  - Returns integer containing either:

```
JOptionPane.YES_OPTION
JOptionPane.NO_OPTION
JOptionPane.CANCEL_OPTION
```

Using Confirm Dialog Boxes (cont’d.)

- You can create a confirm dialog box with five arguments:
  - Parent component
  - Prompt message
  - Title
  - Integer that indicates which option button to show
  - Integer that describes the kind of dialog box

Using Confirm Dialog Boxes (cont'd.)



Select an Option

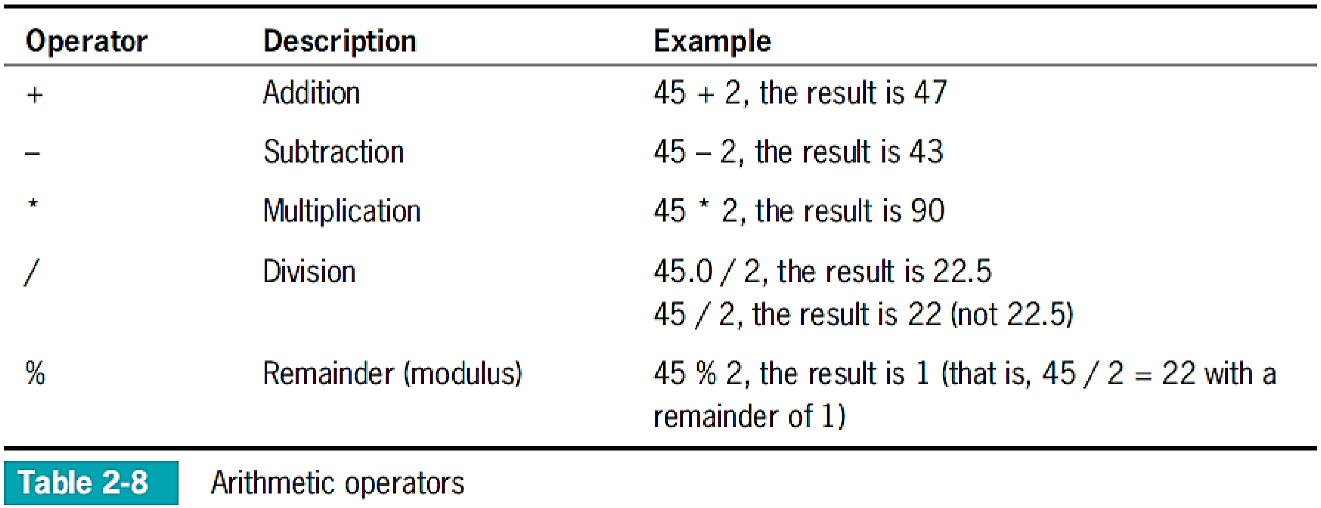
Do you want to upgrade to first class?

Figure 2-33 The confirm dialog box displayed by the `AirlineDialog` application

Performing Arithmetic

- **Standard arithmetic operators**
  - Perform calculations with values in programs
- **Operand**
  - A value used on either side of an operator
- **Integer division**
  - Involves integer constants or integer variables
  - The result is an integer
  - Any fractional part of the result is lost

Performing Arithmetic (cont’d.)



Operator	Description	Example
+	Addition	45 + 2, the result is 47
–	Subtraction	45 – 2, the result is 43
*	Multiplication	45 * 2, the result is 90
/	Division	45.0 / 2, the result is 22.5 45 / 2, the result is 22 (not 22.5)
%	Remainder (modulus)	45 % 2, the result is 1 (that is, 45 / 2 = 22 with a remainder of 1)

Table 2-8 Arithmetic operators

Associativity and Precedence

- **Operator precedence**
  - The rules for the order in which parts of mathematical expressions are evaluated
  - First multiplication, division, and remainder (modulus), then addition or subtraction

Writing Arithmetic Statements Efficiently

- Avoid unnecessary repetition of arithmetic statements
- Example of inefficient calculation:

```
stateWithholding = hours * rate * STATE_RATE;
federalWithholding = hours * rate * FED_RATE;
```

- Example of efficient calculation:

```
grossPay = hours * rate;
stateWithholding = grossPay * STATE_RATE;
federalWithholding = grossPay * FED_RATE;
```

Pitfall: Not Understanding Imprecision in Floating-Point Numbers

- Integer values are exact
  - But floating-point numbers frequently are only approximations
- Imprecision leads to several problems
  - Floating-point output might not look like what you expect or want
  - Comparisons with floating-point numbers might not be what you expect or want

Understanding Type Conversion

- Arithmetic with variables or constants of the same type
  - The result of arithmetic retains the same type
- Arithmetic operations with operands of unlike types
  - Java chooses the unifying type for the result
- **Unifying type**
  - The type to which all operands in an expression are converted for compatibility

Automatic Type Conversion

- Automatically converts nonconforming operands to the unifying type
- Order for establishing unifying types between two variables (highest to lowest):
  1. `double`
  2. `float`
  3. `long`
  4. `int`

Explicit Type Conversions

- **Type casting**
  - Forces a value of one data type to be used as a value of another data type
- **Cast operator**
  - Place desired result type in parentheses
  - Using a cast operator is an **explicit conversion**
- You do not need to perform a cast when assigning a value to a higher unifying type

You Do It

- Declaring and Using a Variable
- Working with Integers
- Working with the `char` Data Type
- Accepting User Input
- Using Arithmetic Operators
- Implicit and Explicit Casting

Don’t Do It

- Don’t attempt to assign a literal constant floating-point number
- Don’t forget precedence rules
- Don’t forget that integer division results in an integer
- Don’t attempt to assign a constant decimal value to an integer using a leading 0
- Don’t use a single equal sign (=) in a Boolean comparison for equality
- Don’t try to store a string of characters in a `char` variable

Don’t Do It (cont’d.)

- Don’t forget that when a `String` and a numeric value are concatenated, the resulting expression is a string
- Don’t forget to consume the Enter key after numeric input using the `Scanner` class when a `nextLine()` method call follows
- Don’t forget to use the appropriate import statement when using the `Scanner` or `JOptionPane` class

Summary

- Variables
  - Named memory locations
- Primitive data types
- Standard arithmetic operators for integers:

  `+, _, *, /,` and `%`
- Boolean type
  - `true` or `false` value
- Relational operators:

  `>, <, ==, >=, <=,` and `!=`

Summary (cont'd.)

- Floating-point data types
  - `float`
  - `double`
- `char` data type
- `Scanner` class
  - Access keyboard input
- `JOptionPane`
  - Confirm dialog box
  - Input dialog box