# NCP2103: Object-Oriented Programming (Java Programming)

# Data Types

Module 2

**Errol John M. Antonio**
Assistant Professor
Computer Engineering Department
University of the East – Manila Campus

# Learning Objectives

- Use constants and variables, integer data type, `boolean` data type, floating-point data types and `char` data type
- Display data and perform arithmetic
- Understand numeric type conversion
- Use the `Scanner` class to accept keyboard input
- Use the `JOptionPane` class for GUI input

# Module Outline

I. Using Constants and Variables
II. Learning about Integer Data Types
III. Using the Boolean Data Type
IV. Learning about Floating-Point Data Types
V. Using the Char Data Type
VI. Displaying Data and Performing Arithmetic
VII. Performing Arithmetic
VIII. Understanding Numeric-Type Conversion

# Module Outline

IX. Using the `Scanner` Class for Keyboard Input

X. Using the JOptionPane Class for GUI Input

# I: Using Constants and Variables

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

# Using Constants and Variables

- **Variable**
  - Named memory location
  - Use to store value
  - Can hold only one value at a time
  - Value can change
- **Data type**
  - Type of data that can be stored
  - How much memory item occupies
  - What types of operations can be performed on data

# Using Constants and Variable

- **Primitive type**
  - Simple data type
- **Reference types**
  - More complex data types

# Using Constants and Variables

| 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
  - Using naming rules for legal class identifiers
- Variable declaration
  - Statement that reserves named memory location
  - Includes:
    - Data type
    - Identifier
    - Optional assignment operator and assigned value
    - Ending semicolon

# Declaring Variable

- **Assignment operator**
  - Equal sign (=)
  - Value to right assigned to variable on left
- **Initialization**
  - Assignment made when declaring variable
- **Assignment**
  - Assignment made after variable declared
- **Associativity**
  - Order in which operands used with operators

# Declaring Variables

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

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

- Declare variables of different types
  - Must use separate statement for each type

# Declaring Named Constants

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

# Declaring Named Constants

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

# Pitfall: Forgetting that a Variable Holds One Value at a Time

- Each constant can hold **only one value**
  - For duration of program
- Switch values of two variables
  - Use third variable

# TWO TRUTHS AND A LIE

1. A variable is a named memory location that you can use to store a value; it can hold only one value at a time, but the value it holds can change.
2. An item's data type determines what legal identifiers can be used to describe variables and whether the variables can occupy memory.
3. A variable declaration is a statement that reserves a named memory location and includes a data type, an identifier, an optional assignment operator and assigned value, and an ending semicolon.

# II: Learning About Integer Data Types

- Type **`int`**
  - Stores integers, or whole numbers
  - 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

| 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

# TWO TRUTHS AND A LIE

1. A variable of type int can hold any whole number value from approximately negative two billion to positive two billion.
2. When you assign a value to an int variable, you do not type any commas; you type only digits and an optional plus or minus sign to indicate a positive or negative integer.
3. You can use the data types byte or short to hold larger values than can be accommodated by an int.

# III: 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

| 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

- Boolean expressions are more meaningful when a variable is used for one or both of the operands in a comparison.

```
boolean isOvertimePay = (hours > 40);
boolean isTaxBracketHigh = (income > HIGH_CUTOFF);
boolean isFirstScoreHigher = (score1 > score2);
```

# TWO TRUTHS AND A LIE

1. A Boolean variable can hold only one of two values—true or false.
2. Java supports six relational operators that are used to make comparisons: =, <, >, =<, =>, and =!.
3. An expression that contains a relational operator has a Boolean value.

# IV: 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 Type

| 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

# TWO TRUTHS AND A LIE

1. Java supports two floating-point data types: float and double. The double data type requires more memory and can hold more significant digits.
2. A floating-point constant, such as 5.6, is a float by default.
3. As with integers, you can perform the mathematical operations of addition, subtraction, multiplication, and division with floating-point numbers.

# V: Working with the `char` Data Type

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

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

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

# Working with the `char` Data Type

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

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

- Produce console output on multiple lines in command window
  - Use newline escape sequence
  - Use `println()` method multiple times

# Working with the `char` Data Type

| 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

# TWO TRUTHS AND A LIE

1. You use the char data type to hold any single character; you place constant character values within single quotation marks.
2. To store a string of characters, you use a data structure called a Text; string constants are written between parentheses
3. An escape sequence always begins with a backslash followed by a character; the pair represents a single character.

# VI: Displaying Data and Performing Arithmetic

- **`print()`** or **`println()`** statement
  - Alone or in combination with string
- Concatenated
  - Numeric variable concatenated to `String` using plus sign
  - Entire expression becomes `String`
- **`println()`** method can accept number or **`String`**

# Displaying Data and Performing Arithmetic

- Use dialog box to display values

  **`JOptionPane.showMessageDialog()`**

  - Does not accept single numeric variable

- Null **`String`**

  - Empty string: “”

# Displaying Data and Performing Arithmetic



```
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 2-5 NumbersDialog class

# VII: Performing Arithmetic

- **Arithmetic operators**
  - Perform calculations with values in programs
- **Operand**
  - Value used on either side of operator
- **Integer division**
  - Integer constants or integer variables
  - Result is integer
  - Fractional part of result lost

# Performing Arithmetic

| 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<br>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 2-7** Arithmetic operators

# Performing Arithmetic

- **Operator precedence**
  - Rules for order in which parts of mathematical expression are evaluated
  - First multiplication, division, and 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

# TWO TRUTHS AND A LIE

1. The arithmetic operators are examples of unary operators, which are so named because they perform one operation at a time.
2. In Java, operator precedence dictates that multiplication, division, and remainder always take place prior to addition or subtraction in an expression.
3. Floating-point arithmetic might produce imprecise results.

# VIII: Understanding Numeric-Type Conversion

- Arithmetic with variables or constants of same type
  - Result of arithmetic retains same type
  - Example: **`int = int / int`**,
    **`double = double - double`**
- Arithmetic operations with operands of unlike types
  - Java chooses unifying type for result

# Understanding Numeric-Type Conversion

- **Automatic Type Conversion**
  - **Unifying type**
    - Type to which all operands in expression are converted for compatibility

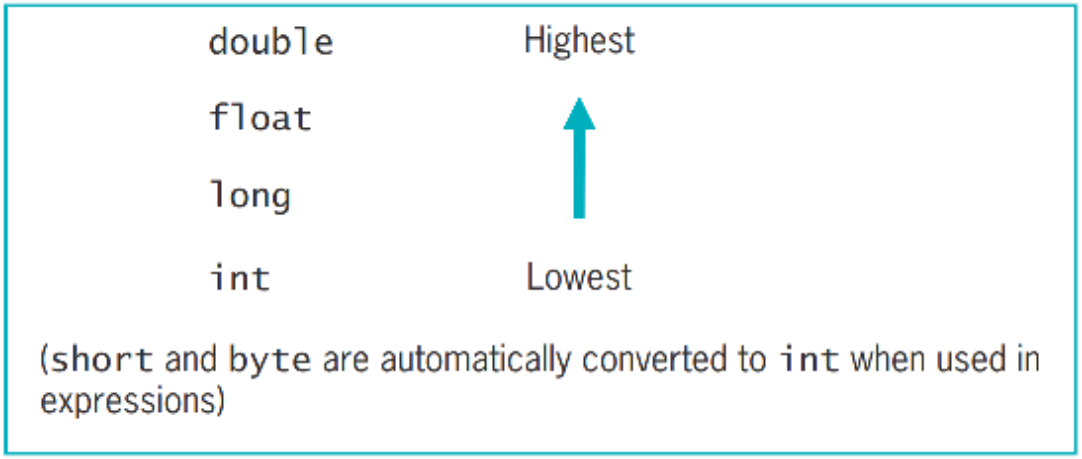



**Figure 2-41** Order for establishing unifying data types

# Understanding Numeric-Type Conversion

```
int hoursWorked = 37;
double payRate = 16.73;
double grossPay = hoursWorked * payRate;
```

- Is this a valid conversion?
  - Yes.

# Understanding Numeric-Type Conversion

```
int hoursWorked = 37;
double payRate = 16.73;
int grossPay = hoursWorked * payRate;
```

- Is this a valid conversion?
  - No. Why?
  - Java does not allow the loss of precision that occurs if you try to store the calculated double result in an int.

# Understanding Numeric-Type Conversion

- **Explicit Type Conversion**
  - Override the unifying type imposed by Java by performing a type cast.
- **Type casting**
  - Forces value of one data type to be used as value of another type
- **Cast operator**
  - Place desired result type in parentheses
  - Explicit conversion

# Understanding Numeric-Type Conversion

- Note: Do not need to perform cast when assigning value to higher unifying type

```
double bankBalance = 189.66;
float weeklyBudget = (float) (bankBalance / 4);
// weeklyBudget is 47.415, one-fourth of bankBalance

float myMoney = 47.82f;
int dollars = (int) myMoney;
// dollars is 47, the integer part of myMoney
```

# Understanding Numeric-Type Conversion

- It is easy to lose data when performing a cast. For example, the largest byte value is 127 and the largest int value is 2,147,483,647, so the following statements produce distorted results:

```
int anOkayInt = 200;
byte aBadByte = (byte)anOkayInt;
//aBadByte = -72
```

# TWO TRUTHS AND A LIE

1. The arithmetic operators are examples of unary operators, which are so named because they perform one operation at a time.
2. In Java, operator precedence dictates that multiplication, division, and remainder always take place prior to addition or subtraction in an expression.
3. Floating-point arithmetic might produce imprecise results.

# IX: Using the `Scanner` Class for Keyboard Input

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

# Using the `Scanner` Class for Keyboard Input

| 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 for Keyboard Input

```
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 2-10** The `GetUserInfo` class

# Using the `Scanner` Class for Keyboard Input

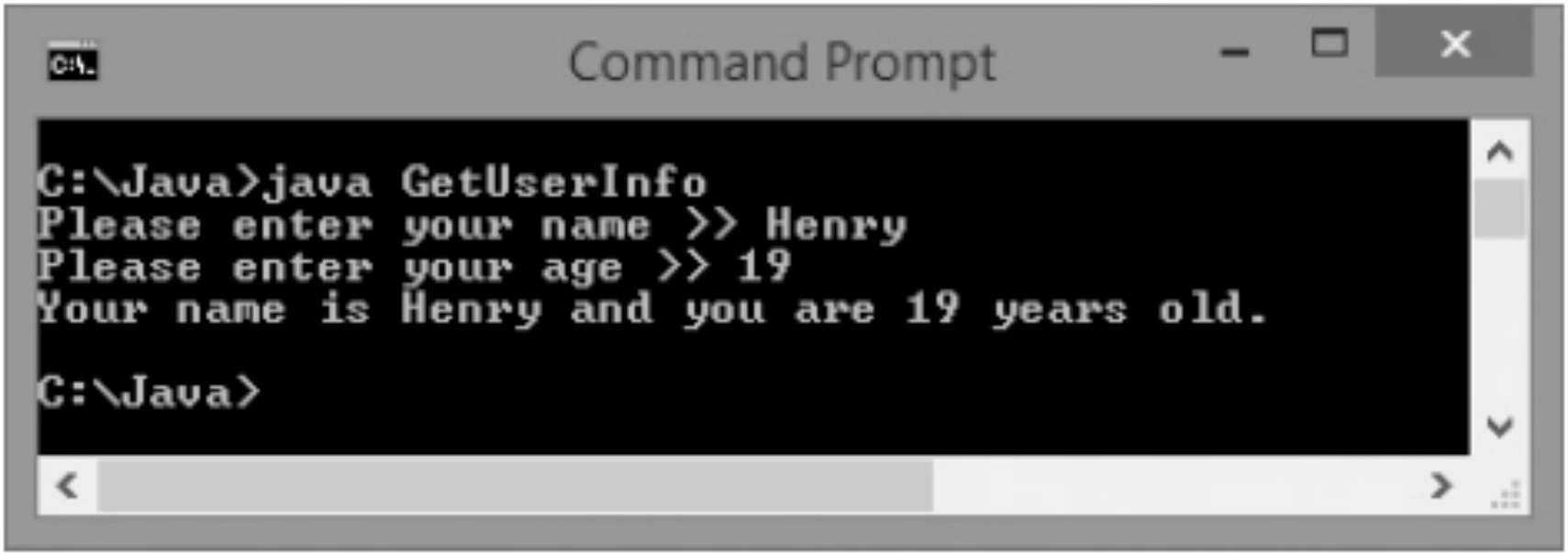



**Figure 2-18** Typical execution of the `GetUserInfo` program

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

```
import java.util.Scanner;
public class GetUserInfo2
{
   public static void main(String[] args)
   {
      String name;
      int age;
      Scanner inputDevice = new Scanner(System.in);
      System.out.print("Please enter your age >> ");
      age = inputDevice.nextInt();
      System.out.print("Please enter your name >> ");
      name = inputDevice.nextLine();
      System.out.println("Your name is " + name +
         " and you are " + age + " years old.");
   }
}
```

***Don't Do It***
If you accept numeric input prior to string input, the string input is ignored unless you take special action.

**Figure 2-19** The `GetUserInfo2` class

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


**Figure 2-20** Typical execution of the `GetUserInfo2` program

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

- Problem when using one numeric **`Scanner`** class retrieval method or **`next()`** method
  - Before using **`nextLine()`** method
- Keyboard buffer
  - When you type characters using the keyboard, they are stored temporarily in a location in memory called the keyboard buffer or the type-ahead buffer.
- After any numeric or **`next()`** input:
  - Add extra **`nextLine()`** method call
  - Will retrieve abandoned Enter key character

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

```
import java.util.Scanner;
public class GetUserInfo3
{
   public static void main(String[] args)
   {
      String name;
      int age;
      Scanner inputDevice = new Scanner(System.in);
      System.out.print("Please enter your age >> ");
      age = inputDevice.nextInt();
      inputDevice.nextLine();
      System.out.print("Please enter your name >> ");
      name = inputDevice.nextLine();
      System.out.println("Your name is " + name +
         " and you are " + age + " years old.");
   }
}
```

This statement gets the integer.

This statement consumes the Enter key that follows the integer.

This statement gets the name and discards the Enter key that follows the name.

**Figure 2-21** The `GetUserInfo3` class

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


**Figure 2-22** Typical execution of the `GetUserInfo3` program

# TWO TRUTHS AND A LIE

1. **`System.in`** refers to the standard input device, which normally is the keyboard.
2. **`System.in`** is more flexible than **`System.out`** because it can read all the basic.
3. When a user types data followed by the Enter key, the Enter key character is left in the keyboard buffer after Scanner class methods retrieve the other keystrokes.

# X: Using the `JOptionPane` Class for GUI Input

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

# Using Input Dialog Boxes

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

# Using Input Dialog Boxes

```
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 2-16** The HelloNameDialog class

# Using Input Dialog Boxes


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

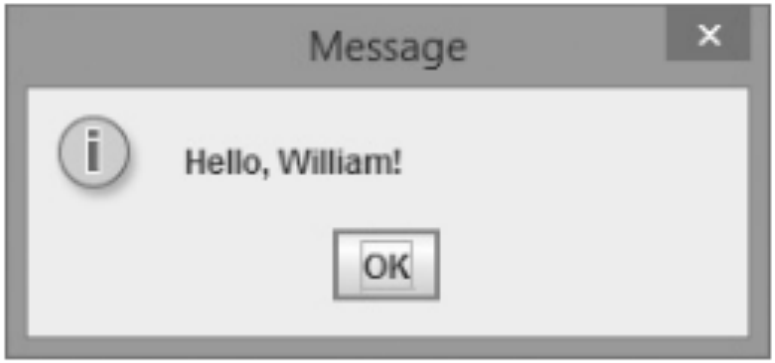



**Figure 2-28** Output of the `HelloNameDialog` application

# Using Input Dialog Boxes

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

# Using Input Dialog Boxes

```
JOptionPane.showInputDialog(null,
      "What is your area code?",
      "Area code information",
      JOptionPane.QUESTION_MESSAGE);
```

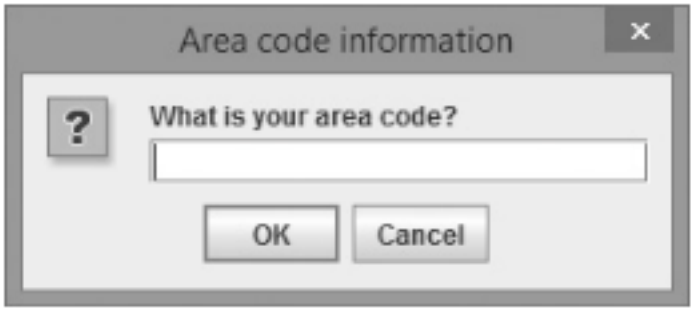



**Figure 2-29** An input dialog box with a `String` in the title bar and a question mark icon

# Using Input Dialog Boxes

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

    **`Integer.parseInt()`**

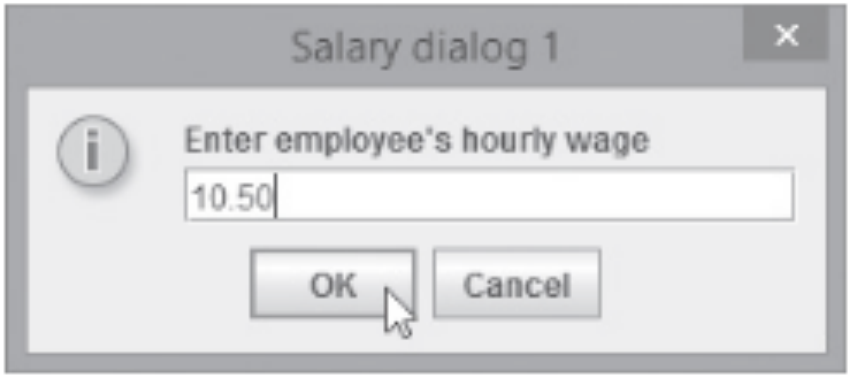
    **`Double.parseDouble()`**
  - Parsing a String converts it to its numeric equivalent.

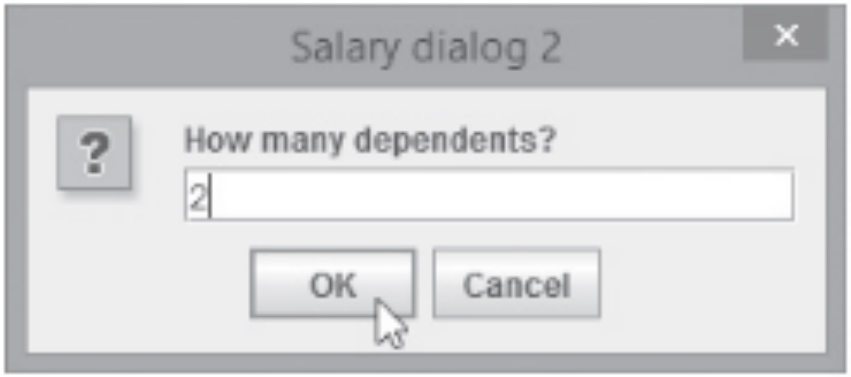
# Using Input Dialog Boxes

```
import javax.swing.JOptionPane;
public class SalaryDialog
{
   public static void main(String[] args)
   {
        String wageString, dependentsString;
        double wage, weeklyPay;
        int dependents;
        final double HOURS_IN_WEEK = 37.5;
        wageString = JOptionPane.showInputDialog(null,
           "Enter employee's hourly wage", "Salary dialog 1",
           JOptionPane.INFORMATION_MESSAGE);
        weeklyPay = Double.parseDouble(wageString) *
           HOURS_IN_WEEK;
        dependentsString = JOptionPane.showInputDialog(null,
           "How many dependents?", "Salary dialog 2",
           JOptionPane.QUESTION_MESSAGE);
        dependents = Integer.parseInt(dependentsString);
        JOptionPane.showMessageDialog(null, "Weekly salary is $" +
           weeklyPay + "\nDeductions will be made for " +
           dependents + " dependents");
   }
}
```

**Figure 2-30** The SalaryDialog class

# Using Input Dialog Boxes



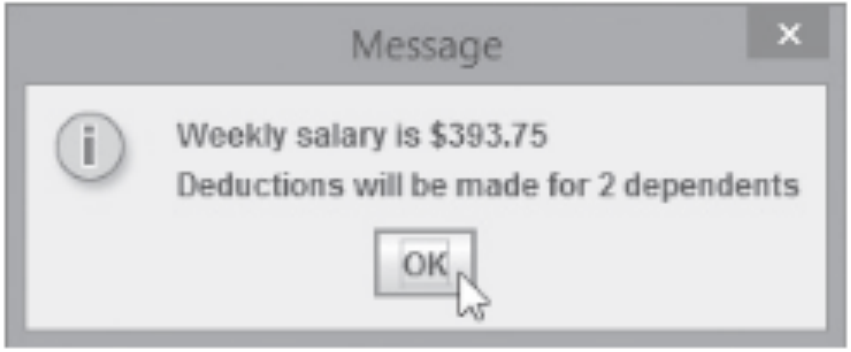




**Figure 2-31** Sample execution of the `SalaryDialog` application

# Using Confirm Dialog Boxes

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

    **`JOptionPane.YES_OPTION`**

    **`JOptionPane.NO_OPTION`**

    **`JOptionPane.CANCEL_OPTION`**

# Using Confirm Dialog Boxes

- Create confirm dialog box with five arguments
  - Parent component
  - Prompt message
  - Title
  - Integer that indicates which option button to show
  - Integer that describes kind of dialog box

# Using Confirm Dialog Boxes

```
import javax.swing.JOptionPane;
public class AirlineDialog
{
   public static void main(String[] args)
   {
      int selection;
      boolean isYes;
      selection = JOptionPane.showConfirmDialog(null,
         "Do you want to upgrade to first class?");
      isYes = (selection == JOptionPane.YES_OPTION);
      JOptionPane.showMessageDialog(null,
         "You responded " + isYes);
   }
}
```

Figure 2-32 The AirlineDialog class

# Using Confirm Dialog Boxes

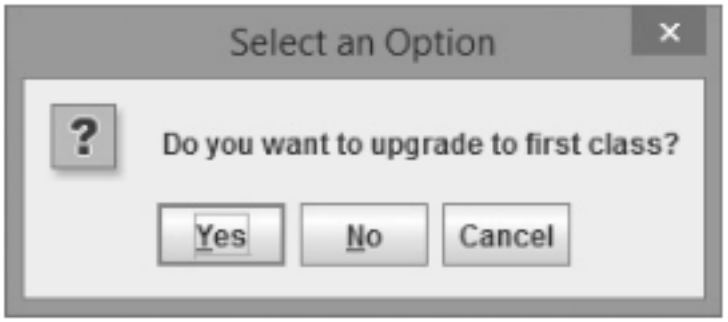



**Figure 2-33** The confirm dialog box displayed by the `AirlineDialog` application

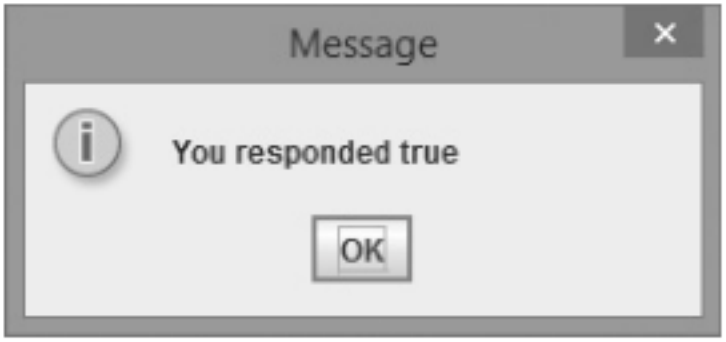



**Figure 2-34** Output of `AirlineDialog` application when user clicks Yes

# TWO TRUTHS AND A LIE

1. You can create an input dialog box using the showInputDialog() method; the method returns a String that represents a user's response.
2. You can use methods from the Java classes Integer and Double when you want to convert a dialog box's returned values to numbers.
3. A confirm dialog box can be created using the showConfirmDialog() method in the JOptionPane class; a confirm dialog box displays the options Accept, Reject, and Escape.

# You Do It

- Working with numeric values
- Accepting user data
- Performing arithmetic
- Experimenting with Java programs

# 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`

# Don't Do It

- 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

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

End of Module.

# REFERENCE:

Farrell, J. (2016). *Java Programming*. 8th Edition. Course Technology, Cengage Learning.