Java Programming: From Problem Analysis to Program Design, 5e

Chapter 3

Introduction to Objects and Input/Output

Chapter Objectives

- Learn about objects and reference variables
- Explore how to use predefined methods in a program
- Become familiar with the `class` `String`
- Learn how to use input and output dialog boxes in a program

Chapter Objectives (continued)

- Explore how to format the output of decimal numbers with the `String` method `format`
- Become familiar with file input and output

Object and Reference Variables

- Declare a reference variable of a class type
  - Allocate memory space for data
  - Instantiate an object of that class type
- Store the address of the object in a reference variable

Object and Reference Variables (continued)

```
int x;                          //Line 1
String str;                     //Line 2
x = 45;                         //Line 3
str = "Java Programming";       //Line 4
```

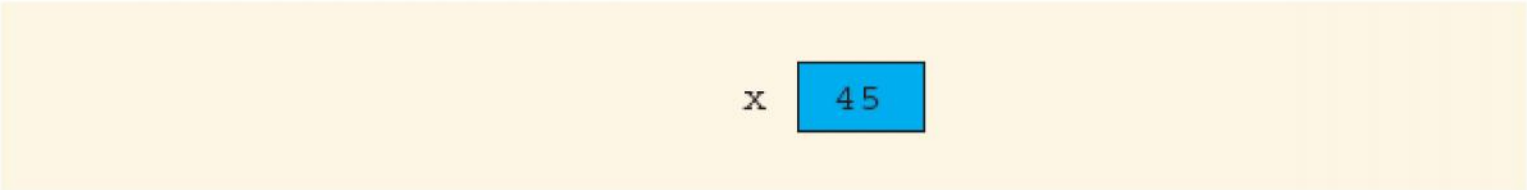

FIGURE 3-1 Variable x and its data

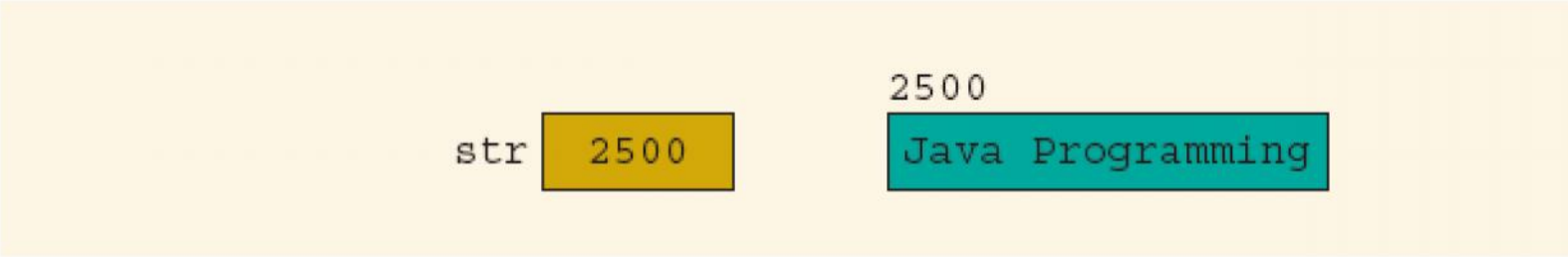

FIGURE 3-2 Variable str and the data it points to

Object and Reference Variables (continued)

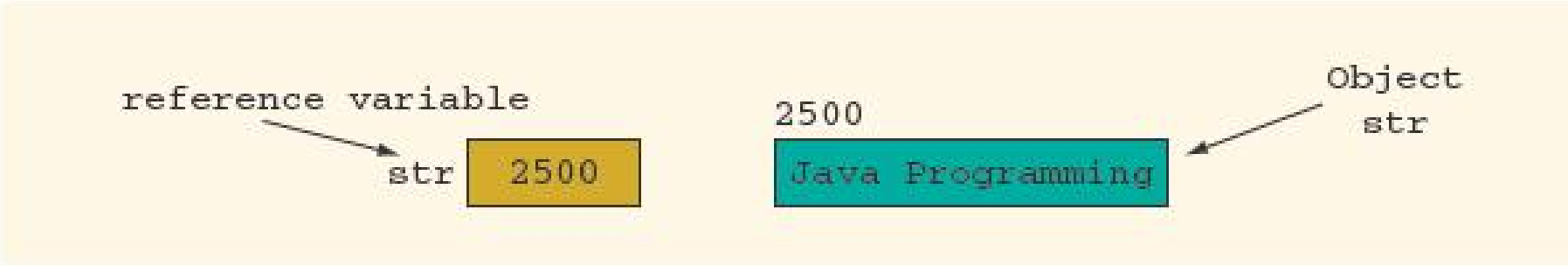

FIGURE 3-3 Variable `str` and object `str`

Object and Reference Variables (continued)

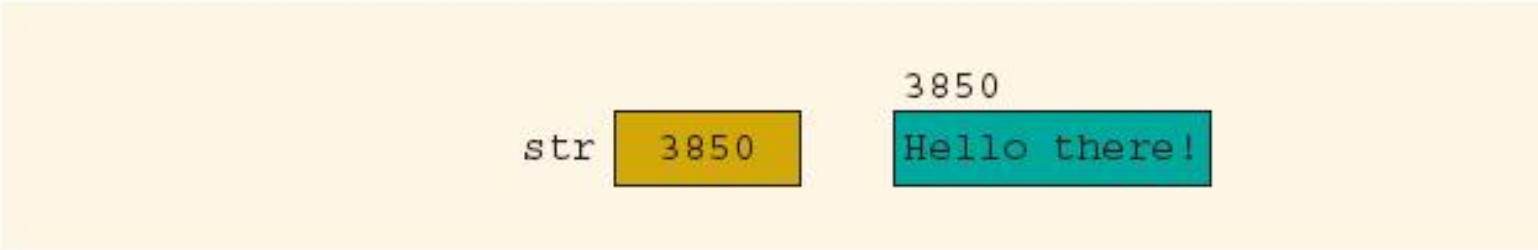

FIGURE 3-4 Variable `str`, its value, and the object `str`

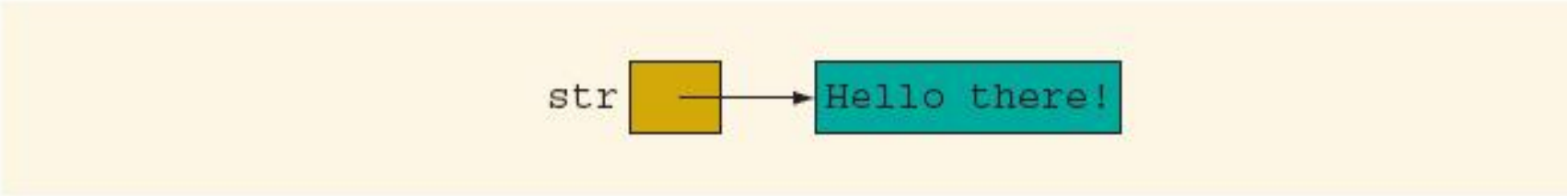

FIGURE 3-5 Variable `str` and the object `str`

Object and Reference Variables (continued)

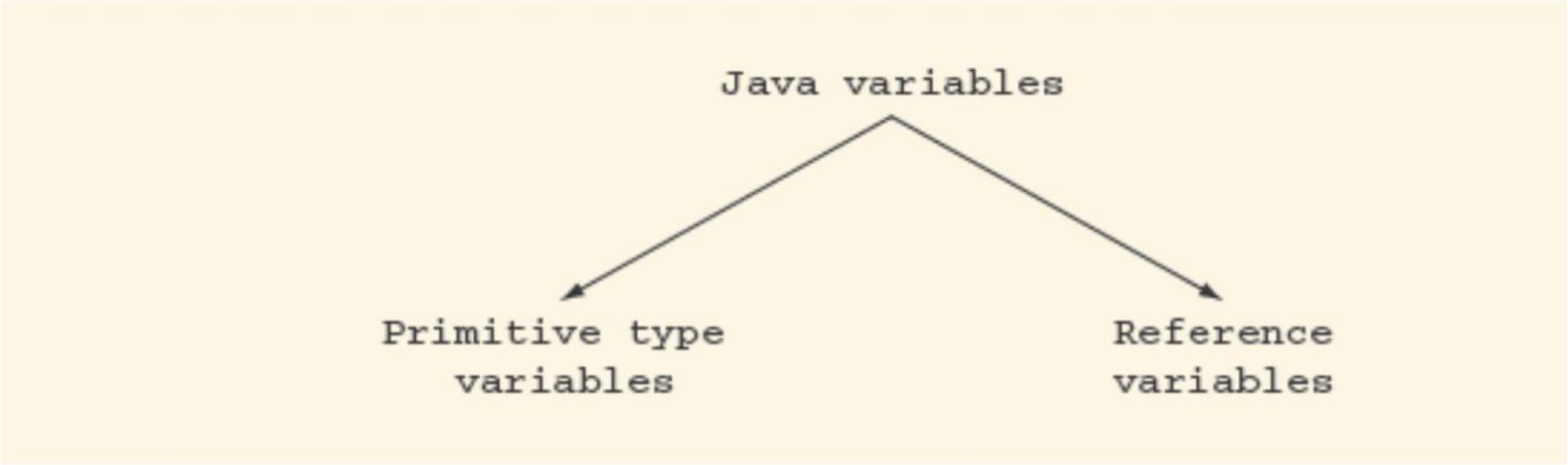

FIGURE 3-6 Java variables

Objects and Reference Variables (continued)

- Primitive type variables directly store data into their memory space
- Reference variables store the address of the object containing the data
- An object is an instance of a `class`, and the operator `new` is used to instantiate an object

Using Predefined Classes and Methods in a Program

- There are many predefined packages, classes, and methods in Java
- Library: collection of packages
- Package: contains several classes
- Class: contains several methods
- Method: set of instructions

Using Predefined Classes and Methods in a Program (continued)

- To use a method, you must know:
  - Name of the class containing method (`Math`)
  - Name of the package containing class (`java.lang`)
  - Name of the method - (`pow`), it has two parameters
  - `Math.pow(x, y)` = x^y

Using Predefined Classes and Methods in a Program (continued)

- Example method call

```
import java.lang; //imports package
Math.pow(2, 3);   //calls power method
                  // in class Math
```

- Dot (`.`) operator: used to access the method in the class

The `class String`

- String variables are reference variables
- Given:

```
String name;
```

  – Similar statements:

```
name = new String("Lisa Johnson");
name = "Lisa Johnson";
```

The `class String` (continued)

- A `String` object is an instance of `class String`
- The address of a `String` object with the value `"Lisa Johnson"` is stored in `name`
- `String` methods are called using the dot operator

The `class String` (continued)

```
sentence = "Programming with Java";
```

sentence = "Programming with Java";																				
P	r	o	g	r	a	m	m	i	n	g	' '	w	i	t	h	' '	J	a	v	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Some Commonly Used String Methods

TABLE 3-1 Some Commonly Used `String` Methods

```
char charAt(int index)
  //Returns the character at the position specified by index
  //Example: sentence.charAt(3) returns 'g'
```

```
int indexOf(char ch)
  //Returns the index of the first occurrence of the character
  //specified by ch; If the character specified by ch does not
  //appear in the string, it returns -1
  //Example: sentence.indexOf('J') returns 17
  //         sentence.indexOf('a') returns 5
```

```
int indexOf(char ch, int pos)
  //Returns the index of the first occurrence of the character
  //specified by ch; The parameter pos specifies where to
  //begin the search; If the character specified by ch does not

  //appear in the string, it returns -1
  //Example: sentence.indexOf('a', 10) returns 18
```

Some Commonly Used String Methods (continued)

```
int indexOf(String str)
  //Returns the index of the first occurrence of the string
  //specified by str; If the string specified by str does not
  //appear in the string, it returns -1
  //Example: sentence.indexOf("with") returns 12
  //         sentence.indexOf("ing") returns 8
```

```
int indexOf(String str, int pos)
  //Returns the index of the first occurrence of the String
  //specified by str; The parameter pos specifies where to begin
  //the search; If the string specified by str does not appear
  //in the string, it returns -1
  //Example: sentence.indexOf("a", 10) returns 18
  //         sentence.indexOf("Pr", 10) returns -1
```

```
String concat(String str)
  //Returns the string that is this string concatenated with str
  //Example: The expression
  //         sentence.concat(" is fun.")
  //         returns the string "Programming with Java is fun."
```

Some Commonly Used String Methods (continued)

```
int length()
  //Returns the length of the string
  //Example: sentence.length() returns 21, the number of characters in
  //         "Programming with Java"
```

```
String replace(char charToBeReplaced, char charReplacedWith)
  //Returns the string in which every occurrence of
  //charToBeReplaced is replaced with charReplacedWith
  //Example: sentence.replace('a', '*') returns the string
  //         "Progr*mming with J*v*"
  //         Each occurrence of a is replaced with *
```

```
String substring(int beginIndex)
  //Returns the string which is a substring of this string
  //beginning at beginIndex until the end of the string.
  //Example: sentence.substring(12) returns the string
  //         "with Java"
```

```
String substring(int beginIndex, int endIndex)
  //Returns the string which is a substring of this string
  //beginning at beginIndex until endIndex - 1
```

Some Commonly Used String Methods (continued)

```
String toLowerCase()
  //Returns the string that is the same as this string, except
  //that all uppercase letters of this string are replaced with
  //their equivalent lowercase letters
  //Example: sentence.toLowerCase() returns "programming with java"
```

```
String toUpperCase()
  //Returns the string that is the same as this string, except
  //that all lowercase letters of this string are replaced with
  //their equivalent uppercase letters
  //Example: sentence.toUpperCase() returns "PROGRAMMING WITH JAVA"
```

Some Commonly Used String Methods (continued)

```
boolean startsWith(String str)
  //Returns true if the string begins with the string specified by str;
  //otherwise, this methods returns false.
```

```
boolean endsWith(String str)
  //Returns true if the string ends with the string specified by str
  //otherwise, this methods returns false.
```

```
boolean regionMatches(int ind, String str, int strIndex, int len)
  //Returns true if the substring of str starting at strIndex and length
  //specified by len is same as the substring of this String
  //object starting at ind and having the same length
```

```
boolean regionMatches(boolean ignoreCase, int ind,
                      String str, int strIndex, int len)
  //Returns true if the substring of str starting at strIndex and length
  //specified by len is same as the substring of this String
  //object starting at ind and having the same length. If ignoreCase
  //is true, then during character comparison, case is ignored.
```

Input/Output

- Input data
- Format output
- Output results
- Format output
- Read from and write to files

Formatting Output with `printf`

- A syntax to use the method `printf` to produce output on the standard output device is:

  ```
  System.out.printf(formatString);
  ```

  or:

  ```
  System.out.printf(formatString,
  argumentList);
  ```

- `formatString` is a string specifying the format of the output, and `argumentList` is a list of arguments

Formatting Output with `printf` (continued)

- `argumentList` is a list of arguments that consists of constant values, variables, or expressions
- If there is more than one argument in `argumentList`, then the arguments are separated with commas

Formatting Output with `printf` (continued)

```
System.out.printf("Hello there!");
```

consists of only the format string, and the statement:

```
System.out.printf("There are %.2f
  inches in %d centimeters.%n",
   centimeters / 2.54, centimeters);
```

consists of both the format string and `argumentList`

Formatting Output with `printf` (continued)

- `%.2f` and `%d` are called format specifiers
- By default, there is a one-to-one correspondence between format specifiers and the arguments in `argumentList`
- The first format specifier `%.2f` is matched with the first argument, which is the expression `centimeters / 2.54`
- The second format specifier `%d` is matched with the second argument, which is centimeters

Formatting Output with `printf` (continued)

- The format specifier `%n` positions the insertion point at the beginning of the next line
- A format specifier for general, character, and numeric types has the following syntax:

```
%[argument_index$][flags][width][.precision]conversion
```

- The expressions in square brackets are optional; they may or may not appear in a format specifier

Formatting Output with `printf` (continued)

- The option *argument_index* is a (decimal) integer indicating the position of the argument in the argument list
  - The first argument is referenced by "`1$`", the second by `"2$"`, etc.
- The option *flags* is a set of characters that modify the output format
- The option *width* is a (decimal) integer indicating the minimum number of characters to be written to the output

Formatting Output with `printf` (continued)

- The option *precision* is a (decimal) integer usually used to restrict the number of characters
- The required *conversion* is a character indicating how the argument should be formatted

Formatting Output with `printf` (continued)

TABLE 3-2 Some of Java's Supported Conversions

`'s'`	general	The result is a string
`'c'`	character	The result is a Unicode character
`'d'`	integral	The result is formatted as a (decimal) integer
`'e'`	floating point	The result is formatted as a decimal number in computerized scientific notation
`'f'`	floating point	The result is formatted as a decimal number
`'%'`	percent	The result is `'%'`
`'n'`	line separator	The result is the platform-specific line separator

EXAMPLE 3-4

```
//Example: Fixed and scientific format

public class ScientificVsFixed
{
    public static void main(String[] args)
    {
        double hours = 35.45;
        double rate = 15.00;
        double tolerance = 0.01000;

        System.out.println("Fixed decimal notation:");
        System.out.printf("hours = %.2f, rate = %.2f, pay = %.2f,"
                        + " tolerance = %.2f%n%n",
                          hours, rate, hours * rate, tolerance);

        System.out.println("Scientific notation:");
        System.out.printf("hours = %e, rate = %e, pay = %e,%n"
                        + "tolerance = %e%n",
                          hours, rate, hours * rate, tolerance);
    }
}
```

Sample Run:

```
Fixed decimal notation:
hours = 35.45, rate = 15.00, pay = 531.75, tolerance = 0.01

Scientific notation:
hours = 3.545000e+01, rate = 1.500000e+01, pay = 5.317500e+02,
tolerance = 1.000000e-02
```

EXAMPLE 3-5

```
//Program to illustrate how to format the outputting of
//decimal numbers.

public class FormattingDecimalNumNew                    //Line 1
{                                                       //Line 2
    static final double PI = 3.14159265;                //Line 3

    public static void main(String[] args)              //Line 4
    {                                                   //Line 5
        double radius = 12.67;                          //Line 6
        double height = 12.00;                          //Line 7
```

```
        System.out.println("Two decimal places: ");          //Line 8

        System.out.printf("Line 9: radius = %.2f, "
                + "height = %.2f, volume = %.2f, "
                + "PI = %.2f%n%n", radius, height,
                PI * radius * radius * height, PI);          //Line 9

        System.out.println("Three decimal places: ");        //Line 10
        System.out.printf("Line 11: radius = %.3f, "
                + "height = %.3f, volume = %.3f,%n"
                + "          PI = %.3f%n%n", radius,
                height,PI * radius * radius * height, PI); //Line 11

        System.out.println("Four decimal places: ");         //Line 12
        System.out.printf("Line 13: radius = %.4f, "
                + "height = %.4f, volume = %.4f,%n "
                + "         PI = %.4f%n%n", radius,
                height,PI * radius * radius * height, PI); //Line 13

        System.out.printf("Line 14: radius = %.3f, "
                  + "height = %.2f, PI = %.5f%n",
                   radius, height, PI);                      //Line 14
    }                                                        //Line 15
}                                                            //Line 16
```

Sample Run:

```
Two decimal places:
Line 9: radius = 12.67, height = 12.00, volume = 6051.80, PI = 3.14

Three decimal places:
Line 11: radius = 12.670, height = 12.000, volume = 6051.797,
         PI = 3.142

Four decimal places:
Line 13: radius = 12.6700, height = 12.0000, volume = 6051.7969,
         PI = 3.1416

Line 14: radius = 12.670, height = 12.00, PI = 3.14159
```

```
num = 96;
rate = 15.50;
```

Consider the following statements:

```
System.out.println("123456789012345");          //Line 1
System.out.printf("%5d %n", num);                //Line 2
System.out.printf("%5.2f %n", rate);            //Line 3
System.out.printf("%5d%6.2f %n", num, rate);    //Line 4
System.out.printf("%5d %6.2f %n", num, rate);   //Line 5
```

The output of these statements is:

```
123456789012345
   96
15.50
   96 15.50
   96  15.50
```

EXAMPLE 3-6

```
public class FormattingOutputWithprintf
{
    public static void main(String[] args)
    {
        int num = 763;                                  //Line 1

        double x = 658.75;                              //Line 2

        String str = "Java Program.";                   //Line 3

        System.out.println("12345678901234567 89"
                           + "01234567890");            //Line 4
        System.out.printf("%5d%7.2f%15s%n",
                            num, x, str);               //Line 5
        System.out.printf("%15s%6d%9.2f%n",
                            str, num, x);               //Line 6
        System.out.printf("%8.2f%7d%15s%n",
                            x, num, str);               //Line 7

        System.out.printf("num = %5d%n", num);          //Line 8
        System.out.printf("x = %10.2f%n", x);           //Line 9
        System.out.printf("str = %15s%n", str);         //Line 10
        System.out.printf("%10s%7d%n",
                           "Program No.", 4);           //Line 11
    }
}
```

Sample Run:

```
12345678901234567890123456789 0
  763 658.75  Java Program.
  Java Program.   763   658.75
  658.75    763  Java Program.
num =   763
x =     658.75
str =   Java Program.
Program No.      4
```

EXAMPLE 3-7

```
public class Example3_7
{
    public static void main(String[] args)
    {
        int num = 763;                                  //Line 1
        double x = 658.75;                              //Line 2
        String str = "Java Program.";                   //Line 3

        System.out.println("12345678901234567 89"
                         + "01234567890");              //Line 4
        System.out.printf("%-5d%-7.2f%-15s ***%n",
                           num, x, str);                //Line 5
        System.out.printf("%-15s%-6d%-9.2f ***%n",
                           str, num, x);                //Line 6
        System.out.printf("%-8.2f%-7d%-15s ***%n",
                           x, num, str);                //Line 7

        System.out.printf("num = %-5d ***%n", num);     //Line 8
        System.out.printf("x = %-10.2f ***%n", x);      //Line 9
        System.out.printf("str = %-15s ***%n", str);    //Line 10
        System.out.printf("%-10s%-7d ***%n",
                          "Program No.", 4);            //Line 11
    }
}
```

Sample Run:

```
1234567890123456789012345678 90
763  658.75 Java Program.   ***
Java Program.  763   658.75    ***
658.75  763    Java Program.   ***
num = 763   ***
x = 658.75     ***
str = Java Program.   ***
Program No.4       ***
```

Parsing Numeric Strings

- A string consisting of only integers or decimal numbers is called a numeric string

1. To convert a string consisting of an integer to a value of the type `int`, we use the following expression:

```
Integer.parseInt(strExpression)

Integer.parseInt("6723") = 6723
Integer.parseInt("-823") = -823
```

Parsing Numeric Strings (continued)

2. To convert a string consisting of a decimal number to a value of the type float, we use the following expression:

```
Float.parseFloat(strExpression)

Float.parseFloat("34.56") = 34.56
Float.parseFloat("-542.97") = -542.97
```

Parsing Numeric Strings (continued)

3. To convert a string consisting of a decimal number to a value of the type double, we use the following expression:

```
Double.parseDouble(strExpression)

Double.parseDouble("345.78") = 345.78
Double.parseDouble("-782.873") = -
782.873
```

Parsing Numeric Strings (continued)

- `Integer`, `Float`, and `Double` are classes designed to convert a numeric string into a number
- These classes are called **wrapper** classes
- `parseInt` is a method of the `class` `Integer`, which converts a numeric integer string into a value of the type `int`

Parsing Numeric Strings (continued)

- `parseFloat` is a method of the `class` `Float` and is used to convert a numeric decimal string into an equivalent value of the type `float`
- `parseDouble` is a method of the `class` `Double`, which is used to convert a numeric decimal string into an equivalent value of the type `double`

Using Dialog Boxes for Input/Output

- Use a graphical user interface (GUI)
- `class JOptionPane`
  - Contained in package `javax.swing`
  - Contains methods `showInputDialog` and `showMessageDialog`
- Syntax

```
str = JOptionPane.showInputDialog(strExpression)
```

- Program must end with `System.exit(0);`

Using Dialog Boxes for Input/Output (continued)

FIGURE 3-8 Input dialog box with user input

Parameters for the Method `showMessageDialog`

TABLE 3-3 Parameters for the Method `showMessageDialog`

Parameter	Description
`parentComponent`	This is an object that represents the parent of the dialog box. For now, we will specify the `parentComponent` to be `null`, in which case the program uses a default component that causes the dialog box to appear in the middle of the screen. Note that `null` is a reserved word in Java.
`messageStringExpression`	The `messageStringExpression` is evaluated and its value appears in the dialog box.
`boxTitleString`	The `boxTitleString` represents the title of the dialog box.
`messageType`	An `int` value representing the type of icon that will appear in the dialog box. Alternatively, you can use certain `JOptionPane` options described below.

JOptionPane Options for the Parameter messageType

TABLE 3-4 JOptionPane Options for the Parameter messageType

messageType	Description
JOptionPane.**ERROR_MESSAGE**	The error icon,  , is displayed in the dialog box.
JOptionPane.**INFORMATION_MESSAGE**	The information icon,  , is displayed in the dialog box.
JOptionPane.**PLAIN_MESSAGE**	No icon appears in the dialog box.
JOptionPane.**QUESTION_MESSAGE**	The question icon,  , is displayed in the dialog box.
JOptionPane.**WARNING_MESSAGE**	The warning icon,  , is displayed in the dialog box.

JOptionPane Example

The output of the statement

```
JOptionPane.showMessageDialog(null, "Hello World!", "Greetings",
                              JOptionPane.INFORMATION_MESSAGE);
```

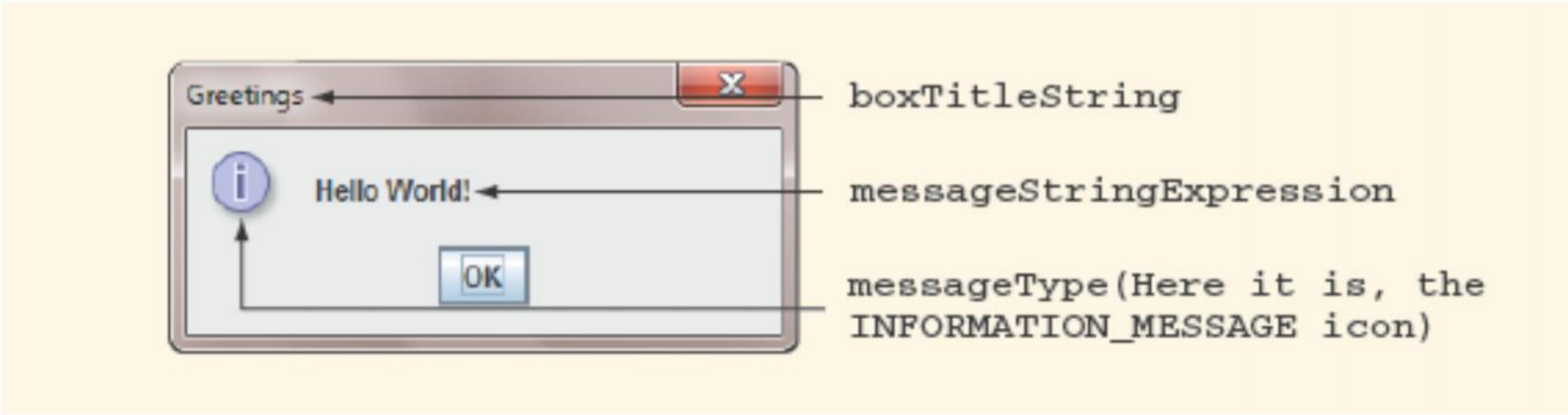

FIGURE 3-9 Message dialog box showing its various components

Formatting the Output Using the `String` Method `format`

Example 3-12

```
double x = 15.674;
double y = 235.73;
double z = 9525.9864;
int num = 83;
String str;
```

Expression	Value
`String.format("%.2f", x)`	`"15.67"`
`String.format("%.3f", y)`	`"235.730"`
`String.format("%.2f", z)`	`"9525.99"`
`String.format("%7s", "Hello")`	`"  Hello"`
`String.format("%5d%7.2f", num, x)`	`"   83  15.67"`
`String.format("The value of num = %5d", num)`	`"The value of num = 83"`
`str = String.format("%.2f", z)`	`str = "9525.99"`

File Input/Output

- File: area in secondary storage used to hold information
- You can also initialize a `Scanner` object to input sources other than the standard input device by passing an appropriate argument in place of the object `System.in`
- We make use of the `class FileReader`

File Input/Output (continued)

- Suppose that the input data is stored in a file, say `prog.dat`, and this file is on the floppy disk `A`
- The following statement creates the `Scanner` object `inFile` and initializes it to the file `prog.dat`
-
```
Scanner inFile = new Scanner
        (new FileReader("prog.dat"));
```

File Input/Output (continued)

- Next, you use the object `inFile` to input data from the file `prog.dat` just the way you used the object `console` to input data from the standard input device using the methods `next`, `nextInt`, `nextDouble`, and so on

File Input/Output (continued)

```
Scanner inFile = new Scanner(new FileReader("prog.dat"));  //Line 1
```

The statement in Line 1 assumes that the file **prog.dat** is in the same directory (subdirectory) as your program. However, if this is in a different directory (subdirectory), then you must specify the path where the file is located, along with the name of the file. For example, suppose that the file **prog.dat** is on a flash memory in drive H. Then, the statement in Line 1 should be modified as follows:

```
Scanner inFile = new Scanner(new FileReader("h:\\prog.dat"));
```

Note that there are two \ after **h:**. Recall from Chapter 2 that in Java \ is the escape character. Therefore, to produce a \ within a string you need \\. (Moreover, to be absolutely sure about specifying the source where the input file is stored, such as the flash drive **h:**, check your system's documentation.)

File Input/Output (continued)

```
Scanner inFile = new Scanner(new FileReader("prog.dat"));  //Line 1
```

The statement in Line 1 assumes that the file prog.dat is in the same directory (subdirectory) as your program. However, if this is in a different directory (subdirectory), then you must specify the path where the file is located, along with the name of the file. For example, suppose that the file `prog.dat` is on a flash memory in drive `H`. Then the statement in Line 1 should be modified as follows:

```
Scanner inFile = new Scanner(new
                    FileReader("h:\\prog.dat"));
```

Note that there are two `\` after `h:`. Recall that in Java `\` is the escape character. Therefore, to produce a `\` within a string you need `\\`. (Moreover, to be absolutely sure about specifying the source where the input file is stored, such as the flash memory in drive `h:\\`, check your system's documentation.)

File Input/Output (continued)

- Java file I/O process
  1. Import necessary classes from the packages `java.util` and `java.io` into the program
  2. Create and associate appropriate objects with the input/output sources
  3. Use the appropriate methods associated with the variables created in Step 2 to input/output data
  4. Close the files

File Input/Output (continued)

Example 3-16

Suppose an input file, say `employeeData.txt`, consists of the following data:

```
Emily Johnson 45 13.50

Scanner inFile = new Scanner
      (new FileReader("employeeData.txt"));
String firstName;
String lastName;
double hoursWorked;
double payRate;
double wages;
firstName = inFile.next();
lastName = inFile.next();
hoursWorked = inFile.nextDouble();
payRate = inFile.nextDouble();
wages = hoursWorked * payRate;

inFile.close();  //close the input file
```

Storing (Writing) Output to a File

- To store the output of a program in a file, you use the `class` `PrintWriter`
- Declare a `PrintWriter` variable and associate this variable with the destination
- Suppose the output is to be stored in the file `prog.out`

Storing (Writing) Output to a File (continued)

- Consider the following statement:

```
PrintWriter outFile = new
PrintWriter("prog.out");
```

- This statement creates the `PrintWriter` object `outFile` and associates it with the file `prog.out`
- You can now use the methods `print`, `println`, and `printf` with `outFile` just the same way they have been used with the object `System.out`

Storing (Writing) Output to a File (continued)

- The statement:

```
outFile.println("The paycheck is: $" + pay);
```

stores the output—`The paycheck is: $565.78`—in the file `prog.out`

-This statement assumes that the value of the variable `pay` is `565.78`

Storing (Writing) Output to a File (continued)

- Step 4 requires closing the file; you close the input and output files by using the method close

```
inFile.close();
outFile.close();
```

- Closing the output file ensures that the buffer holding the output will be emptied; that is, the entire output generated by the program will be sent to the output file

`throws` Clause

- During program execution, various things can happen; for example, division by zero or inputting a letter for a number
- In such cases, we say that an exception has occurred
- If an exception occurs in a method, the method should either handle the exception or *throw* it for the calling environment to handle
- If an input file does not exist, the program throws a `FileNotFoundException`

`throws` Clause (continued)

- If an output file cannot be created or accessed, the program throws a `FileNotFoundException`
- For the next few chapters, we will simply throw the exceptions
- Because we do not need the method `main` to handle the `FileNotFoundException` exception, we will include a command in the heading of the method `main` to throw the `FileNotFoundException` exception

Skeleton of I/O Program

```
import java.io.*;
import java.util.*;

//Add additional import statements as needed

public class ClassName
{
      //Declare appropriate variables
    public static void main(String[] args)
                                throws FileNotFoundException
    {
              //Create and associate the stream objects
        Scanner inFile =
              new Scanner(new FileReader("prog.dat"));

        PrintWriter outFile = new PrintWriter("prog.out");

         //Code for data manipulation

            //Close file
        inFile.close();
        outFile.close();
    }
}
```

Programming Example: Movie Ticket Sale and Donation to Charity

- Input: movie name, adult ticket price, child ticket price, number of adult tickets sold, number of child tickets sold, percentage of gross amount to be donated to charity
- Output:

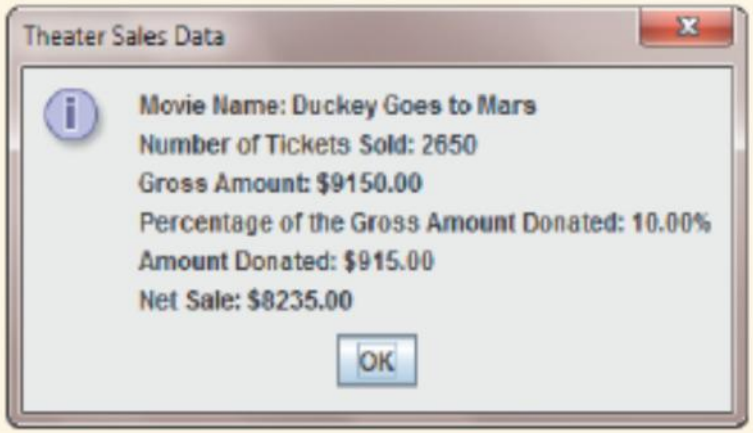

FIGURE 3-14 Output of theater sales program

Programming Example: Movie Ticket Sale and Donation to Charity (continued)

- Import appropriate packages
- Get inputs from user using `JOptionPane.showInputDialog`
- Perform appropriate calculations
- Display output using `JOptionPane.showMessageDialog`

Programming Example: Student Grade

- Input: file containing student's first name, last name, five test scores
- Output: file containing student's first name, last name, five test scores, average of five test scores

Programming Example: Student Grade (continued)

- Import appropriate packages
- Get input from file using the `class`es `Scanner` and `FileReader`
- Read and calculate the average of test scores
- Write to output file using the `class` `PrintWriter`
- Close files

Chapter Summary

- Primitive type variables store data into their memory space
- Reference variables store the address of the object containing the data
- An object is an instance of a class

Chapter Summary (continued)

- Operator `new` is used to instantiate an object
- Garbage collection is reclaiming memory not being used
- To use a predefined method, you must know its name and the class and package it belongs to
- The dot (.) operator is used to access a certain method in a class

Chapter Summary (continued)

- Methods of the `class String` are used to manipulate input and output data
- Dialog boxes can be used to input data and output results
- Data can be read from and written to files
- Data can be formatted using the `String` method `format`