Lecture #6
CSC 200-04L Fall 2018

Ambrose Lewis
tjl274@email.vccs.edu

Motivations

Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
System.out.println("Welcome to Java!");
```

So, how do you solve this problem?

Opening Problem

Problem:

100 times

```
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");

…

…

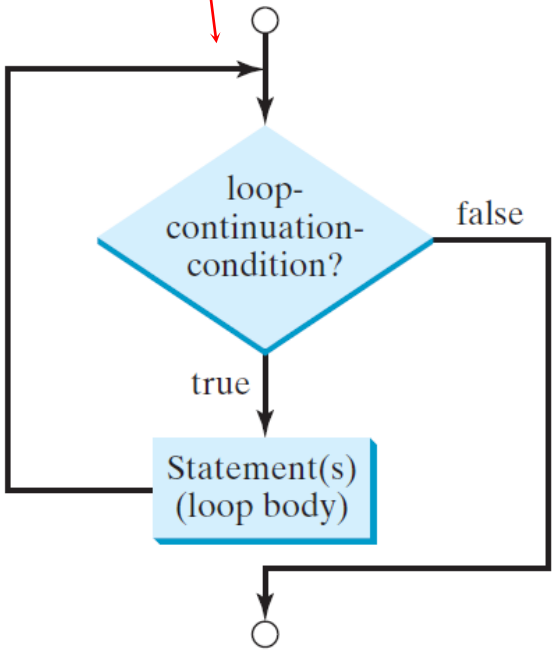
…
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
```

Introducing while Loops

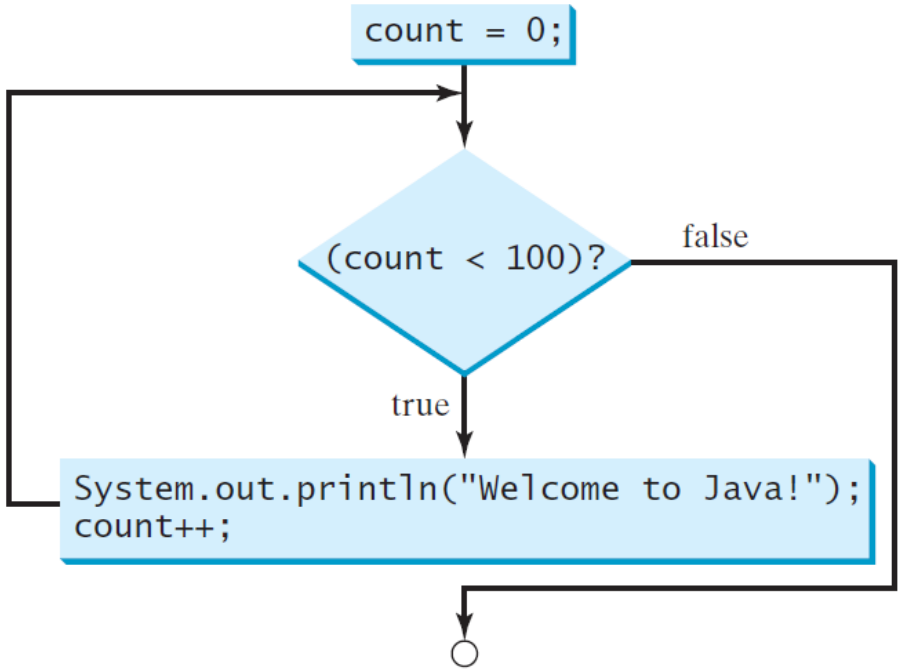
```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java");
  count++;
}
```

`while` Loop Flow Chart

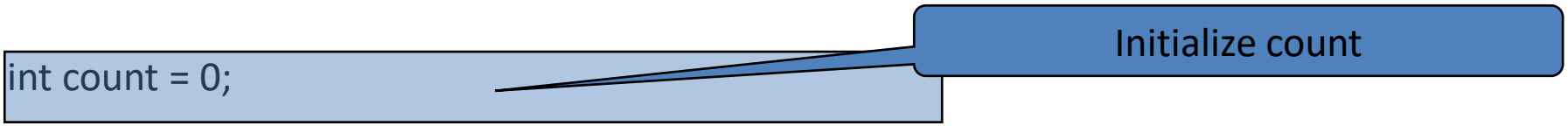
```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```




```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```

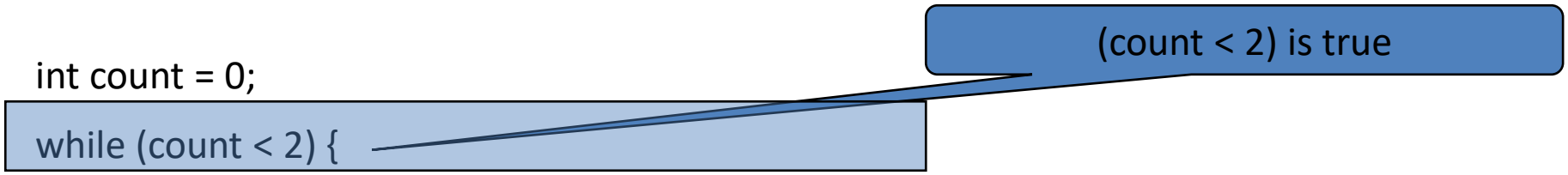


Trace while Loop



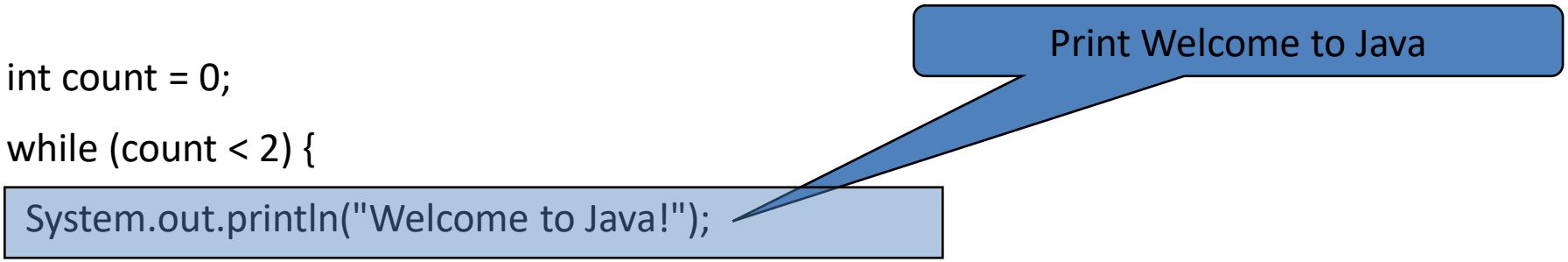

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Trace while Loop, cont.



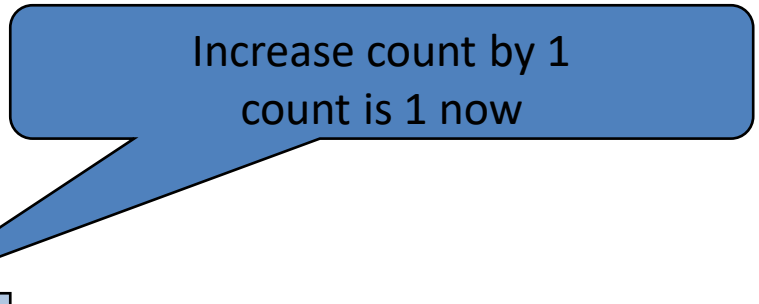

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Trace while Loop, cont.



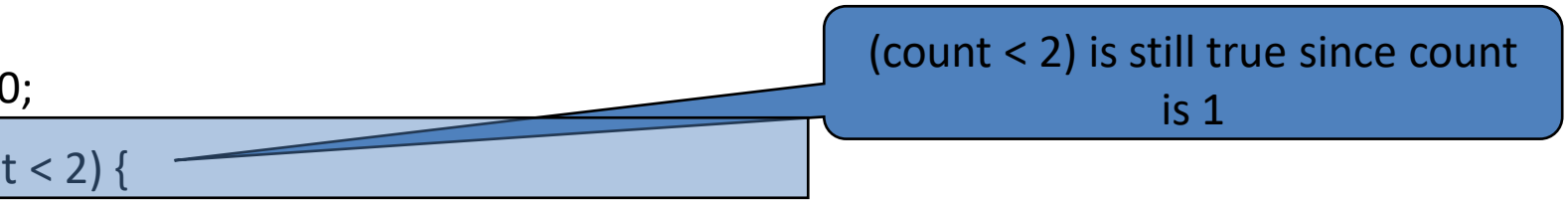
Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```



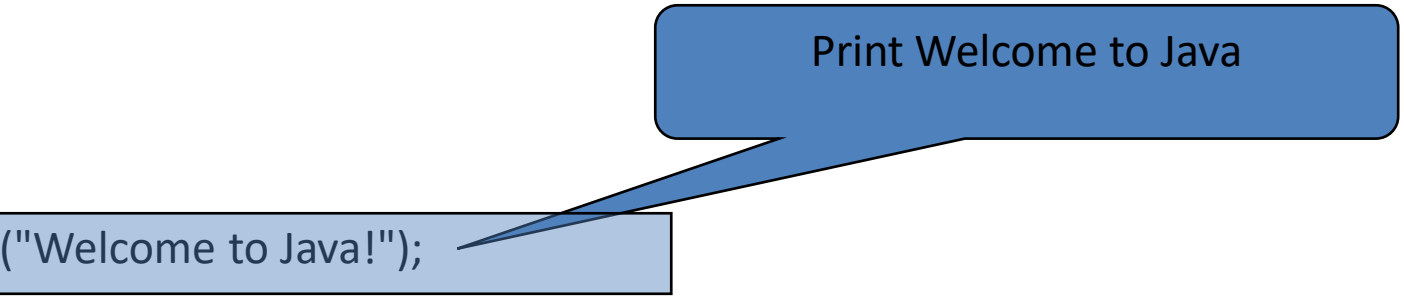
Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```



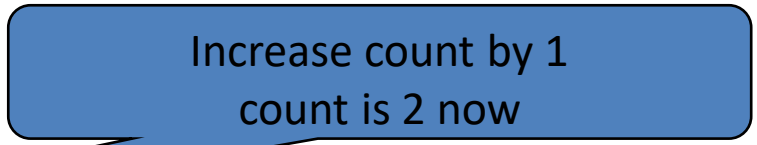
Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

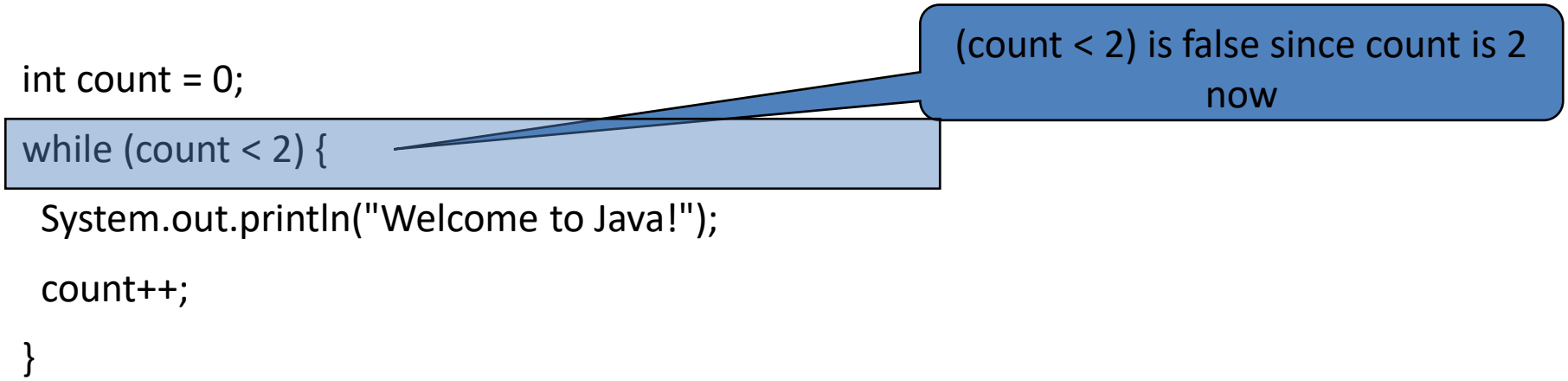


Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

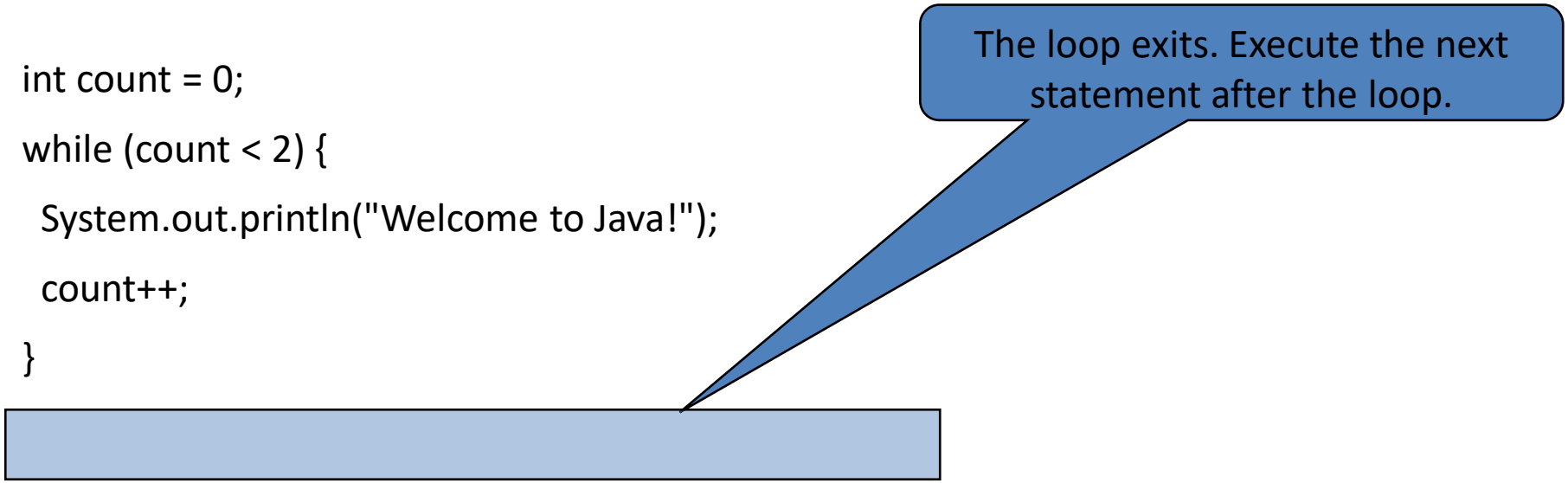


Trace while Loop, cont.



Trace while Loop

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```



Problem: Repeat Addition Until Correct

Write a Java program, AdditionQuiz.java, that prompts the user to enter an answer for a question on addition of two single digits. Using a loop, write the program to let the user enter a new answer until it is correct.

Problem: Guessing Numbers

Write a program, Guesser.java, that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.

Problem: Subtraction Tester

Write a Java program that generates five random single digit subtraction problems and gets an answer from the user. After the user answers the five questions, the program reports the number of the correct answers after the user answers all five questions.

Ending a Loop with a Sentinel Value

Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

Write a program named Sentinel.java, that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

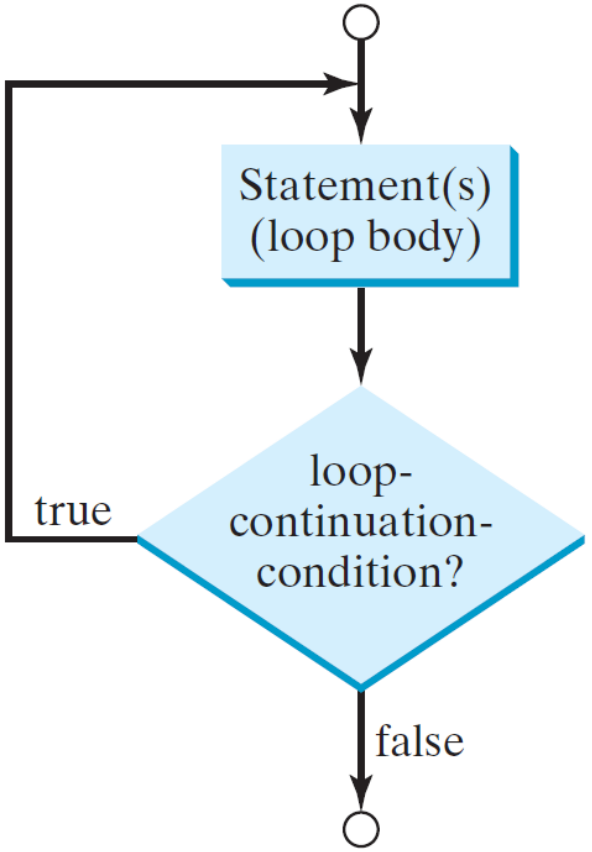
Caution

Don’t use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1:

```
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```

do-while Loop

- Loop body (statements between do and while lines) will be executed at least once!
- This is a good way to validate input from user. We will program an example, DoWhileExample.java, that shows this together

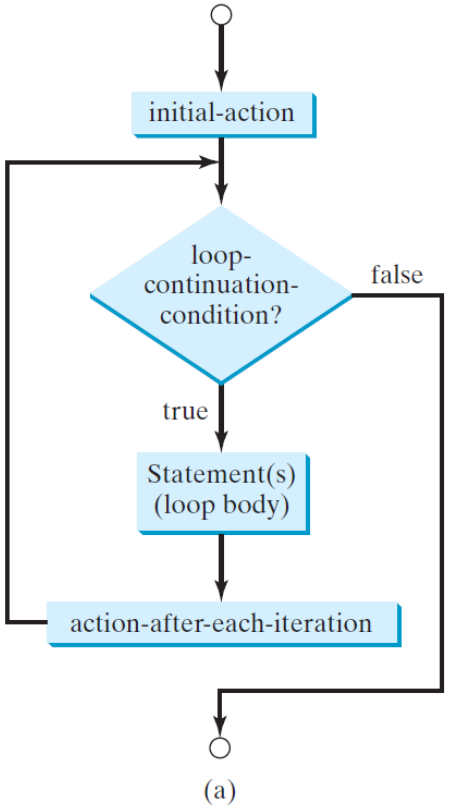



```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

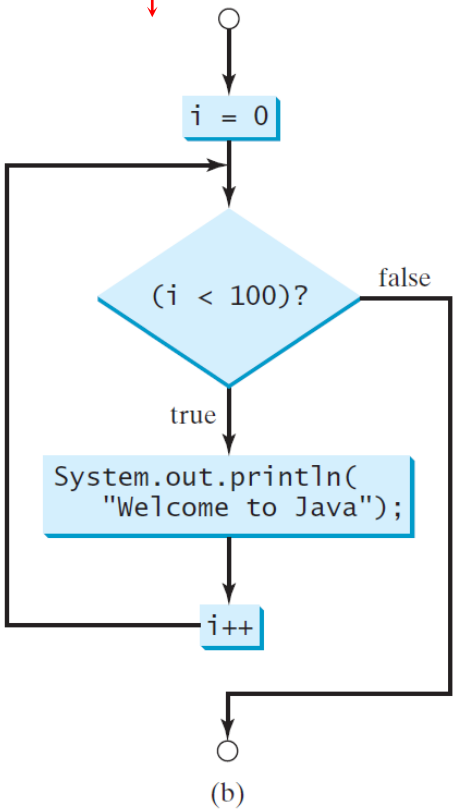
`for` Loops

```
for (initial-action; loop-
    continuation-condition; action-
    after-each-iteration) {
  // loop body;
  Statement(s);
}
```

```
int i;
for (i = 0; i < 100; i++) {
  System.out.println(
    "Welcome to Java!");
}
```



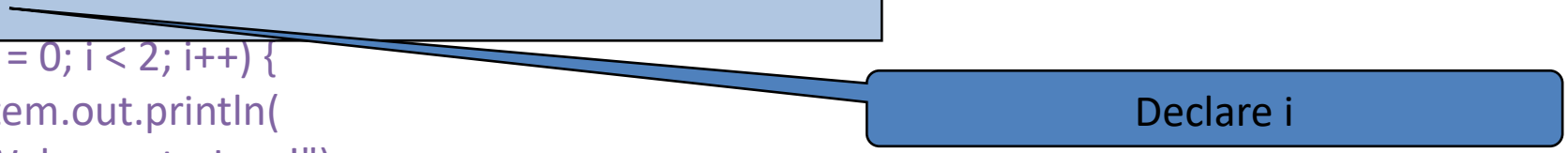
(a)



(b)

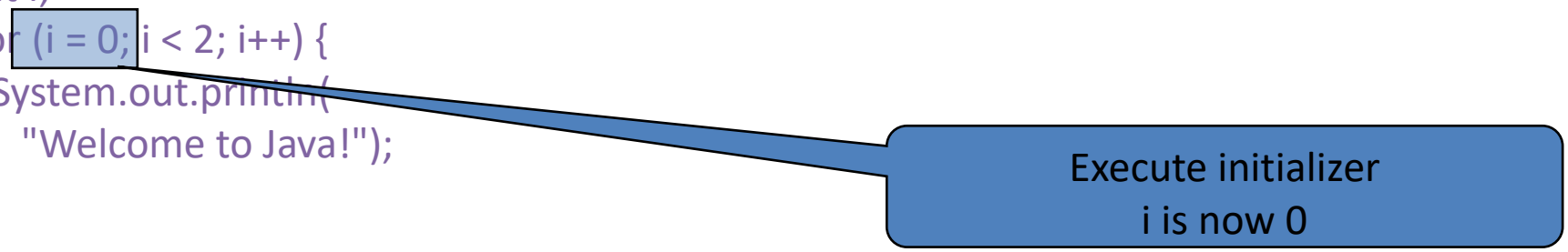
Trace for Loop

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```



Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
    "Welcome to Java!");
}
```

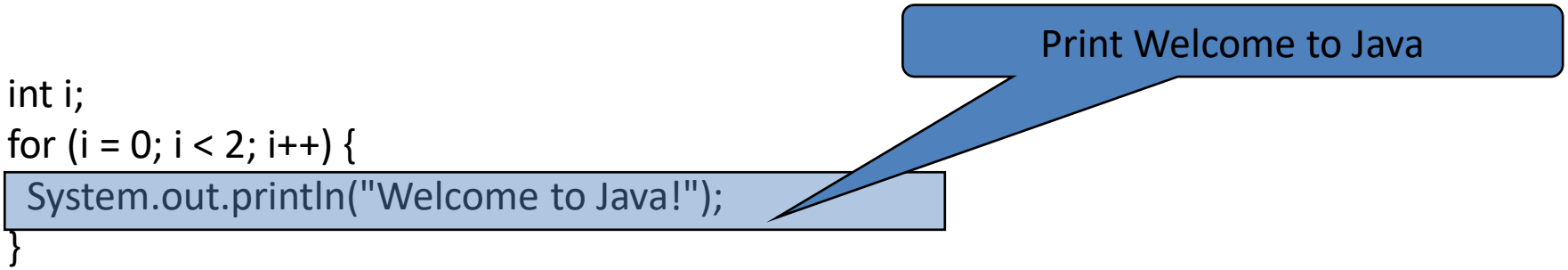


Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println( "Welcome to Java!");
}
```

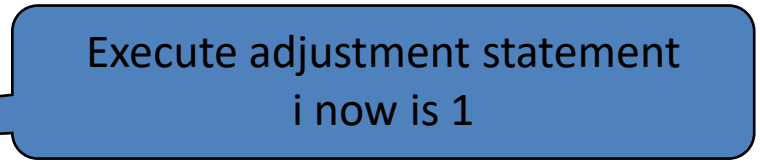
Evaluate Logical Expression
(i < 2) is true since i is 0

Trace for Loop, cont.



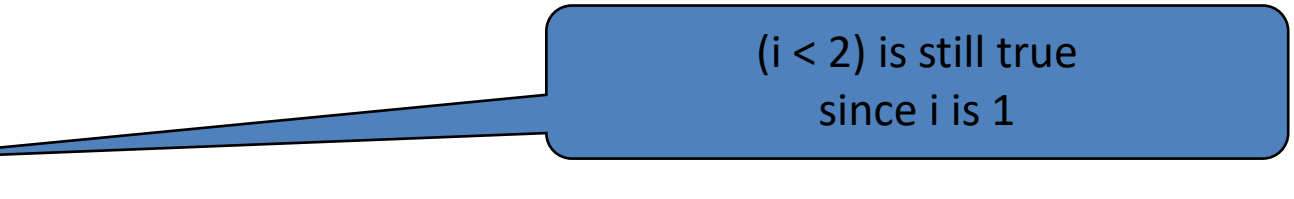
Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

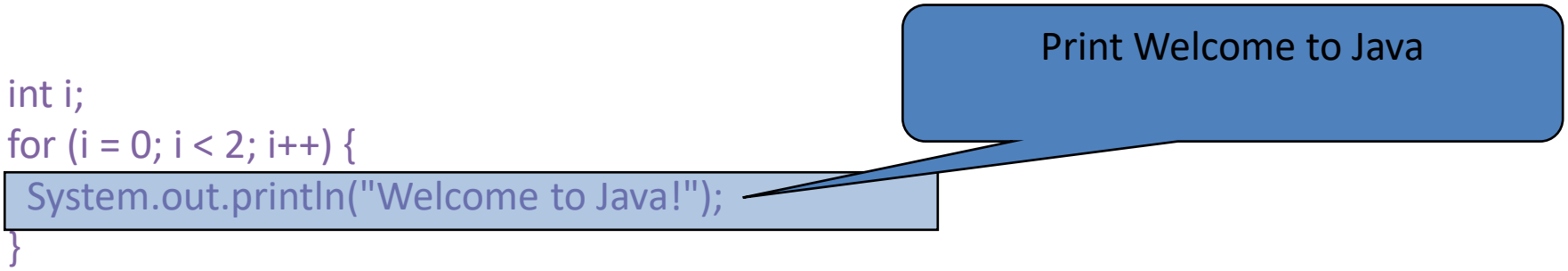


Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```



Trace for Loop, cont.

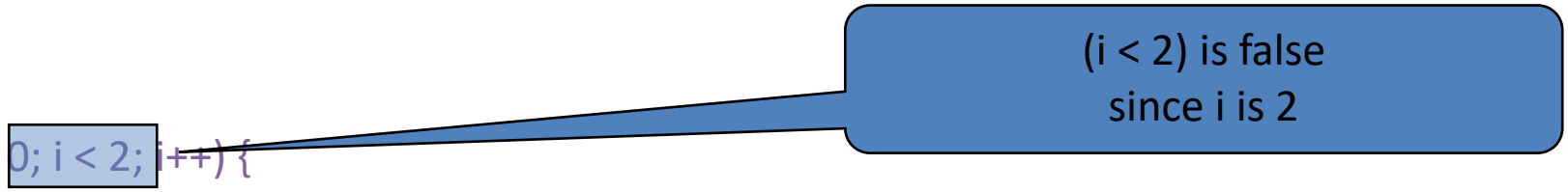


Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

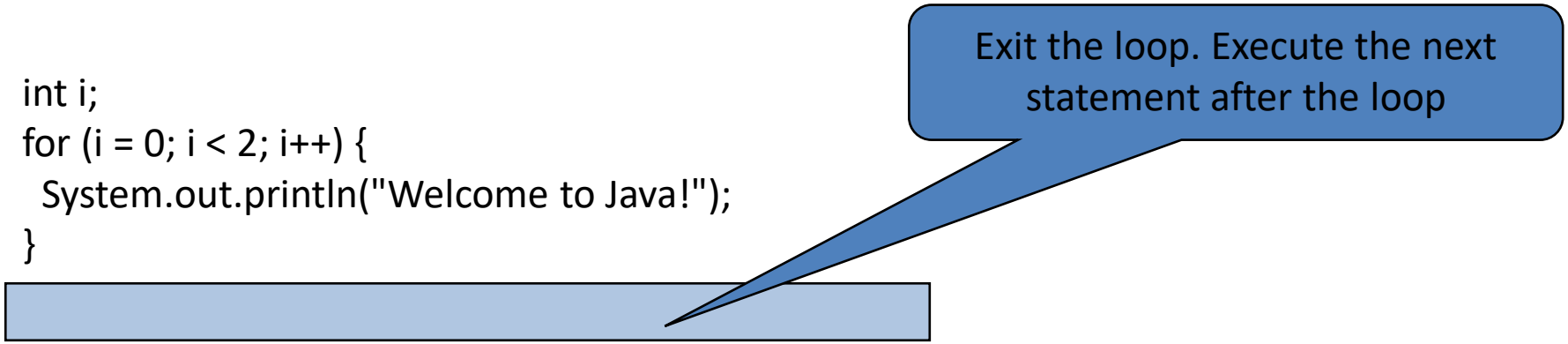
Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```



Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```



Note

The initial-action in a for loop can be a list of zero or more comma-separated expressions. The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. Therefore, the following two for loops are correct. They are rarely used in practice, however.

```
for (int i = 1; i < 100; System.out.println(i++));
```

```
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
  // Do something

}
```

Warning: It is considered very bad practice to adjust the value of the loop control variable inside the loop body…Don't do it!

Note

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

Caution

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

Logic Error

```
for (int i=0; i<10; i++);
{
  System.out.println("i is " + i);
}
```

Caution, cont.

Similarly, the following loop is also wrong:

```
int i=0;
while (i < 10);
{
  System.out.println("i is " + i);
  i++;
}
```

Logic Error

In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
  System.out.println("i is " + i);
  i++;
} while (i<10);
```

Correct

Which Loop to Use?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

```
while (loop-continuation-condition) {
  // Loop body
}
```

(a)

Equivalent

```
for ( ; loop-continuation-condition; )
  // Loop body
}
```

(b)

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases (see Review Question 3.19 for one of them):

```
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
}
```

(a)

Equivalent

```
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```

(b)

Recommendations

Use the one that is most intuitive and comfortable for you. In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times. A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0. A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

Nested Loops

Problem: Write a program, Table.java, that uses nested for loops to print a multiplication table.

Problem:
Finding the Greatest Common Divisor

Problem: Write a program, GCD.java, that prompts the user to enter two positive integers and finds their greatest common divisor.

Solution: Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8. So, how do you find the greatest common divisor? Let the two input integers be n1 and n2. You know number 1 is a common divisor, but it may not be the greatest commons divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

[Code on next slide…]

GCD.java

```
import java.util.Scanner;
public class GreatestCommonDivisor {
   public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter first integer: ");
        int n1 = input.nextInt();
        System.out.print("Enter second integer: ");
        int n2 = input.nextInt();
        int gcd = 1;
        int k = 2;
        while (k <= n1 && k <= n2) {
                if (n1 % k == 0 && n2 % k == 0) {
                 gcd = k; k++;
                }
        }
        System.out.println("GCD for " + n1 + " and " + n2 + " is " + gcd);
   }
}
```

Problem: Predicting the Future Tuition

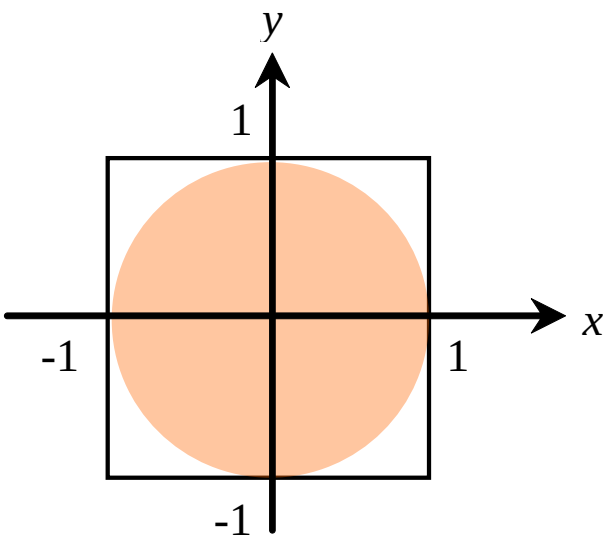
Problem: Suppose that the tuition for a university is $10,000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?

Problem: Predicating the Future Tuition

```
double tuition = 10000;   int year = 0  // Year 0
tuition = tuition * 1.07; year++;       // Year 1
tuition = tuition * 1.07; year++;       // Year 2
tuition = tuition * 1.07; year++;       // Year 3
...
```

Problem: *Monte Carlo Simulation*

The Monte Carlo simulation refers to a technique that uses random numbers and probability to solve problems. This method has a wide range of applications in computational mathematics, physics, chemistry, and finance. This section gives an example of using the Monto Carlo simulation for estimating π.



circleArea / squareArea = π / 4.

π can be approximated as 4 * numberOfHits / numberOfTrials

Problem: Monte Carlo Simulation

```
public class MonteCarloSimulation {
 public static void main(String[] args) {
   final int NUMBER_OF_TRIALS = 10000000;
   int numberOfHits = 0;
   for (int i = 0; i < NUMBER_OF_TRIALS; i++) {
           double x = Math.random() * 2.0 - 1;
           double y = Math.random() * 2.0 - 1;
           if (x * x + y * y <= 1) numberOfHits++;
   }
   double pi = 4.0 * numberOfHits / NUMBER_OF_TRIALS;
   System.out.println("PI is " + pi);
  } // endmain…
 } // endclass…
```

Using `break` and `continue`

The `break` statement tells Java to exit out of a loop

The `continue` statement tells Java to jump immediately to the end of the loop (that is, skip the rest of the loop body for this iteration)

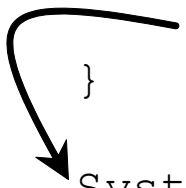
Example of each follow…

break

```
public class TestBreak {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      sum += number;
      if (sum >= 100)
        break;
    }

    System.out.println("The number is " + number);
    System.out.println("The sum is " + sum);
  }
}
```



continue

```
public class TestContinue {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      if (number == 10 || number == 11)
        continue;
      sum += number;
    }

    System.out.println("The sum is " + sum);
  }
}
```

Problem: Displaying Prime Numbers

Problem: Write a program named Prime.java that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is *prime* if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

Solution: The problem can be broken into the following tasks:

- For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
- Determine whether a given number is prime.
- Count the prime numbers.
- Print each prime number, and print 10 numbers per line.

Code on the next page!

```
public class PrimeNumber {
    public static void main(String[] args) {
            final int NUMBER_OF_PRIMES = 50; // Number of primes to display
            final int NUMBER_OF_PRIMES_PER_LINE = 10; // Display 10 per line
            int count = 0; // Count the number of prime numbers
            int number = 2; // A number to be tested for primeness
            System.out.println("The first 50 prime numbers are \n");
            // Repeatedly find prime numbers
            while (count < NUMBER_OF_PRIMES) {
                        // Assume the number is prime
                        boolean isPrime = true; // Is the current number prime?
                        // Test if number is prime
                        for (int divisor = 2; divisor <= number / 2; divisor++) {
                                    if (number % divisor == 0) { // If true, number is not prime
                                                isPrime = false; // Set isPrime to false
                                                break; // Exit the for loop
                                    }  // endif...
                        } // endfor..
                        // Print the prime number and increase the count
                        if (isPrime) {
                                    count++; // Increase the count

                                    if (count % NUMBER_OF_PRIMES_PER_LINE == 0) {
                                                // Print the number and advance to the new line
                                                System.out.println(number);
                                    } else {
                                                System.out.print(number + " ");
                                    } // endif...
                        } // endif
                        // Check if the next number is prime
                        number++;
            }// endwhile...
    } // end main...
} // end class
```

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
```

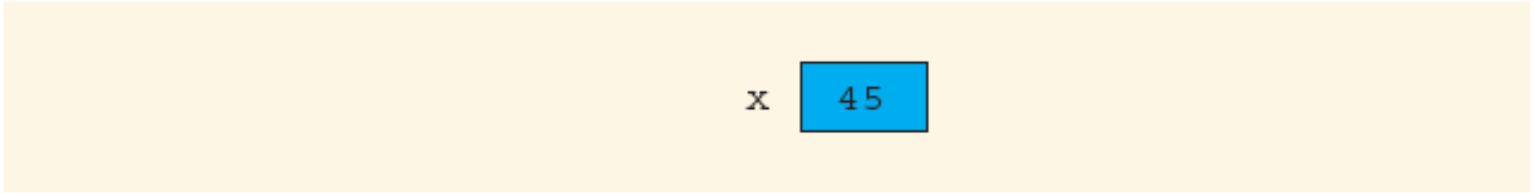


FIGURE 3-1 Variable x and its data

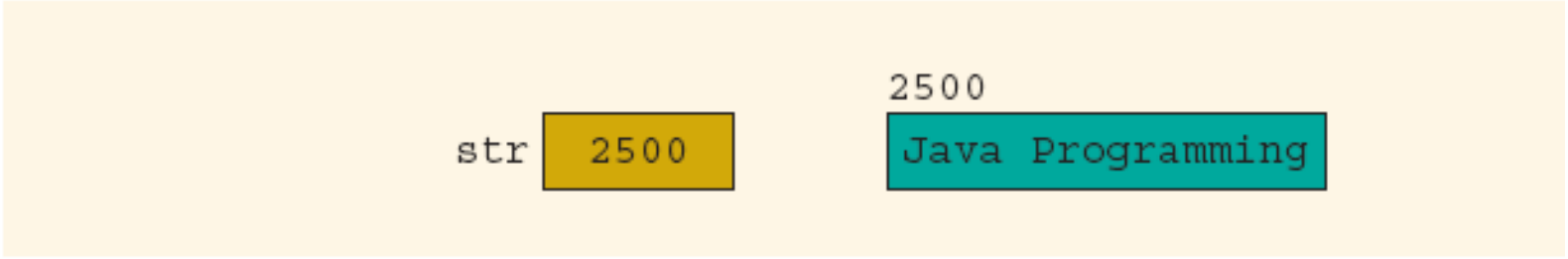


FIGURE 3-2 Variable str and the data it points to

Object and Reference Variables (continued)

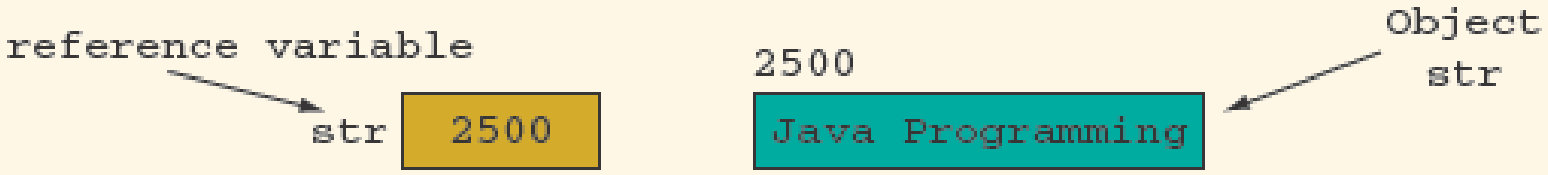


FIGURE 3-3 Variable `str` and object `str`

Object and Reference Variables (continued)

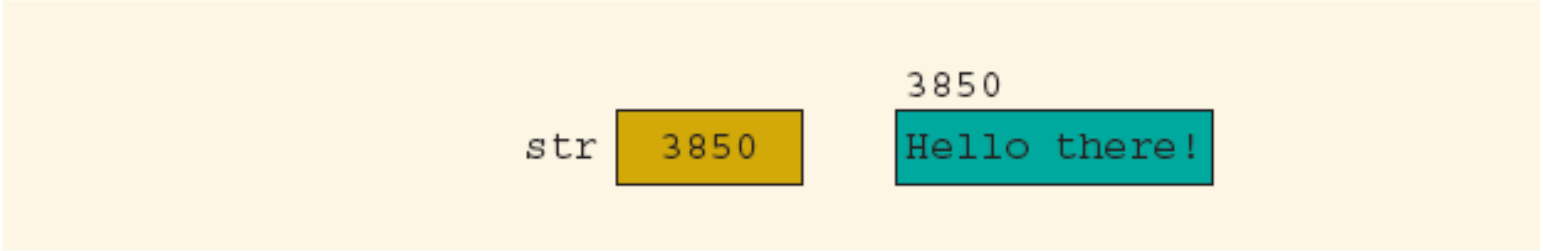


FIGURE 3-4 Variable `str`, its value, and the object `str`

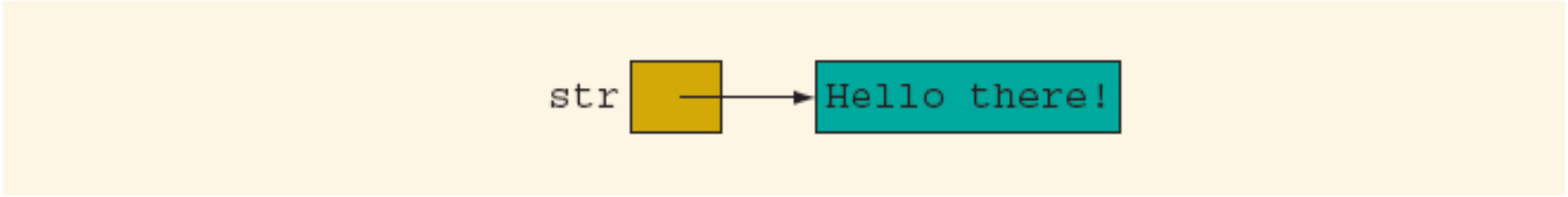


FIGURE 3-5 Variable `str` and the object `str`

Object and Reference Variables (continued)

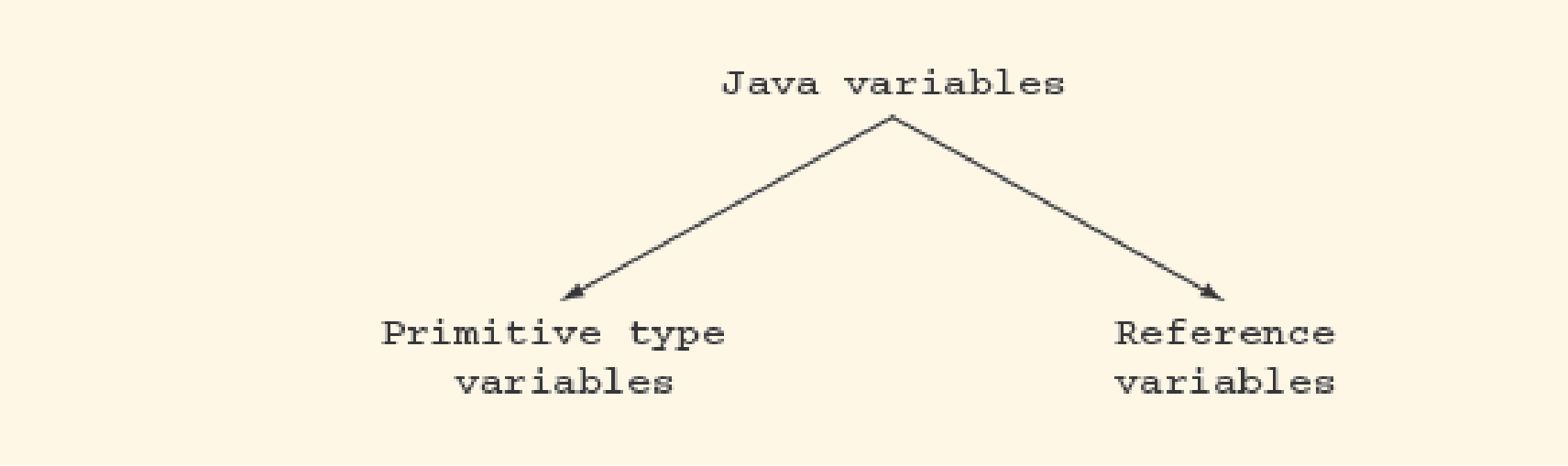


FIGURE 3-6 Java variables

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
```

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: 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
```

```
//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
```

```
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("123456789012345678 9"
                           + "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:

```
12345678901234567890123456789O
  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
```

```
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("123456789012345678 9"
                           + "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:

```
12345678901234567890123456789O
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`

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