Chapter 2
Java basics

MATHEMATICAL FUNCTIONS, CHARACTERS, AND STRINGS

1. Common Mathematical Functions

Trigonometric Methods

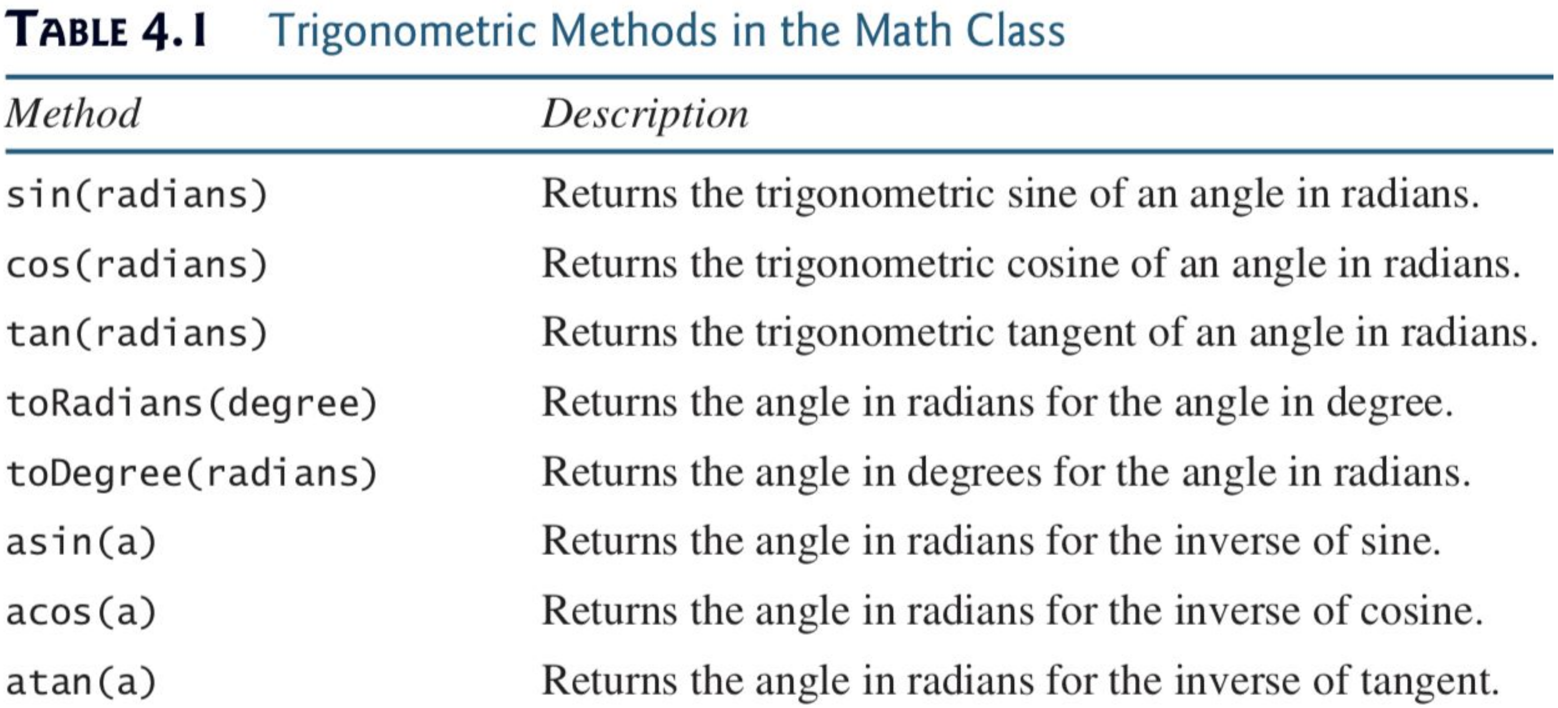


TABLE 4.1 Trigonometric Methods in the Math Class

Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.

1. Common Mathematical Functions

Exponent Methods

```
Math.exp(1);
Math.log(Math.E);
Math.log10(10);
Math.pow(2, 3);
Math.pow(3, 2);
Math.pow(4.5, 2.5);
Math.sqrt(4);
Math.sqrt(10.5);
```

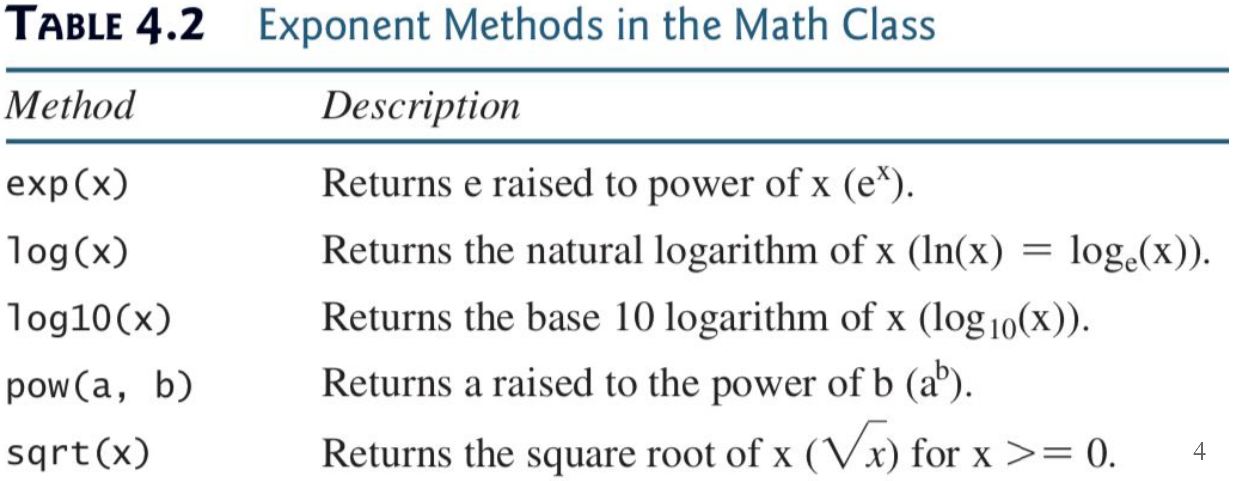


TABLE 4.2 Exponent Methods in the Math Class

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for x >= 0.

1. Common Mathematical Functions

The Rounding Methods

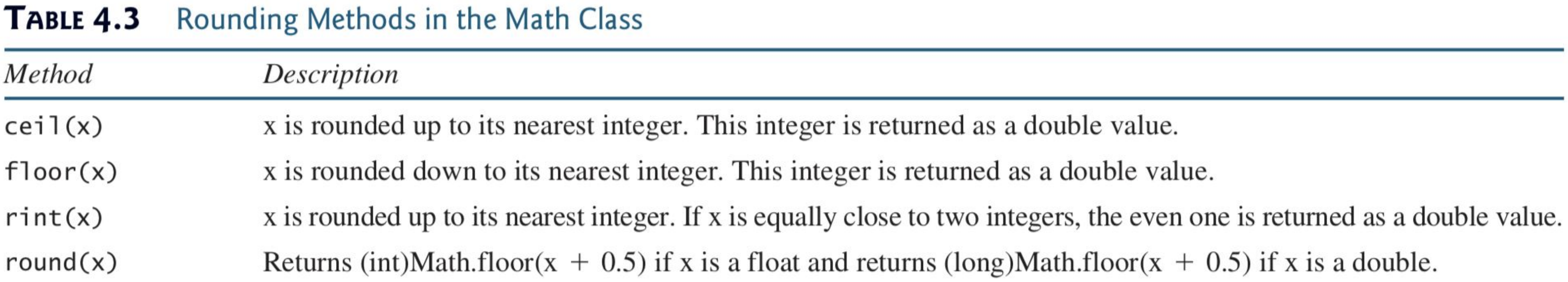
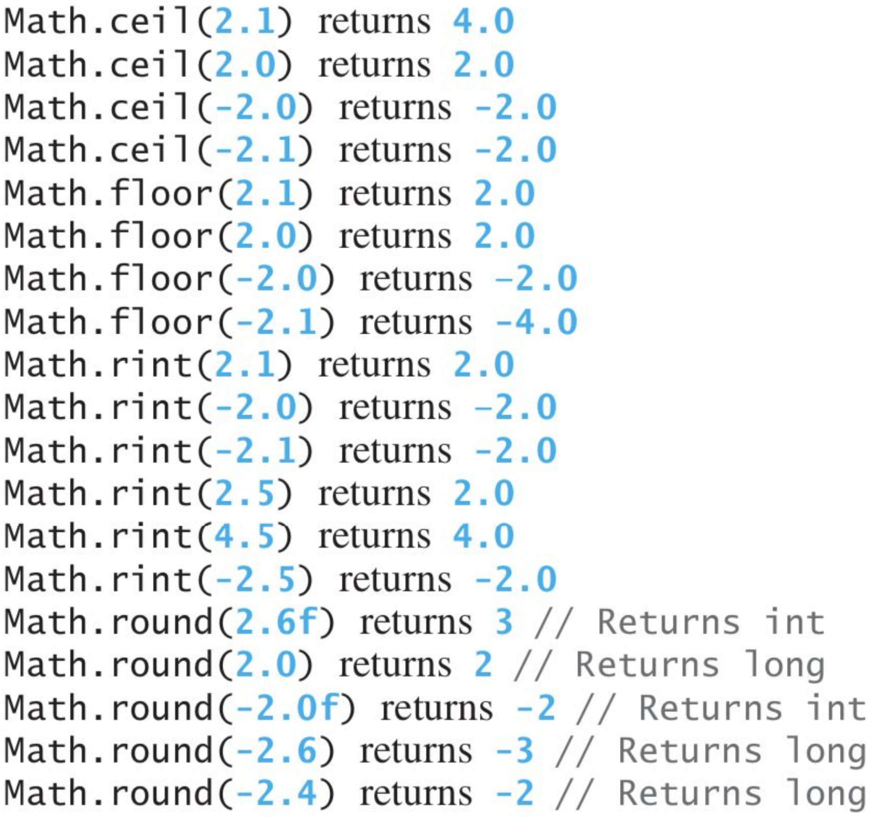


TABLE 4.3 Rounding Methods in the Math Class

Method	*Description*
`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.
`rint(x)`	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value.
`round(x)`	Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double.

1. Common Mathematical Functions

The Rounding Methods



```
Math.ceil(2.1) returns 4.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -4.0
Math.rint(2.1) returns 2.0
Math.rint(-2.0) returns -2.0
Math.rint(-2.1) returns -2.0
Math.rint(2.5) returns 2.0
Math.rint(4.5) returns 4.0
Math.rint(-2.5) returns -2.0
Math.round(2.6f) returns 3 // Returns int
Math.round(2.0) returns 2 // Returns long
Math.round(-2.0f) returns -2 // Returns int
Math.round(-2.6) returns -3 // Returns long
Math.round(-2.4) returns -2 // Returns long
```

1. Common Mathematical Functions

The min, max, and abs Methods

Math.max(2, 3) returns 3

Math.max(2.5, 3) returns 4.0

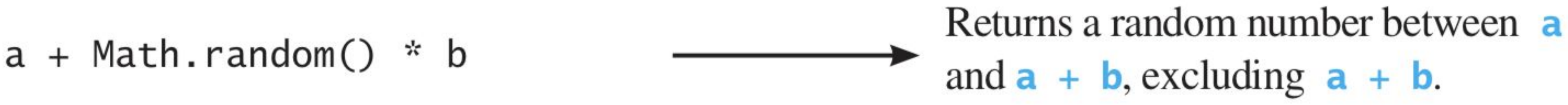
Math.min(2.5, 4.6) returns 2.5

Math.abs(-2) returns 2

Math.abs(-2.1) returns 2.1

1. Common Mathematical Functions

The random Method: generates a random double value greater than or equal to 0.0 and less than 1.0 (**0 <= Math.random() < 1.0**).



```
a + Math.random() * b
```

→ Returns a random number between a and a + b, excluding a + b.

2. Character Data Type and Operations

A character data type represents a single character.

Unicode and ASCII code

Mapping a character to its binary representation is called encoding.

Unicode was originally designed as a 16-bit character encoding.

ASCII (American Standard Code for Information Interchange), an 8-bit encoding scheme for representing all uppercase and lowercase letters, digits, punctuation marks, and control characters.

2. Character Data Type and Operations

Escape Sequences for Special Characters

```
System.out.println("He said \"Java is fun\""); // He said "Java is fun"
```

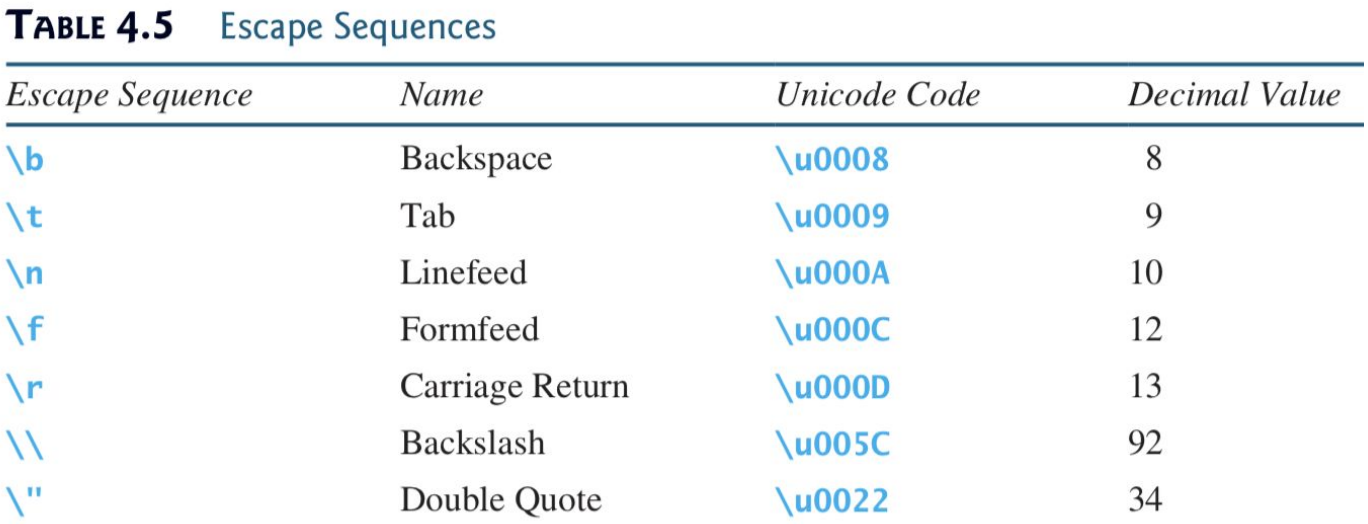


TABLE 4.5 Escape Sequences

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\	Backslash	\u005C	92
\"	Double Quote	\u0022	34

2. Character Data Type and Operations

Casting between char and Numeric Types

```
int i = (int)'A'; // The Unicode of character A is assigned to i
System.out.println(i); // i is 65

char ch = (char)65.25; // Decimal 65 is assigned to ch
System.out.println(ch); // ch is character A
```

2. Character Data Type and Operations

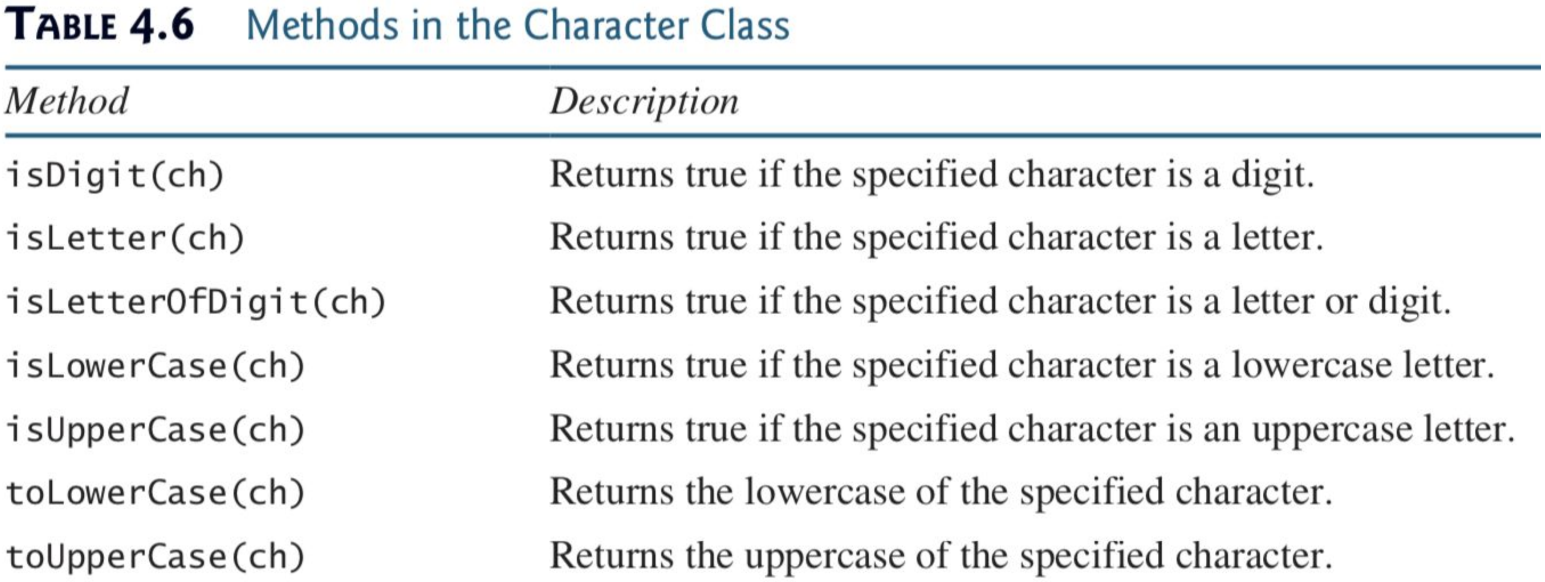


TABLE 4.6 Methods in the Character Class

Method	*Description*
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOfDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

```
System.out.println("isDigit('a') is " + Character.isDigit('a'));
System.out.println("isLetter('a') is " + Character.isLetter('a'));
```

3. The String Type

A string is a sequence of characters.

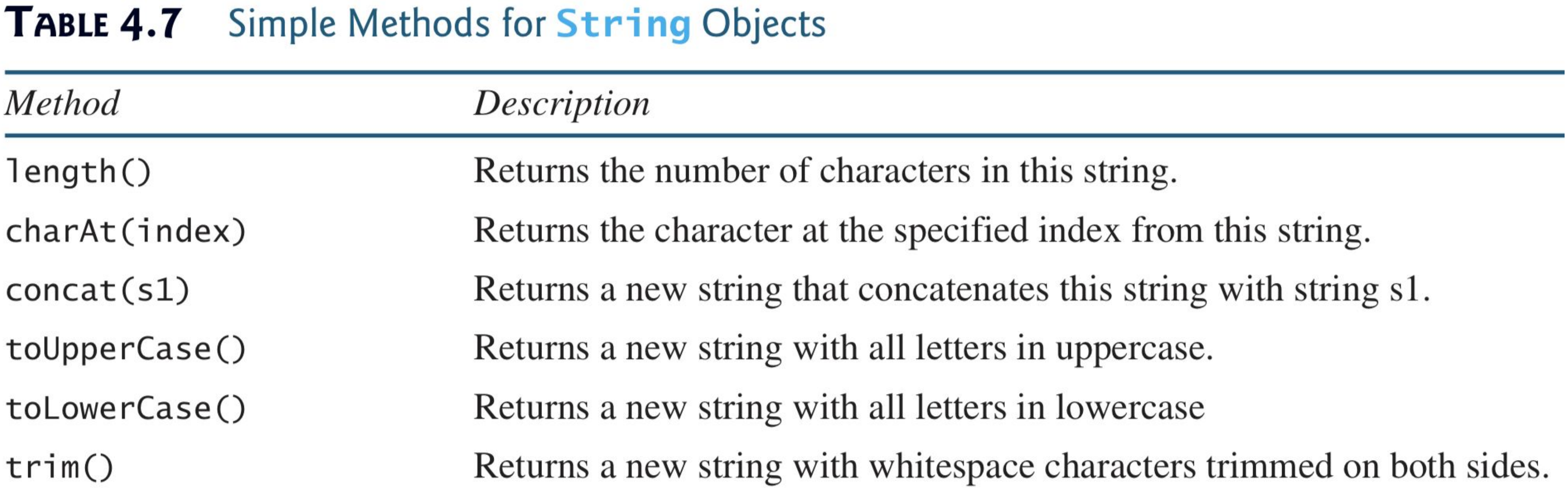


TABLE 4.7 Simple Methods for `String` Objects

Method	*Description*
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

3. The String Type

Getting String Length

```
String message = "Welcome to Java";
System.out.println("The length of " + message + " is " + message.length());
```

Getting Characters from a String

```
System.out.println(message.charAt(0));
```

Concatenating Strings

```
String s1="s1 ", s2="s2 ", s3;
s3 = s1.concat(s2);
s3 = s1 + s2;
```

4. Converting Strings

"Welcome".toLowerCase() returns a new string welcome.
"Welcome".toUpperCase() returns a new string WELCOME.

"\t Good Night \n".trim() returns a new string Good Night.

5. Reading a String from the Console

next() method reads a string that ends with a whitespace character

```
 Scanner input = new Scanner(System.in);
 System.out.print("Enter three words separated by spaces: ");
 String s1 = input.next();
 String s2 = input.next();
 String s3 = input.next();
 System.out.println("s1 is " + s1);
 System.out.println("s2 is " + s2);
 System.out.println("s3 is " + s3);
```

nextLine() method to read an entire line of text

```
Scanner input = new Scanner(System.in);
System.out.println("Enter a line: ");
String s = input.nextLine();
System.out.println("The line entered is " + s);
```

6. Comparing Strings

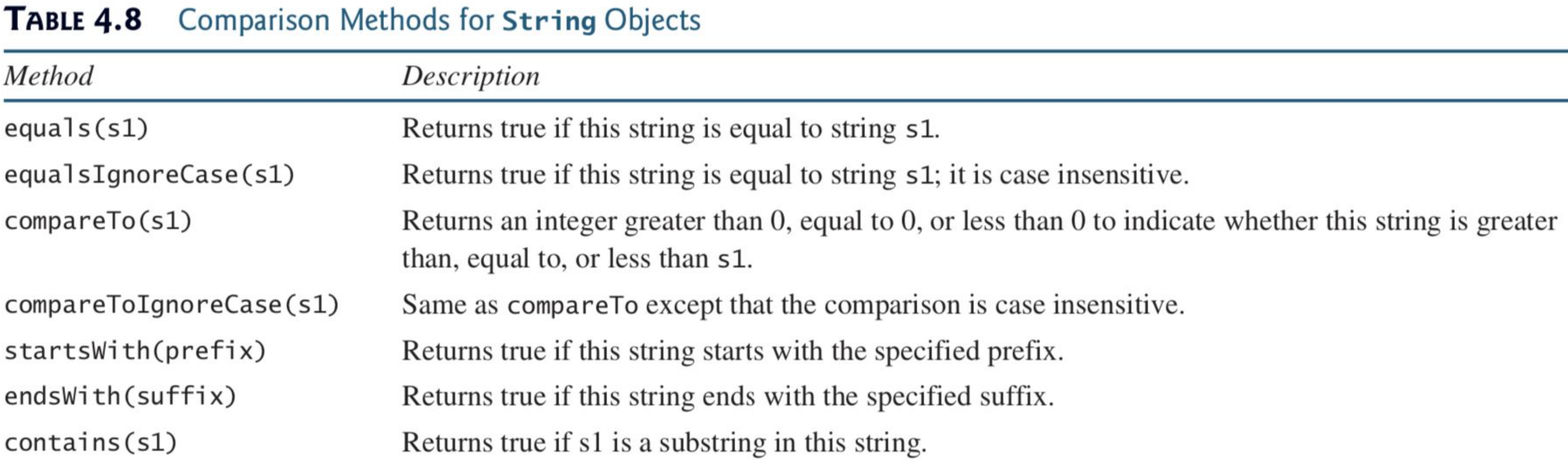


TABLE 4.8 Comparison Methods for `String` Objects

Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`contains(s1)`	Returns true if s1 is a substring in this string.

6. Comparing Strings

==: same objects?

```
if (string1 == string2)
    System.out.println("string1 and string2 are the same object");
else
    System.out.println("string1 and string2 are different objects");
```

string1.equals(string2): same contents?

```
if (string1.equals(string2))
    System.out.println("string1 and string2 have the same contents");
else
    System.out.println("string1 and string2 are not equal");
```

6. Comparing Strings

- s1.compareTo(s2): compare two strings, return the offset of **the first two distinct characters** in s1 and s2 from left to right.
  s1="abc"; s2="abg";
  s1.compareTo(s2) returns -4

- Example: OrderTwoCities.java

7. Obtaining Substrings

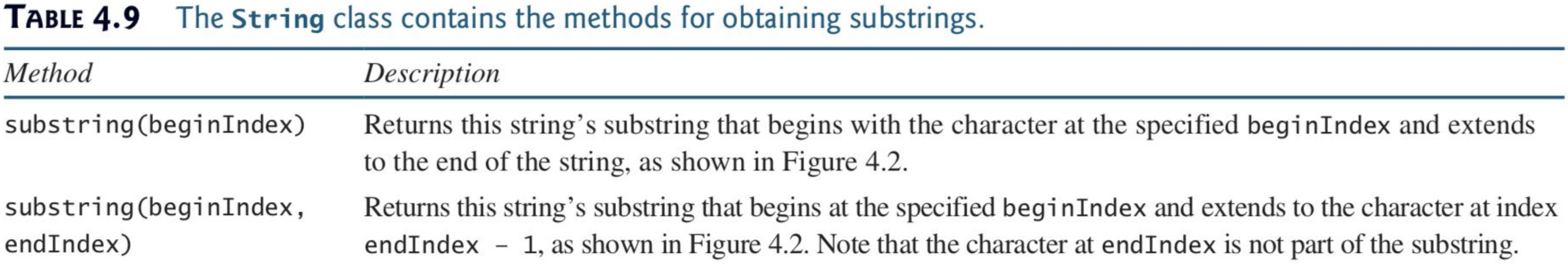


TABLE 4.9 The **String** class contains the methods for obtaining substrings.

Method	*Description*
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex – 1`, as shown in Figure 4.2. Note that the character at `endIndex` is not part of the substring.

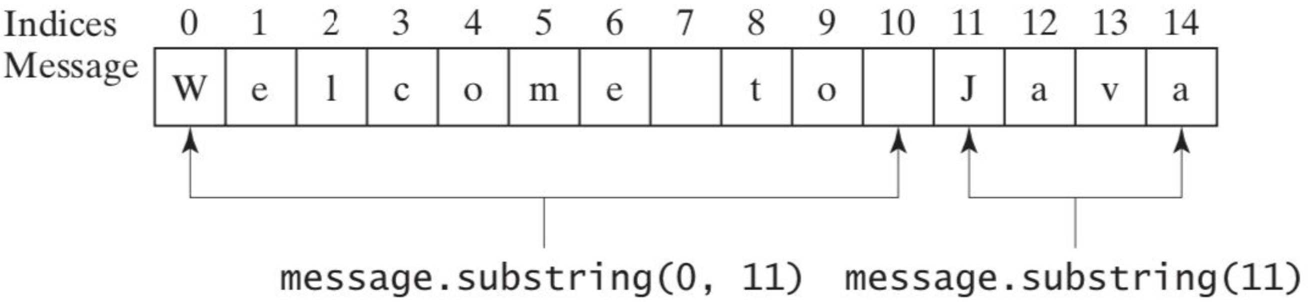


FIGURE 4.2 The **substring** method obtains a substring from a string.

8. Finding a Character or a Substring in a String

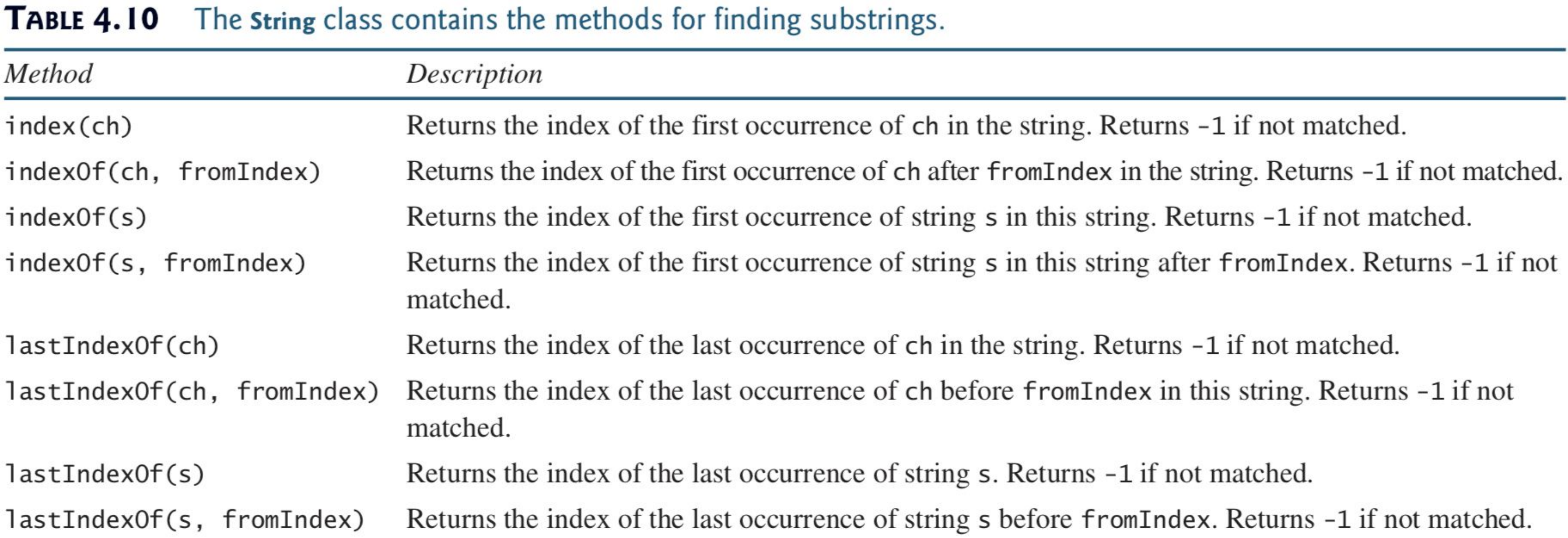


TABLE 4.10 The **String** class contains the methods for finding substrings.

Method	*Description*
`index(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

8. Finding a Character or a Substring in a String

"Welcome to Java".indexOf('W') returns 0.

"Welcome to Java".indexOf('o') returns 4.

"Welcome to Java".indexOf('o', 5) returns 9.

"Welcome to Java".indexOf("come") returns 3.

"Welcome to Java".indexOf("Java", 5) returns 11.

"Welcome to Java".indexOf("java", 5) returns -1.

"Welcome to Java".lastIndexOf('W') returns 0.

"Welcome to Java".lastIndexOf('o') returns 9.

"Welcome to Java".lastIndexOf('o', 5) returns 4.

"Welcome to Java".lastIndexOf("come") returns 3.

"Welcome to Java".lastIndexOf("Java", 5) returns -1.

"Welcome to Java".lastIndexOf("Java") returns 11.

9. Conversion between Strings and Numbers

String to number

```
    int intValue = Integer.parseInt(intString);
    double doubleValue = Double.parseDouble(doubleString);
```

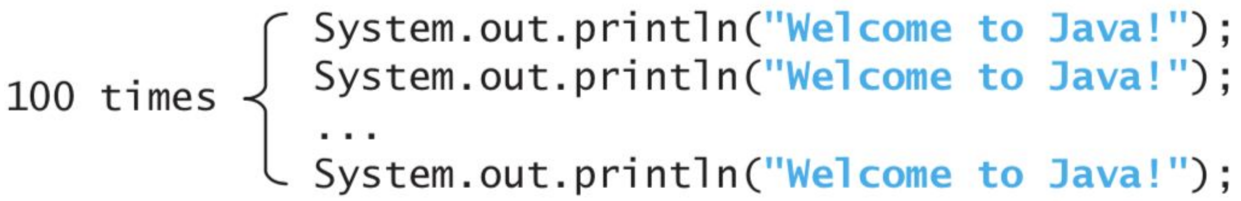
Number to string

```
    String s = number + "";
```

LOOPS

1 Introduction

A loop can be used to tell a program to execute statements repeatedly.



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

2 The while Loop

A while loop executes statements repeatedly while the condition is true.

Syntax:

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

2 The while Loop

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

Example:

RepeatAdditionQuiz.java

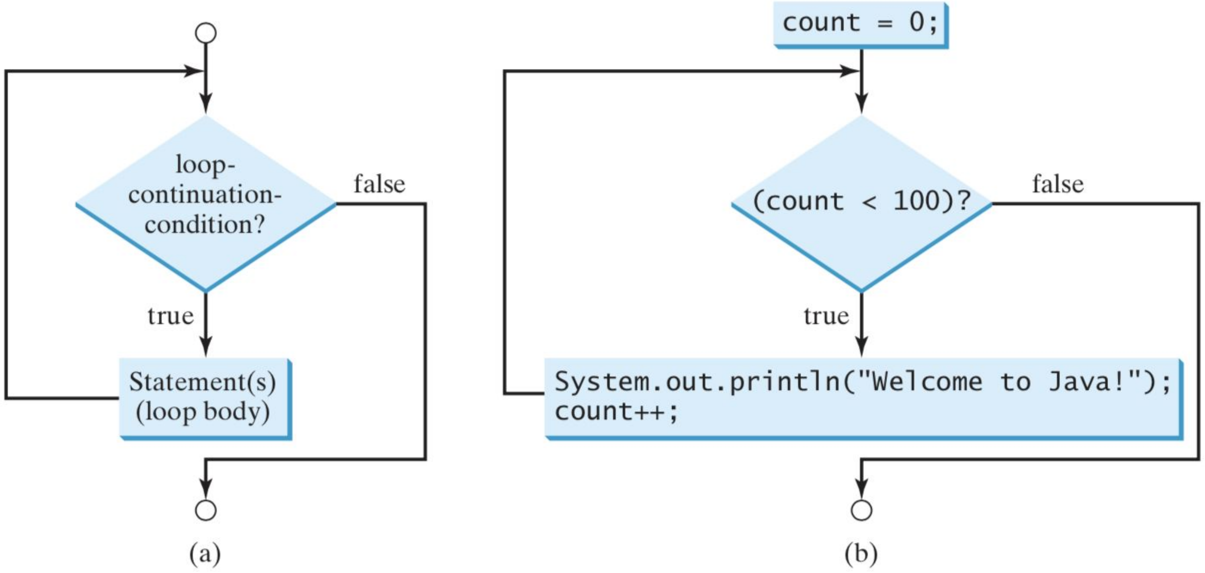
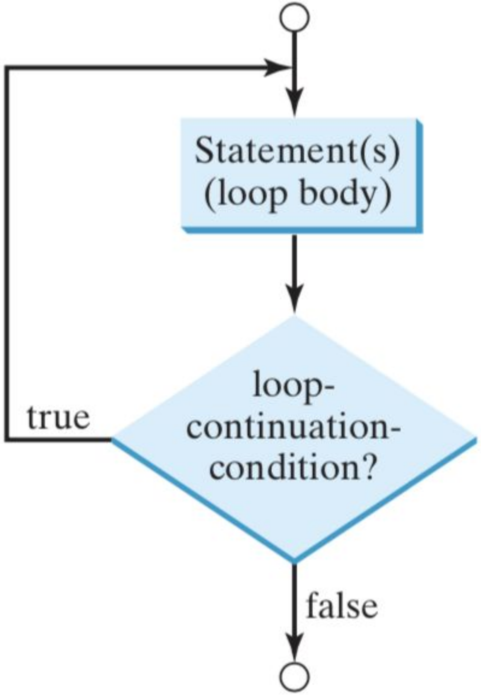


FIGURE 5.1 The while loop repeatedly executes the statements in the loop body when the loop-continuation-condition evaluates to true.

2 The while Loop

- A do-while loop executes the loop body first and then checks the loop continuation condition.
- Syntax:

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



2 The while Loop

TestDoWhile example

```
public static void main(String[] args) {
       int data;
       int sum = 0;
       // Create a Scanner
       Scanner input = new Scanner(System.in);
       // Keep reading data until the input is 0
       do {
           // Read the next data
           System.out.print(
                   "Enter an integer (the input ends if it is 0): ");
           data = input.nextInt();
           sum += data;
       } while (data != 0);
       System.out.println("The sum is " + sum);
   }
```

4 The for Loop

Syntax:

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

A **for** loop generally uses a **variable** (i.e., control variable) to control how many times the loop body is executed and when the loop terminates.

- initial-action often initializes a control variable,
- action-after-each-iteration usually increments or decrements the control variable,
- loop-continuation-condition tests whether the control variable has reached a termination value.
- initial-action, action-after-each-iteration can be a list of zero or more comma-separated statements

4 The for Loop

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

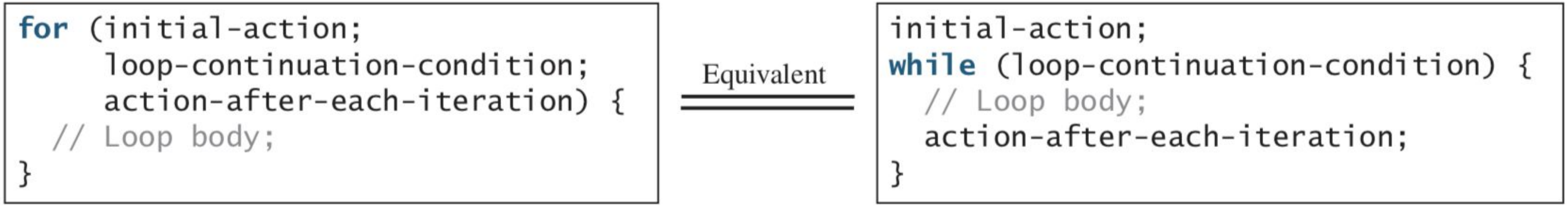
```
for (int i = 0, j = 0; i + j < 10; i++, j++) { // Do something

}
```

5 Which Loop to Use?

while and **for** loop are called **pretest** loops because the continuation condition is checked before the loop body is executed.

do-while loop is called a posttest loop because the condition is checked after the loop body is executed.

while, do-while, and fo: expressively equivalent;



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

Equivalent

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

(a)

(b)

5 Which Loop to Use?

- **for** may be used if the number of repetitions is known in advance, as, for example, when you need to display a message a hundred times.
- **while** may be used if the number of repetitions is not fixed, as in the case of reading the numbers until the input is 0.
- **do-while** can be used to replace a while loop if the loop body has to be executed before the continuation condition is tested.

6 Nested Loops

A loop can be nested inside another loop.

```java
public class MultiplicationTable {
   public static void main(String[] args) {
       // Display the table heading
       System.out.println("           Multiplication Table");
       // Display the number title
       System.out.print("    ");
       for (int j = 1; j <= 9; j++)
         System.out.print("   " + j);
       System.out.println("\n------------------------------")
       for (int i = 1; i <= 9; i++) {
         System.out.print(i + " | ");
         for (int j = 1; j <= 9; j++) {
           // Display the product and align properly
           System.out.printf("%4d", i * j);
         }
         System.out.println();
       }
     }
}
```

7 Minimizing Numeric Errors

- Using floating-point numbers in the loop continuation condition may cause numeric errors.
- Control variable should be integer.

8 Keywords break and continue

break: immediately terminate the loop.

```
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);
     }
}
```

8 Keywords break and continue

continue: ends the current iteration and program control goes to the end of the loop body

```
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);
     }
}
```