Methods

Introduction

Methods can be used to define reusable code and organize and simplify coding.

Suppose that you need to find the sum of integers

from 1 to 10

from 20 to 37

from 35 to 49

You may write the code as follows:

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 37; i++)
  sum += i;
System.out.println("Sum from 20 to 37 is " + sum);

sum = 0;
for (int i = 35; i <= 49; i++)
  sum += i;
System.out.println("Sum from 35 to 49 is " + sum);
```

The preceding code can be simplified as follows:

```
1  public static int sum(int i1, int i2) {
2    int result = 0;
3    for (int i = i1; i <= i2; i++)
4      result += i;
5
6    return result;
7  }
8
9  public static void main(String[] args) {
10   System.out.println("Sum from 1 to 10 is " + sum(1, 10));
11   System.out.println("Sum from 20 to 37 is " + sum(20, 37));
12   System.out.println("Sum from 35 to 49 is " + sum(35, 49));
13 }
```

Defining a Method

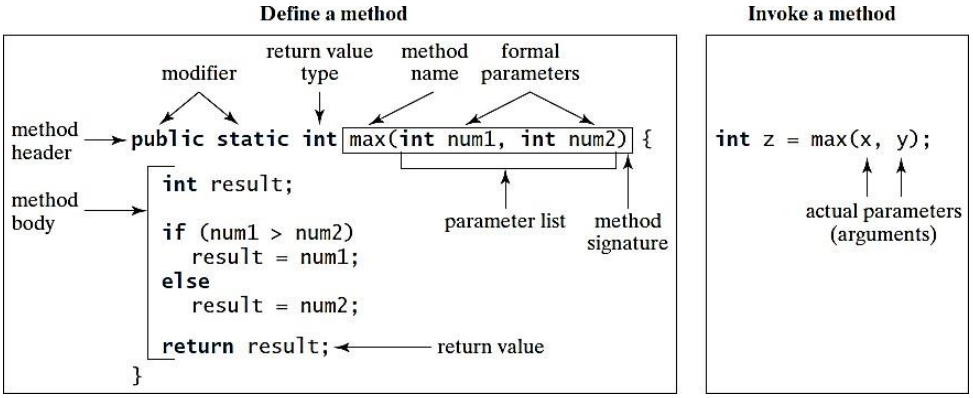
A method definition consists of

- **Method name**
- **Parameters**
- **Return value type**
- **Method body.**

The syntax for defining a method is as follows:

```
modifier   returnValueType   methodName   (list of parameters) {
    // Method body;
}
```

For example:



The components of this method,

named max, has two int parameters, num1 and num2 and the larger of which is returned by the method.

Note:

- **The method header specifies the modifiers, return value type, method name, and parameters of the method.**
- **A method may return a value. The returnValueType is the data type of the value the method returns.**

- **Some methods perform desired operations without returning a value. In this case, the returnValueType is the keyword void. For example, the returnValueType is void in the main method, as well as in System.exit, and System.out.println.**
- **If a method returns a value, it is called a *value-returning method;* otherwise it is called a *void method.***
- **The variables defined in the method header are known as *parameters.***
- **When a method is invoked (called), you pass a value to the parameter. This value is referred to as an *actual parameter or argument.* The *parameter list* refers to the method's type, order, and number of the parameters. The method name and the parameter list together constitute the *method signature.***

Calling (invoke) a Method

To execute the method, you have to *call* or *invoke* it.

There are two ways to call a method:

1- **If a method returns a value, a call to the method is usually treated as a value (i.e. using assignment operation). For example,**

```
int larger = max (3, 4);
```

calls max(3, 4) and assigns the result of the method to the variable larger.

Or call that is treated as a value (i.e. using print the method)

```
System.out.println (max(3, 4));
```

which prints the return value of the method call max(3, 4).

2- **If a method returns void, a call to the method must be a statement.**

For example, the

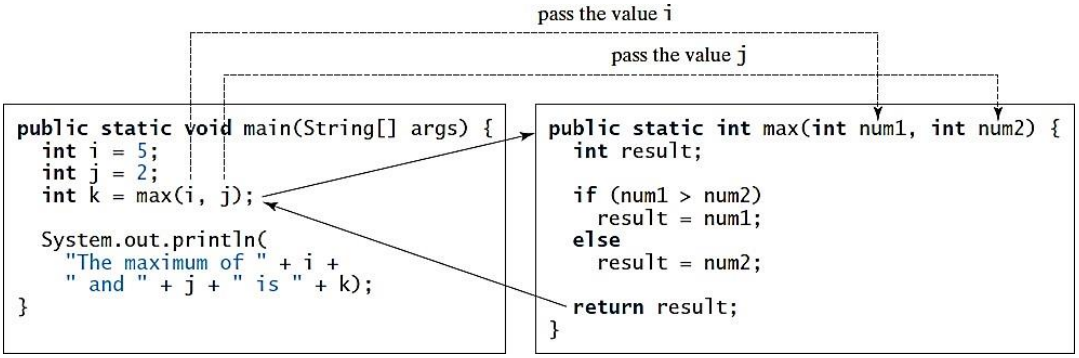
method (println) returns void. The following call is a statement:

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

min (3, 4);

random(10) ;
```

value-returning Method Example

LISTING 6.1 TestMax.java

```
1  public class TestMax {
2    /** Main method */
3    public static void main(String[] args) {
4      int i = 5;
5      int j = 2;
6      int k = max(i, j);
7      System.out.println("The maximum of " + i +
8        " and " + j + " is " + k);
9    }
10
11   /** Return the max of two numbers */
12   public static int max(int num1, int num2) {
13     int result;
14
15     if (num1 > num2)
16       result = num1;
17     else
18       result = num2;
19
20     return result;
21   }
22 }
```



Write class (program) named *FindGrad* to print the grade of student. The class have method named (getGrad) have one double parameter and return value of char datatype ('A','B',…,'F')

LISTING 6.3 TestReturnGradeMethod.java

```
1  public class FindGrade          {
2    public static void main(String[] args) {
3      System.out.print("The grade is " + getGrade(78.5));
4      System.out.print("\nThe grade is " + getGrade(59.5));
5    }
6
7    public static char getGrade(double score) {
8      if (score >= 90.0)
9        return 'A';
10     else if (score >= 80.0)
11       return 'B';
12     else if (score >= 70.0)
13       return 'C';
14     else if (score >= 60.0)
15       return 'D';
16     else
17       return 'F';
18   }
19 }
```

Define method named (sign) have one integer parameter and return value (1, 0, or -1)

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```

void Method Example

A void method does not return a value.

Example

LISTING 6.2 TestVoidMethod.java

```
1  public class TestVoidMethod {
2    public static void main(String[] args) {
3      System.out.print("The grade is ");
4      printGrade(78.5);
5
6      System.out.print("The grade is ");
7      printGrade(59.5);
8    }
9
10   public static void printGrade(double score) {
11     if (score >= 90.0) {
12       System.out.println('A');
13     }
14     else if (score >= 80.0) {
15       System.out.println('B');
16     }
17     else if (score >= 70.0) {
18       System.out.println('C');
19     }
20     else if (score >= 60.0) {
21       System.out.println('D');
22     }
23     else {
24       System.out.println('F');
25     }
26   }
27 }
```

Overloading Methods

Overloading methods means define multi methods with the same name and different signatures (i.e. create two or more methods with the same name but different parameters)

LISTING 6.9 TestMethodOverloading.java

```
1  public class TestMethodOverloading {
2    /** Main method */
3    public static void main(String[] args) {
4      // Invoke the max method with int parameters
5      System.out.println("The maximum of 3 and 4 is "
6        + max(3, 4));
7
8      // Invoke the max method with the double parameters
9      System.out.println("The maximum of 3.0 and 5.4 is "
10       + max(3.0, 5.4));
11
12     // Invoke the max method with three double parameters
13     System.out.println("The maximum of 3.0, 5.4, and 10.14 is "
14       + max(3.0, 5.4, 10.14));
15   }
16
17   /** Return the max of two int values */
18   public static int max(int num1, int num2) {
19     if (num1 > num2)
20       return num1;
21     else
22       return num2;
23   }
24
25   /** Find the max of two double values */
26   public static double max(double num1, double num2) {
27     if (num1 > num2)
28       return num1;
29     else
30       return num2;
31   }
32
33   /** Return the max of three double values */
34   public static double max(double num1, double num2, double num3) {
35     return max(max(num1, num2), num3);
36   }
37 }
```

- **When calling max(3, 4) (line 6), the max method for finding the maximum of two integers is invoked.**
- **When calling max(3.0, 5.4) (line 10), the max method for finding the maximum of two doubles is invoked.**
- **When calling max(3.0, 5.4, 10.14) (line 14), the max method for finding the maximum of three double values is invoked.**

Ambiguous Invocation.

Ambiguous invocation causes a compile error. Consider the following code:

```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

  public static double max(int num1, double num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }

  public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }
}
```

Both max (int, double) and max (double, int) are possible candidates to match max(1, 2). Because neither is better than the other, the invocation is ambiguous, resulting in a compile error.

Given two method definitions,

public static double m (double x, double y)
public static double m (int x, double y)

Tell which of the two methods is invoked for:

a. **double** z = m(4, 5);
b. **double** z = m(4, 5.4);
c. **double** z = m(4.5, 5.4);

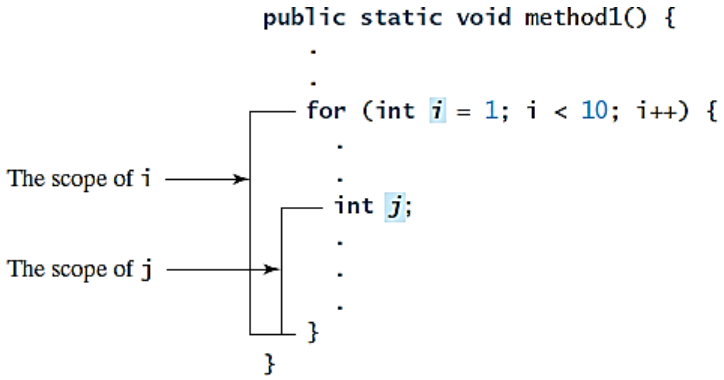
The Scope of Variables

The scope of a variable is the part of the program where the variable can be referenced.

- **A block is begun with an opening curly brace ({) and ended by a closing curly brace (}).**
- **A block defines a *scope*. Thus, each time you start a new block, you are creating a new scope.**

- **A scope determines what objects are visible to other parts of your program. It also determines the lifetime of those objects.**

- **Variables declared inside a scope are not visible (cannot access and/or modification) to code that is defined outside that scope.**



For example

```
// block scope.
class Scope {
  public static void main(String args[])
{
    int x;              // x is global variable (known to all code within main method block)
    x = 10;
    if (x == 10)
      {                 // Start new scope
        int y = 20;     // y is local variable( known only to if  block)

       // x and y both known here.

        x = y * 2;
      }                 // End new scope

   y = 100;             // Error! y not known here Here, y is outside of its  scope.

 // x is still known here.
  System.out.println("x is " + x);
  }
}
```

- **A *global variable is a* variable defined inside a class or main method.**

- **A *local variable is a* variable defined inside a method, if, for. etc.**

- **Within a block, variables can be declared at any point, but are valid only after they are declared. Thus, if you define a variable at the start of a method, it is available to all of the code within that method.**

- **Variables are created when their scope is entered, and destroyed when their scope is left. This means that a variable will not hold its value once it has gone out of scope. Therefore, variables declared within a method will not hold their values between calls to that method. Also, a variable declared within a block will lose its value when the block is left. Thus, the lifetime of a variable is confined to its scope.**

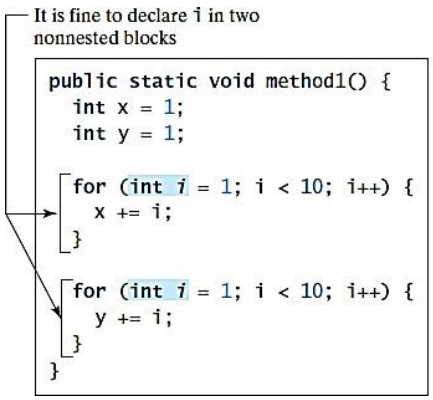
Ex:-

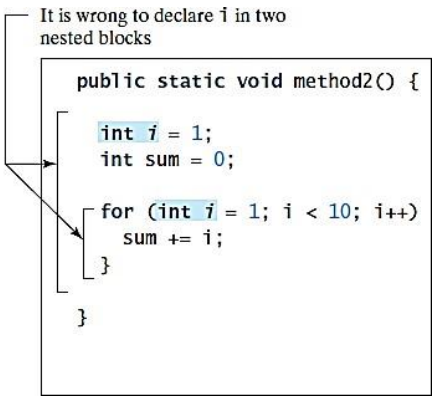
```
class VarInitDemo {
  public static void main(String args[]) {
    int x;
    for (x = 0; x < 3; x++)
    {
      int y = -1;                          // y is initialized each time block is entered
      System.out.println("y is: " + y);     // this always prints -1
      y = 100;
      System.out.println("y is now: " + y);
    }
  }
}
```

Note: - A variable declared in the initial part of a for-loop header has its scope in the entire loop. But a variable declared inside a for-loop body has its scope limited in the loop body from its declaration to the end of the block that contains the variable, as shown in Figure bellow.

Common Error

You can declare a local variable with the same name in different blocks in a method, but you cannot declare a local variable twice in the same block or in nested blocks, as shown in Figure bellow





```
class NestVar {
  public static void main(String args[]) {
    int count;

    for(count = 0; count < 10; count = count+1) {
      System.out.println("This is count: " + count);

      int count; // illegal!!!
      for(count = 0; count < 2; count++)
        System.out.println("This program is in error!");
    }
  }
}
```

Can't declare **count** again because it's already declared.

Common Mathematical Functions

Java provides many useful methods in the Math class for performing common mathematical functions.

The min, max, and abs Methods

The min and max methods return the minimum and maximum numbers of two numbers (int, long, float, or double).

The abs method returns the absolute value of the number (int, long, float, or double).

For example,

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

The random Method

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).

You can use it to write a simple expression to generate random numbers in any range.

For example,

Math.random( )	Returns a random integer between 0.0 and 0.9.

(**int**)(Math.random( ) * **10**) Returns a random integer between 0 and **9**.

50 + (**int**)(Math.random( ) * **50**) Returns a random integer between **50** and **99**.

Exponent Methods

There are five methods related to exponents in the Math class as shown in Table bellow.

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.

Math.exp(**1**) returns **2.71828**

Math.log(Math.E) returns **1.0**

Math.log10(**10**) returns **1.0**

Math.pow(**2**, **3**) returns **8.0**

Math.pow(**3**, **2**) returns **9.0**

Math.pow(**4.5**, **2.5**) returns **22.91765**

Math.sqrt(**4**) returns **2.0**

Math.sqrt(**10.5**) returns **4.24**

The Rounding Methods

The Math class contains five rounding methods as shown in Table 4.3

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.

Unicode and ASCII code

A character is stored in a computer as a sequence of 0s and 1s. Mapping a character to its binary representation is called *encoding*.

Java supports *Ascii* and *Unicode*, an encoding scheme to support processing, and display of written texts.

- **Unicode was originally designed as a 16-bit character encoding.**
- **ASCII code was originally designed as a 8-bit character encoding.**

TABLE 4.4 ASCII Code for Commonly Used Characters

Characters	*Code Value in Decimal*	*Unicode Value*
`'0'` to `'9'`	48 to 57	\u0030 to \u0039
`'A'` to `'Z'`	65 to 90	\u0041 to \u005A
`'a'` to `'z'`	97 to 122	\u0061 to \u007A

Character Datatype

The char type represents only one character

Use single quotation (′ ′) with character

```
char ch = ′ A ′;
char ch2 = ′ c ′;
char ch3 = ′ 4 ′;
char ch4 = ′ + ′;
```

Reading a Character from the Console

To read a character from the console, use the nextLine( ) or next( ) method to read a string and then invoke (call) the charAt(0) method on the string to return a character.

For example, the following

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");

String s = input.nextLine();
char ch = s.charAt(0);

System.out.println("The character entered is " + ch);
```

Shortly you can use the following:

```
char ch = input.nextLine().charAt(0);
```

Casting between char and Numeric Types

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

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

Comparing and Testing Characters

Two characters can be compared using the relational operators just like comparing two numbers. This is done by comparing the Unicodes of the two characters.

For example,

'a' < 'b'	is true because the Unicode for 'a' (97) is less than the Unicode for 'b' (98).
'a' < 'A'	is false because the Unicode for 'a' (97) is greater than the Unicode for 'A' (65).
'1' < '8'	is true because the Unicode for '1' (49) is less than the Unicode for '8' (56).

For example:

```
if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

Class Character

Java have class named Character. The Character class have several method as shown in table below:

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.

For example,

```
System.out.println( Character.isDigit('a'));          // print  false
System.out.println (Character.isLetter('a'));         // print true
System.out.println( Character.isLowerCase('a'));      // print true
System.out.println( Character.isUpperCase('a'));      // print false
System.out.println( Character.toLowerCase('T'));      // print t
System.out.println( Character.toUpperCase('q'));      // print  Q
```

Show the output of the following program:

```
public class Test {
  public static void main(String[ ] args) {
    char x = 'a';
    char y = 'c';
    System.out.println(++x);
    System.out.println(y++);
    System.out.println(x - y);
  }
}
```

The String Type

A string is a sequence of characters.

The char type represents only one character. To represent a string of characters, use the data type called String.

For example, the following code declares message to be a string with the value "Welcome to Java".

```
String message = "Welcome to Java";
```

- **String is a predefined class in the Java library, just like the classes System and Scanner.**
- **The String type is not a primitive type. It is known as a *reference type*.**

Reading a String from the Console

To read a string from the console, invoke the next() or nextLine method on a Scanner object.

- **The next( ) method reads a string that ends with a whitespace character.**
- **The nextLine( ) method to read an entire line of text. The nextLine( ) method reads a string that ends with the *Enter* key pressed**

For example, the following code reads three strings from the keyboard:

```
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);
```

Methods of Class String

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.

Getting String Length

You can use the length() method to return the number of characters in a string. For example, the following code

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

the output is

The length of Welcome to Java is 15

Getting Characters from a String

The charAt(index) method can be used to retrieve a specific character in a string s, where the index is between 0 and s.length( ) –1.

For example,

```
message.charAt(0)          // returns the character W,
```

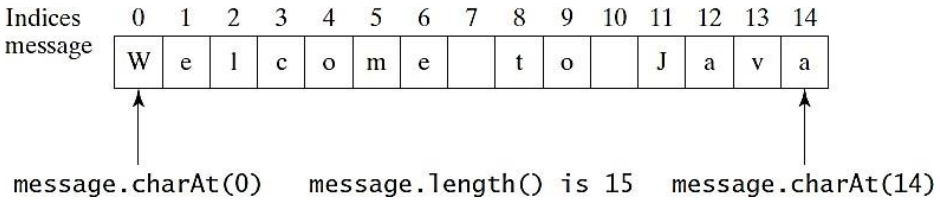


FIGURE 4.1 The characters in a String object can be accessed using its index.

Concatenating Strings

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

For example:

```
String s1="my name";
String s1="is Ali";

System.out.println(s1.concat (s2));          // returns my name is Ali
System.out.println(s1 +  s2);                // returns my name is Ali
```

Converting Strings

- **The toLowerCase() method returns a new string with all lowercase letters**
- **The toUpperCase() method returns a new string with all uppercase letters.**

For example,

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

Comparing Strings

The String class contains the methods as shown in Table 4.8 for comparing two strings.

- **The == operator checks only whether string1 and string2 refer to the same object; it does not tell you whether they have the same contents.**

- **The equals ( ) method is used to compare two string variables have the 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");
```

For example

```
String s1 = "Welcome to Java";
String s2 = "Welcome to Java";
String s3 = "Welcome to C++";

System.out.println(s1.equals(s2));        // true
System.out.println(s1.equals(s3));        // false
```

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.

Obtaining Substrings

- **You can obtain a single character from a string using the charAt( ) method.**
- **You can also obtain a substring from a string using the substring( ) method**

For example,

```
String message = "Welcome to Java";
String message = message.substring(0, 11) + "HTML";
```

The string **message** now becomes **Welcome to HTML**.

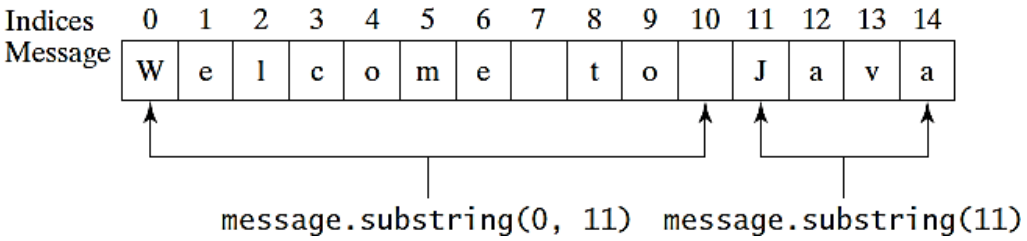
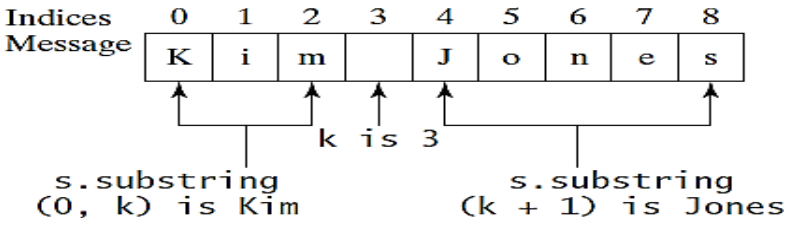


TABLE 4.9 The `String` class contains the methods for obtaining substrings.

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.



Finding a Character or a Substring in a String

The String class provides several versions of indexOf( )and lastIndexOf( ) methods to find 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.

For example:

```
String s ="Welcome to Java";
        s.indexOf('W')              // returns 0.
        s.indexOf('o')              // returns 4.
        s.indexOf('o', 5)           // returns 9.
        s.indexOf("come")           // returns 3.
        s.indexOf("Java", 5)        // returns 11.
        s.indexOf("java", 5)        // returns -1.
```