

Chapter 4 Mathematical Functions, Characters, and Strings



1

Mathematical Functions

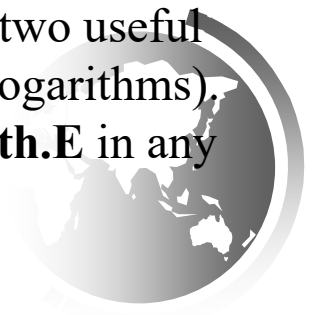
Java provides many useful methods in the **Math** class for performing common mathematical functions.



2

The Math Class

- A method is a group of statements that performs a specific task. You have already used the **pow(a, b)** method to compute a^b and the **random()** method for generating a random number. There are many other useful methods in the **Math** class. They can be categorized as *trigonometric methods*, *exponent methods*, *rounding methods* and *service methods*. Service methods include the rounding, min, max, absolute, and random methods.
- In addition to methods, the **Math** class provides two useful **double** constants, **PI** and **E** (the base of natural logarithms). You can use these constants as **Math.PI** and **Math.E** in any program.



3

The Math Class

- Class constants:
 - PI
 - E
- Class methods:
 - Trigonometric Methods
 - Exponent Methods
 - Rounding Methods
 - min, max, abs, and random Methods



4

Trigonometric Methods

- `sin(double a)`
- `cos(double a)`
- `tan(double a)`
- `acos(double a)`
- `asin(double a)`
- `atan(double a)`

Examples:

`Math.sin(0)` returns 0.0

`Math.sin(Math.PI / 6)`
returns 0.5

`Math.sin(Math.PI / 2)`
returns 1.0

`Math.cos(0)` returns 1.0

`Math.cos(Math.PI / 6)`
returns 0.866

`Math.cos(Math.PI / 2)`
returns 0

Exponent Methods

- `exp(double a)`
Returns e raised to the power of a.
- `log(double a)`
Returns the natural logarithm of a.
- `log10(double a)`
Returns the 10-based logarithm of a.
- `pow(double a, double b)`
Returns a raised to the power of b.
- `sqrt(double a)`
Returns the square root of a.

Examples:

`Math.exp(1)` returns 2.71

`Math.log(2.71)` returns 1.0

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

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

`Math.pow(3.5, 2.5)` returns
22.91765

`Math.sqrt(4)` returns 2.0

`Math.sqrt(10.5)` returns 3.24

Rounding Methods

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



7

Rounding Methods Examples

```
Math.ceil(2.1) returns 3.0
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 -3.0
Math.rint(2.1) returns 2.0
Math.rint(2.0) returns 2.0
Math.rint(-2.6) returns -3.0
Math.rint(-2.1) returns -2.0
Math.rint(2.5) returns 2.0
Math.rint(2.501) returns 3.0
Math.rint(-2.5) returns -2.0
Math.round(2.5) returns 3
Math.round(2.501) returns 3
Math.round(2.0) returns 2
Math.round(-2.4) returns -2
Math.round(-2.6) returns -3
```



8

min, max, and abs

- `max(a, b)` and `min(a, b)`
Returns the maximum or minimum of two parameters.
- `abs(a)`
Returns the absolute value of the parameter.
- `random()`
Returns a random double value in the range $[0.0, 1.0)$.

Examples:

`Math.max(2, 3)` returns 3

`Math.max(2.5, 3)` returns 3.0

`Math.min(2.5, 3.6)`
returns 2.5

`Math.abs(-2)` returns 2

`Math.abs(-2.1)` returns 2.1



9

The random Method

Generates a random double value greater than or equal to 0.0 and less than 1.0 ($0 \leq \text{Math.random()} < 1.0$).

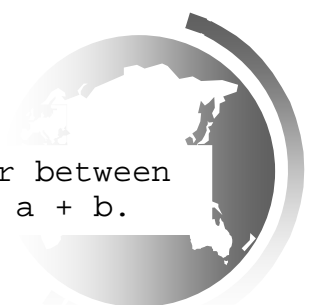
Examples:

`(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.

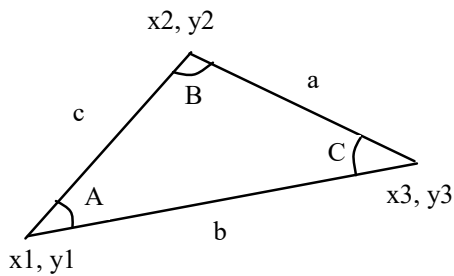
In general,

`a + Math.random() * b` → Returns a random number between a and a + b, excluding a + b.



10

Case Study: Computing Angles of a Triangle



$$A = \arccos\left(\frac{a^2 - b^2 - c^2}{-2bc}\right)$$
$$B = \arccos\left(\frac{b^2 - a^2 - c^2}{-2ac}\right)$$
$$C = \arccos\left(\frac{c^2 - b^2 - a^2}{-2ab}\right)$$

Write a program that prompts the user to enter the x- and y-coordinates of the three corner points in a triangle and then displays the triangle's angles.



11

```
import java.util.Scanner;

public class ComputeAngles {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter three points
        System.out.print("Enter three points: ");
        double x1 = input.nextDouble();
        double y1 = input.nextDouble();
        double x2 = input.nextDouble();
        double y2 = input.nextDouble();
        double x3 = input.nextDouble();
        double y3 = input.nextDouble();

        // Compute three sides
        double a = Math.sqrt((x2 - x3) * (x2 - x3)
            + (y2 - y3) * (y2 - y3));
        double b = Math.sqrt((x1 - x3) * (x1 - x3)
            + (y1 - y3) * (y1 - y3));
        double c = Math.sqrt((x1 - x2) * (x1 - x2)
            + (y1 - y2) * (y1 - y2));

        // Compute three angles
        double A = Math.toDegrees(Math.acos((a * a - b * b - c * c)
            / (-2 * b * c)));
        double B = Math.toDegrees(Math.acos((b * b - a * a - c * c)
            / (-2 * a * c)));
        double C = Math.toDegrees(Math.acos((c * c - b * b - a * a)
            / (-2 * a * b)));

        // Display results
        System.out.println("The three angles are " +
            Math.round(A * 100) / 100.0 + " " +
            Math.round(B * 100) / 100.0 + " " +
            Math.round(C * 100) / 100.0);
    }
}
```



12

Character Data Type

Four hexadecimal digits.

```
char letter = 'A'; (ASCII)
```

```
char numChar = '4'; (ASCII)
```

```
char letter = '\u0041'; (Unicode)
```

```
char numChar = '\u0034'; (Unicode)
```

NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character. For example, the following statements display character b.

```
char ch = 'a';
```

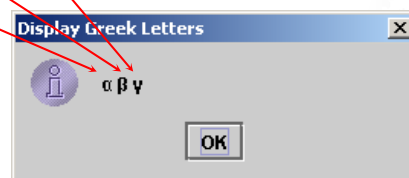
```
System.out.println(++ch);
```

13

Unicode Format

Java characters use *Unicode*, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world's diverse languages. Unicode takes two bytes, preceded by `\u`, expressed in four hexadecimal numbers that run from `\u0000` to `\uFFFF`. So, Unicode can represent $65535 + 1$ characters.

Unicode `\u03b1` `\u03b2` `\u03b3` for three Greek letters



14

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 |



15

Escape Sequences for Special Characters

| <i>Escape Sequence</i> | <i>Name</i> | <i>Unicode Code</i> | <i>Decimal Value</i> |
|------------------------|-----------------|---------------------|----------------------|
| \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 |



16

Casting between char and Numeric Types

A char can be cast into any numeric type and vice versa.

```
int i = 'a'; // Same as int i = (int)'a';
```

```
char c = 97; // Same as char c = (char)97;
```



17

Comparing and Testing Characters

```
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");
```



18

Methods in the Character Class

| Method | Description |
|----------------------------------|---|
| <code>isDigit(ch)</code> | Returns true if the specified character is a digit. |
| <code>isLetter(ch)</code> | Returns true if the specified character is a letter. |
| <code>isLetterOfDigit(ch)</code> | Returns true if the specified character is a letter or digit. |
| <code>isLowerCase(ch)</code> | Returns true if the specified character is a lowercase letter. |
| <code>isUpperCase(ch)</code> | Returns true if the specified character is an uppercase letter. |
| <code>toLowerCase(ch)</code> | Returns the lowercase of the specified character. |
| <code>toUpperCase(ch)</code> | Returns the uppercase of the specified character. |



19

Methods in the Character Class

```
System.out.println("isDigit('a') is " + Character.isDigit('a'));
System.out.println("isLetter('a') is " + Character.isLetter('a'));
System.out.println("isLowerCase('a') is "
    + Character.isLowerCase('a'));
System.out.println("isUpperCase('a') is "
    + Character.isUpperCase('a'));
System.out.println("toLowerCase('T') is "
    + Character.toLowerCase('T'));
System.out.println("toUpperCase('q') is "
    + Character.toUpperCase('q'));
```

displays

```
isDigit('a') is false
isLetter('a') is true
isLowerCase('a') is true
isUpperCase('a') is false
toLowerCase('T') is t
toUpperCase('q') is Q
```



20

The String Type

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

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

String is actually a predefined class in the Java library just like the System class and Scanner class.

The **String** type is not a primitive type. It is known as a *reference type*. Any Java class can be used as a reference type for a variable. The variable declared by a reference type is known as a **reference variable** that references an object. Here, **message** is a reference variable that references a string object with contents **Welcome to Java**.



21

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. |



22

Simple Methods for String Objects

Strings are objects in Java. The methods in the preceding table can only be invoked from a specific string instance. For this reason, these methods are called *instance methods*. A non-instance method is called a *static method*. A static method can be invoked without using an object. All the methods defined in the **Math** class are static methods. They are not tied to a specific object instance. The syntax to invoke an instance method is

referenceVariable.methodName(arguments).



23

Getting String Length

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

displays

The length of Welcome to Java is 15



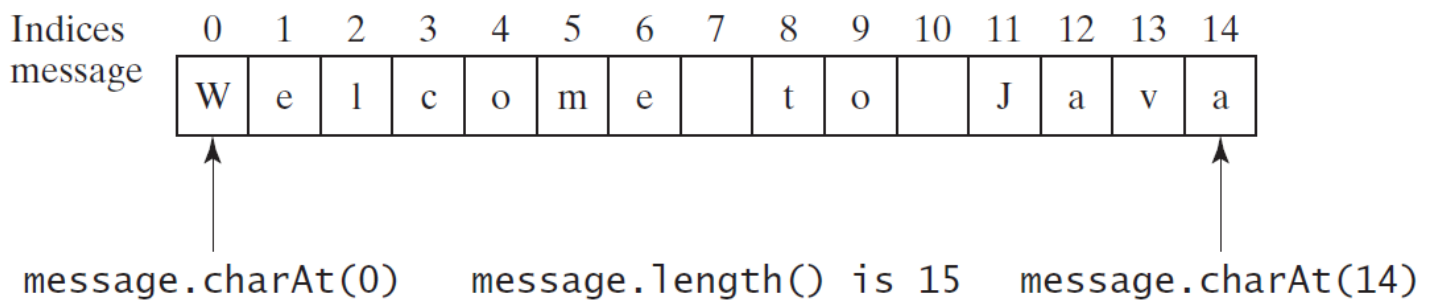
Note

When you use a string, you often know its literal value. For convenience, Java allows you to use the string literal to refer directly to strings without creating new variables. Thus, `"Welcome to Java".length()` is correct and returns `15`. Note that `""` denotes an *empty string* and `"".length()` is `0`.



24

Getting Characters from a String



String message = "Welcome to Java";

System.out.println("The first character in message is "
+ message.charAt(0));



Caution

Attempting to access characters in a string `s` out of bounds is a common programming error. To avoid it, make sure that you do not use an index beyond `s.length() - 1`. For example, `s.charAt(s.length())` would cause a `StringIndexOutOfBoundsException`.



25

Converting Strings

- "Welcome".toLowerCase() returns a new string, `welcome`.
- "Welcome".toUpperCase() returns a new string, `WELCOME`.
- " Welcome Home ".trim() returns a new string, `Welcome Home`.



26

String Concatenation

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

```
// Three strings are concatenated
```

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

```
// String Chapter is concatenated with number 2
```

```
String s = "Chapter" + 2;
```

```
or String s = "Chapter" + "2"; // s becomes Chapter2
```

```
// String Supplement is concatenated with character B
```

```
String s1 = "Supplement" + 'B';
```

```
// s1 becomes SupplementB
```



27

Reading a String from the Console

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

```
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);
```



28

Reading a String from the Console

```
Enter three words separated by spaces: Welcome to Java   
s1 is Welcome  
s2 is to  
s3 is Java
```

The `next()` method reads a string that ends with a whitespace character. You can use 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 statements read a 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);
```

```
Enter a line: Welcome to Java   
The line entered is Welcome to Java
```

29

Reading a Character from the Console

To read a character from the console, use the `nextLine()` method to read a string and then invoke the `charAt(0)` method on the string to return a character. For example, the following code reads a character from the keyboard:

```
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);
```

30

Two Types of String Objects

String objects can be created in following two expressions:

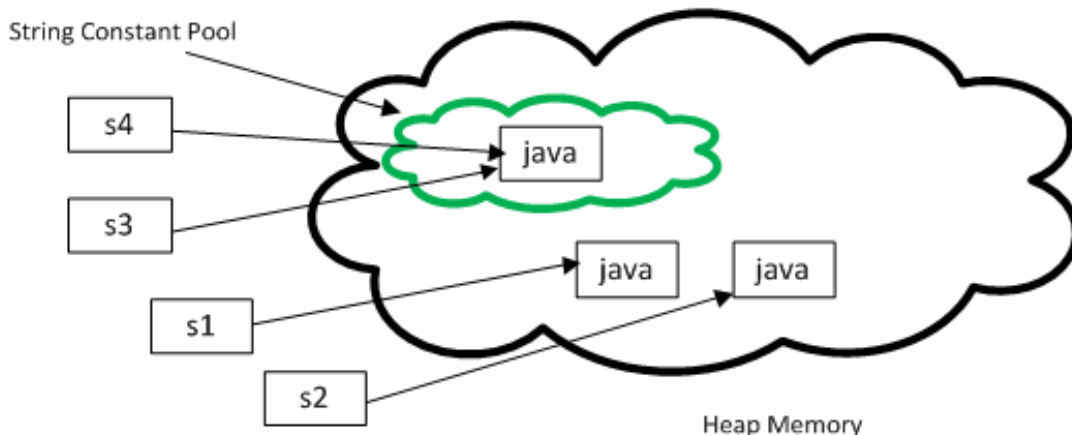
```
String strObject = new String("Java");
```

```
String strLiteral = "Java";
```

When you create String object using new() operator, it always create a new object in the heap memory . On the other hand, if you create object using String literal syntax e.g. "Java", **it may return an existing object from String pool**. Otherwise it will create a new string object and put in string pool for future re-use.

31

REMARK:



```
String a = "Java";  
String b = "Java";  
System.out.println(a == b); // True  
String c = new String("Java");  
String d = new String("Java");  
System.out.println(c == d); // False  
String e = "JDK";  
String f = new String("JDK");  
System.out.println(e == f); // False
```

```
String a = new String("Java");  
String b = new String("Java");  
a = b;  
System.out.println(a==b); //True  
  
String c = "Java";  
String d = "Java";  
c = d;  
System.out.println(c==d); //True
```

32

Comparing Strings

- The `==` operator checks only whether **two string variables** refer to the same object; it does not tell you whether they have the same contents.
- You cannot use the `==` operator to find out whether two string variables have the same contents. Instead, you should use the **`equals`** method.



33

Comparing Strings

The `String` class contains the methods as shown below for comparing two strings.

| Method | Description |
|--------------------------------------|---|
| <code>equals(s1)</code> | Returns true if this string is equal to string <code>s1</code> . |
| <code>equalsIgnoreCase(s1)</code> | Returns true if this string is equal to string <code>s1</code> ; it is case insensitive. |
| <code>compareTo(s1)</code> | 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 <code>s1</code> . |
| <code>compareToIgnoreCase(s1)</code> | Same as <code>compareTo</code> except that the comparison is case insensitive. |
| <code>startsWith(prefix)</code> | Returns true if this string starts with the specified prefix. |
| <code>endsWith(suffix)</code> | Returns true if this string ends with the specified suffix. |



34

Comparing Strings

The following code, for instance, can be used to compare two strings:

```
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, the following statements display **true** and then **false**.

```
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
```



35

Comparing Strings

The actual value returned from the `compareTo` method depends on the offset of the first two distinct characters in `s1` and `s2` from left to right. For example, suppose `s1` is `abc` and `s2` is `abg`, and `s1.compareTo(s2)` returns `-4`. The first two characters (`a` vs. `a`) from `s1` and `s2` are compared. Because they are equal, the second two characters (`b` vs. `b`) are compared. Because they are also equal, the third two characters (`c` vs. `g`) are compared. Since the character `c` is 4 less than `g`, the comparison returns `-4`.



Caution

Syntax errors will occur if you compare strings by using relational operators `>`, `>=`, `<`, or `<=`. Instead, you have to use `s1.compareTo(s2)`.



Note

The `equals` method returns `true` if two strings are equal and `false` if they are not. The `compareTo` method returns `0`, a positive integer, or a negative integer, depending on whether one string is equal to, greater than, or less than the other string.



36

Comparing Strings

```
import java.util.Scanner;

public class OrderTwoCities {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

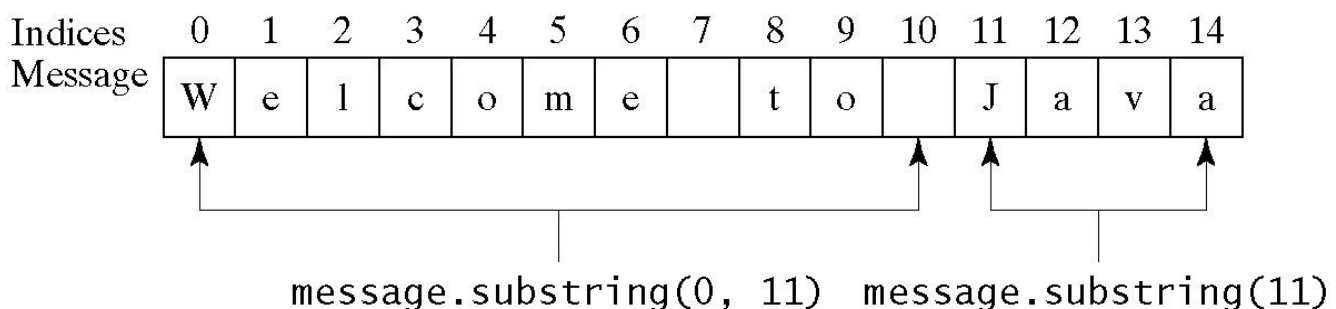
        // Prompt the user to enter two cities
        System.out.print("Enter the first city: ");
        String city1 = input.nextLine();
        System.out.print("Enter the second city: ");
        String city2 = input.nextLine();

        if (city1.compareTo(city2) < 0)
            System.out.println("The cities in alphabetical order are " +
                               city1 + " " + city2);
        else
            System.out.println("The cities in alphabetical order are " +
                               city2 + " " + city1);
    }
}
```

37

Obtaining Substrings

| Method | Description |
|--|---|
| <code>substring(beginIndex)</code> | Returns this string's substring that begins with the character at the specified <code>beginIndex</code> and extends to the end of the string, as shown in Figure 4.2. |
| <code>substring(beginIndex, endIndex)</code> | Returns this string's substring that begins at the specified <code>beginIndex</code> and extends to the character at index <code>endIndex - 1</code> , as shown in Figure 9.6. Note that the character at <code>endIndex</code> is not part of the substring. |



38

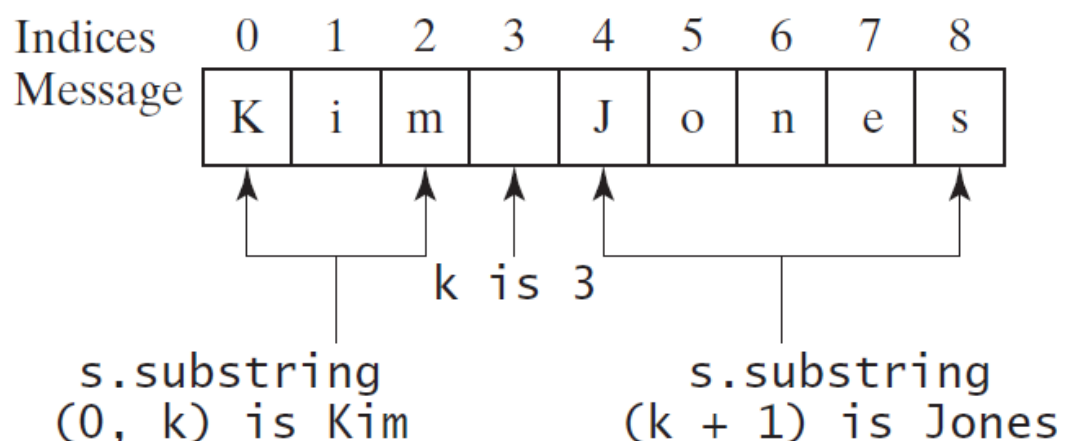
Finding a Character or a Substring in a String

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

39

Finding a Character or a Substring in a String

```
int k = s.indexOf(' ');  
String firstName = s.substring(0, k);  
String lastName = s.substring(k + 1);
```



40

Conversion between Strings and Numbers

You can convert a numeric string into a number. To convert a string into an `int` value, use the `Integer.parseInt` method, as follows:

```
int intValue = Integer.parseInt(intString);
```

where `intString` is a numeric string such as `"123"`.

To convert a string into a `double` value, use the `Double.parseDouble` method, as follows:

```
double doubleValue = Double.parseDouble(doubleString);
```

where `doubleString` is a numeric string such as `"123.45"`.

If the string is not a numeric string, the conversion would cause a runtime error. The `Integer` and `Double` classes are both included in the `java.lang` package, and thus they are automatically imported.

You can convert a number into a string, simply use the string concatenating operator as follows:

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

Formatting Output

Use the `printf` statement.

```
System.out.printf(format, items);
```

where `format` is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

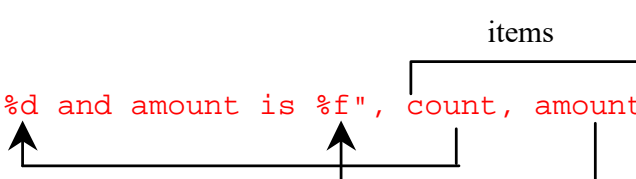


Frequently-Used Specifiers

| Specifier | Output | Example |
|-----------------|--|----------------|
| <code>%b</code> | a boolean value | true or false |
| <code>%c</code> | a character | 'a' |
| <code>%d</code> | a decimal integer | 200 |
| <code>%f</code> | a floating-point number | 45.460000 |
| <code>%e</code> | a number in standard scientific notation | 4.556000e+01 |
| <code>%s</code> | a string | "Java is cool" |

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

display count is 5 and amount is 45.560000



43

Examples

| Example | Output |
|---------------------|--|
| <code>%5c</code> | Output the character and add four spaces before the character item, because the width is 5. |
| <code>%6b</code> | Output the Boolean value and add one space before the false value and two spaces before the true value. |
| <code>%5d</code> | Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number. If the number of digits in the item is > 5, the width is automatically increased. |
| <code>%10.2f</code> | Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus, there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7, add spaces before the number. If the number of digits before the decimal point in the item is > 7, the width is automatically increased. |
| <code>%10.2e</code> | Output the floating-point item with width at least 10 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number. |
| <code>%12s</code> | Output the string with width at least 12 characters. If the string item has fewer than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased. |

44