**Java String contains()**

The contains() method checks whether the specified string (sequence of characters) is present in the string or not.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "Java String contains()";

// check if str1 contains "Java"

boolean result = str1.contains("Java");

System.out.println(result);

}

}

// Output: true

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of contains()**

The syntax of the String contains() method is:

string.contains(CharSequence ch)

Here, string is an object of the String class.

**contains() Parameters**

The contains() method takes a single parameter.

* **ch** (charSequence) - a sequence of characters

**Note:** A charSequence is a sequence of characters such as: String, CharBuffer, StringBuffer etc.

**contains() Return Value**

* **returns true** if the string contains the specified character
* **returns false** if the string doesn't contain the specified character

**Example 1: Java String contains()**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

Boolean result;

// check if str1 contains "Java"

result = str1.contains("Java");

System.out.println(result); // true

// check if str1 contains "Python"

result = str1.contains("Python");

System.out.println(result); // false

// check if str1 contains ""

result = str1.contains("");

System.out.println(result); // true

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, str.contains("") gives true because the empty string is a subset of every other string.

**Example 2: Using contains() With if...else**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "Java";

String str3 = "java";

Boolean result;

// true because "Learn Java" contains "Java"

if (str1.contains(str2)) {

System.out.println(str1 + " contains " + str2);

}

else {

System.out.println(str1 + " doesn't contains " + str2);

}

// contains() is case-sensitive

// false because "Learn Java" doesn't contains "java"

if (str1.contains(str3)) {

System.out.println(str1 + " contains " + str3);

}

else {

System.out.println(str1 + " doesn't contain " + str3);

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Learn Java contains Java

Learn Java doesn't contain java

**Java String substring()**

The Java substring() method extracts a part of the [string](https://www.programiz.com/java-programming/string) (substring) and returns it.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "java is fun";

// extract substring from index 0 to 3

System.out.println(str1.substring(0, 4));

}

}

// Output: java

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of substring()**

The syntax of the substring() method is:

string.substring(int startIndex, int endIndex)

**substring() Argument**

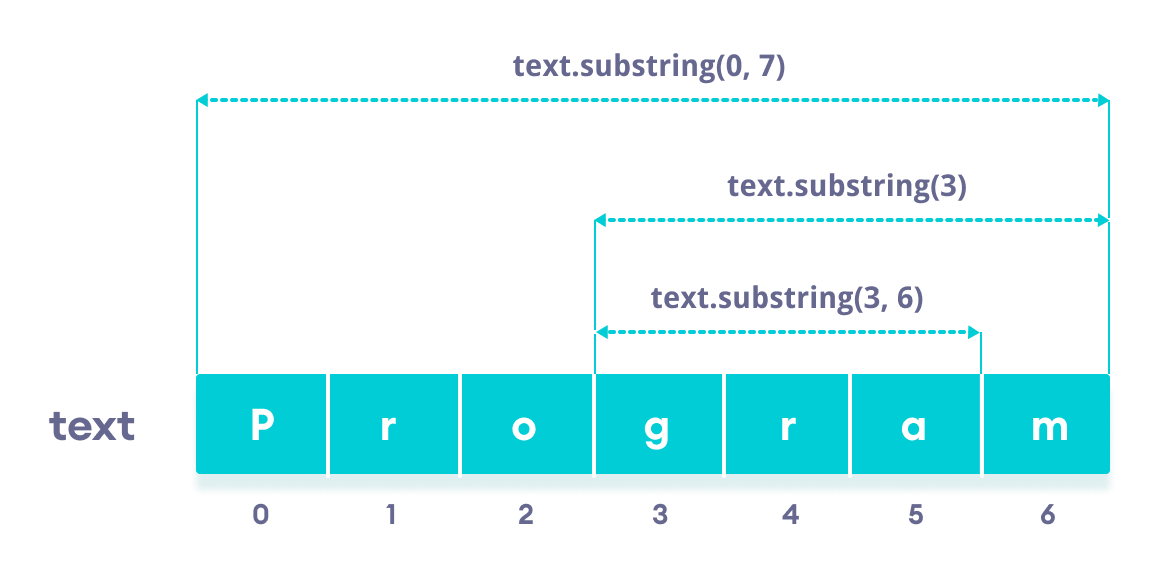
The substring() method can take a maximum of two arguments.

* **startIndex** - the beginning index
* **endIndex** (optional) - the ending index

**substring() Return Value**

The substring() method returns a substring from the given string.

* The substring begins with the character at the startIndex and extends to the character at index endIndex - 1.
* If the endIndex is not passed, the substring begins with the character at the specified index and extends to the end of the string.

Working of Java String substring() method

**Note:** You will get an error if

* startIndex/endIndex is negative or greater than string's length
* startIndex is greater than endIndex

**Example 1: Java substring() With Only Start Index**

class Main {

public static void main(String[] args) {

String str1 = "program";

// 1st character to the last character

System.out.println(str1.substring(0)); // program

// 4th character to the last character

System.out.println(str1.substring(3)); // gram

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Example 2: Java substring() With Start and End Index**

class Main {

public static void main(String[] args) {

String str1 = "program";

// 1st to the 7th character

System.out.println(str1.substring(0, 7)); // program

// 1st to the 5th character

System.out.println(str1.substring(0, 5)); // progr

// 4th to the 5th character

System.out.println(str1.substring(3, 5)); // gr

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

If you need to find the index of the first occurrence of the specified substring from the given string, use the [Java String indexOf()](https://www.programiz.com/java-programming/library/string/indexof) method.

**Java String join()**

The join() method returns a new string with the given elements joined with the specified delimiter.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "I";

String str2 = "love";

String str3 = "Java";

// join strings with space between them

String joinedStr = String.join(" ", str1, str2, str3);

System.out.println(joinedStr);

}

}

// Output: I love Java

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of join()**

The syntax of the string join() method is either:

String.join(CharSequence delimiter,

Iterable elements)

or

String.join(CharSequence delimiter,

CharSequence... elements)

Here, ... signifies there can be one or more CharSequence.

**Note:** join() is a static method. You do not need to create a string object to call this method. Rather, we call the method using the class name String.

**join() Parameters**

The join() method takes two parameters.

* **delimiter** - the delimiter to be joined with the elements
* **elements** - elements to be joined

**Notes:**

* You can pass any class that implements CharSequence to join().
* If an iterable is passed, its elements will be joined. The iterable must implement CharSequence.
* **String**, **StringBuffer**, **CharBuffer** etc. are **CharSequence** as these classes implement it.

**join() Return Value**

* returns a string

**Example 1: Java String join() With CharSequence()**

class Main {

public static void main(String[] args) {

String result;

result = String.join("-", "Java", "is", "fun");

System.out.println(result); // Java-is-fun

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, we have passed three strings Java, is and fun to the join() method. These strings are joined using the - delimiter.

**Example 2: Java String join() With Iterable**

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

ArrayList<String> text = new ArrayList<>();

// adding elements to the arraylist

text.add("Java");

text.add("is");

text.add("fun");

String result;

result = String.join("-", text);

System.out.println(result); // Java-is-fun

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, an ArrayList of String type is created. The elements of array list are joined using the - delimiter.

**Java String replace()**

The replace() method replaces each matching occurrence of a character/text in the string with the new character/text.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "bat ball";

// replace b with c

System.out.println(str1.replace('b', 'c'));

}

}

// Output: cat call

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**replace() Syntax**

The syntax of the replace() method is either

string.replace(char oldChar, char newChar)

or

string.replace(CharSequence oldText, CharSequence newText)

Here, string is an object of the String class.

**replace() Parameters**

To replace a single character, the replace() method takes these two parameters:

* **oldChar** - the character to be replaced in the string
* **newChar** - matching characters are replaced with this character

To replace a substring, the replace() method takes these two parameters:

* **oldText** - the substring to be replaced in the string
* **newText** - matching substrings are replaced with this string

**replace() Return Value**

* The replace() method returns a new string where each occurrence of the matching character/text is replaced with the new character/text.

**Example 1: Java String replace() Characters**

class Main {

public static void main(String[] args) {

String str1 = "abc cba";

// all occurrences of 'a' is replaced with 'z'

System.out.println(str1.replace('a', 'z')); // zbc cbz

// all occurences of 'L' is replaced with 'J'

System.out.println("Lava".replace('L', 'J')); // Java

// character not in the string

System.out.println("Hello".replace('4', 'J')); // Hello

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Note:** If the character to be replaced is not in the string, replace() returns the original string.

**Example 2: Java String replace() Substrings**

class Main {

public static void main(String[] args) {

String str1 = "C++ Programming";

// all occurrences of "C++" is replaced with "Java"

System.out.println(str1.replace("C++", "Java")); // Java Programming

// all occurences of "aa" is replaced with "zz"

System.out.println("aa bb aa zz".replace("aa", "zz")); // zz bb zz zz

// substring not in the string

System.out.println("Java".replace("C++", "C")); // Java

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Note:** If the substring to be replaced is not in the string, replace() returns the original string.

It is important to note that the replace() method replaces substrings starting from the start to the end. For example,

"zzz".replace("zz", "x") // xz

The output of the above code is xz, not zx. It's because the replace() method replaced the first zz with x.

If you need to replace substrings based on a regular expression, use the [Java String replaceAll() method](https://www.programiz.com/java-programming/library/string/replaceall).

**Java String replaceAll()**

The replaceAll() method replaces each substring that matches the regex of the string with the specified text.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "Java123is456fun";

// regex for sequence of digits

String regex = "\\d+";

// replace all occurrences of numeric

// digits by a space

System.out.println(str1.replaceAll(regex, " "));

}

}

// Output: Java is fun

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of replaceAll()**

The syntax of the replaceAll() method is:

string.replaceAll(String regex, String replacement)

Here, string is an object of the String class.

**replaceAll() Parameters**

The replaceAll() method takes two parameters.

* **regex** - a regex (can be a typical string) that is to be replaced
* **replacement** - matching substrings are replaced with this string

**replaceAll() Return Value**

The replaceAll() method

* returns a new string where each occurrence of the matching substring is replaced with the **replacement** string.

**Example 1: Java String replaceAll()**

class Main {

public static void main(String[] args) {

String str1 = "aabbaaac";

String str2 = "Learn223Java55@";

// regex for sequence of digits

String regex = "\\d+";

// all occurrences of "aa" is replaceAll with "zz"

System.out.println(str1.replaceAll("aa", "zz")); // zzbbzzac

// replace a digit or sequence of digits with a whitespace

System.out.println(str2.replaceAll(regex, " ")); // Learn Java @

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

In the above example, "\\d+" is a regular expression that matches one or more digits. To learn more, visit *Java regex*.

**Escaping Characters in replaceAll()**

The replaceAll() method can take a regex or a typical string as the first argument. It is because a typical string in itself is a regex.

In regex, there are characters that have special meaning. These metacharacters are:

\ ^ $ . | ? \* + {} [] ()

If you need to match substring containing these metacharacters, you can either escape these characters using \ or use the replace() method.

// Program to replace the + character

class Main {

public static void main(String[] args) {

String str1 = "+a-+b";

// replace "+" with "#" using replaceAll()

// need to escape "+"

System.out.println(str1.replaceAll("\\+", "#")); // #a-#b

// replace "+" with "#" using replace()

System.out.println(str1.replace("+", "#")); // #a-#b

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

As you can see, when we use the replace() method, we do not need to escape metacharacters. To learn more, visit: [Java String replace()](https://www.programiz.com/java-programming/library/string/replace)

If you need to replace only the first occurrence of the matching substring, use the [Java String replaceFirst()](https://www.programiz.com/java-programming/library/string/replacefirst) method.

**Java String replaceFirst()**

The syntax of the replaceFirst() method is:

string.replaceFirst(String regex, String replacement)

Here, string is an object of the String class.

**replaceFirst() Parameters**

The replaceFirst() method takes two parameters.

* **regex** - a regex (can be a typical string) that is to be replaced
* **replacement**- the first matching substring is replaced with this string

**replaceFirst() Return Value**

* The replaceFirst() method returns a new string where the first occurrence of the matching substring is replaced with the **replacement** string.

**Example 1: Java String replaceFirst()**

class Main {

public static void main(String[] args) {

String str1 = "aabbaaac";

String str2 = "Learn223Java55@";

// regex for sequence of digits

String regex = "\\d+";

// the first occurrence of "aa" is replaced with "zz"

System.out.println(str1.replaceFirst("aa", "zz")); // zzbbaaac

// replace the first sequence of digits with a whitespace

System.out.println(str2.replaceFirst(regex, " ")); // Learn Java55@

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

In the above example, "\\d+" is a regular expression that matches a sequence of digits. To learn more, visit *Java regex*.

**Escaping Characters in replaceFirst()**

The replaceFirst() method can take a regex or a typical string as the first argument. It is because a typical string in itself is a regex.

In regex, there are characters that have special meaning. These metacharacters are:

\ ^ $ . | ? \* + {} [] ()

If you need to match substring containing these metacharacters, you can escape these characters using \.

// Program to the first + character

class Main {

public static void main(String[] args) {

String str = "a+a-++b";

// replace the first "+" with "#"

System.out.println(str.replaceFirst("\\+", "#")); // a#a-++b

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

If you need to replace each substring that matches the regex, use the [Java String replaceAll() method](https://www.programiz.com/java-programming/library/string/replaceall).

**Java String charAt()**

The charAt() method returns the character at the specified index.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "Java Programming";

// returns character at index 2

System.out.println(str1.charAt(2));

}

}

// Output: v

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of charAt()**

The syntax of the string charAt() method is:

string.charAt(int index)

Here, string is an object of the String class.

**charAt() Parameters**

* **index** - the index of the character (an int value)

**charAt() Return Value**

* returns the character at the specified index

**Note:** If the index passed to chartAt() is negative or out of bounds, it throws an exception.

**Example: Java String charAt()**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "Learn\nJava";

// first character

System.out.println(str1.charAt(0)); // 'L'

// seventh character

System.out.println(str1.charAt(6)); // 'J'

// sixth character

System.out.println(str2.charAt(5)); // '\n'

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

In Java, the index of Strings starts from **0**, not **1**. That's why chartAt(0) returns the first character. Similarly, charAt(5) and charAt(6) return the sixth and seventh character respectively.

If you need to find the index of the first occurrence of the specified character, use the [Java String indexOf()](https://www.programiz.com/java-programming/library/string/indexof) method.

**Java String getBytes()**

The syntax of the String getBytes() method are:

string.getBytes()

string.getBytes(Charset charset)

string.getBytes(String charsetName)

Here, string is an object of the String class.

The getBytes() method returns a byte array.

**1. getBytes() Without Any Parameters**

If you do not pass any parameters, getBytes() encodes the string using the platform's default charset.

**Example: getBytes() Without Any Parameters**

import java.util.Arrays;

class Main {

public static void main(String[] args) {

String str = "Java";

byte[] byteArray;

// convert the string to a byte array

// using platform's default charset

byteArray = str.getBytes();

System.out.println(Arrays.toString(byteArray));

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

[74, 97, 118, 97]

**Note:** We have used the Arrays class in the above example to print the byte array in a readable form. It has nothing to do with getBytes().

**2. getBytes() With CharSet Parameter**

Here are different CharSet available in java:

* **UTF-8** - Eight-bit UCS Transformation Format
* **UTF-16** - Sixteen-bit UCS Transformation Format
* **UTF-16BE** - Sixteen-bit UCS Transformation Format, big-endian byte order
* **UTF-16LE** - Sixteen-bit UCS Transformation Format, little-endian byte order
* **US-ASCII** - Seven-bit ASCII
* **ISO-8859-1** - ISO Latin Alphabet No. 1

**Example: getBytes() With CharSet Parameter**

import java.util.Arrays;

import java.nio.charset.Charset;

class Main {

public static void main(String[] args) {

String str = "Java";

byte[] byteArray;

// using UTF-8 for encoding

byteArray = str.getBytes(Charset.forName("UTF-8"));

System.out.println(Arrays.toString(byteArray));

// using UTF-16 for encoding

byteArray = str.getBytes(Charset.forName("UTF-16"));

System.out.println(Arrays.toString(byteArray));

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

[74, 97, 118, 97]

[-2, -1, 0, 74, 0, 97, 0, 118, 0, 97]

**Note:**In the above program, we have imported java.nio.charset.Charset to use CharSet. And, we have imported the Arrays class to print the byte array in a readable form.

**3. getBytes() With String Parameter**

You can also specify the encoding type to getBytes() using strings. When you use getBytes() in this way, you must wrap the code inside [try...catch block](https://www.programiz.com/java-programming/exception-handling).

**Example: getBytes() With String Parameter**

import java.util.Arrays;

class Main {

public static void main(String[] args) {

String str = "Java";

byte[] byteArray;

try {

byteArray = str.getBytes("UTF-8");

System.out.println(Arrays.toString(byteArray));

byteArray = str.getBytes("UTF-16");

System.out.println(Arrays.toString(byteArray));

// wrong encoding

// throws an exception

byteArray = str.getBytes("UTF-34");

System.out.println(Arrays.toString(byteArray));

} catch (Exception e) {

System.out.println(e + " encoding is wrong");

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

[74, 97, 118, 97]

[-2, -1, 0, 74, 0, 97, 0, 118, 0, 97]

java.io.UnsupportedEncodingException: UTF-34 encoding is wrong

**Note:** We have imported java.util.Arrays to print the byte array in a readable form. It has nothing to do with getBytes().

**Java String indexOf()**

The indexOf() method returns the index of the first occurrence of the specified character/substring within the string.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "Java is fun";

int result;

// getting index of character 's'

result = str1.indexOf('s');

System.out.println(result);

}

}

// Output: 6

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of indexOf()**

The syntax of the String indexOf() method either

string.indexOf(int ch, int fromIndex)

or

string.indexOf(String str, int fromIndex)

Here, string is an object of the String class.

**indexOf() Parameters**

To find the index of a character, indexOf() takes these two parameters:

* **ch** - the character whose starting index is to be found
* **fromIndex** (optional) - if fromIndex is passed, the ch character is searched starting from this index

To find the index of the specified substring within the string, indexOf() takes these two parameters:

* **str** - the string whose starting index is to be found
* **fromIndex** (optional) - if fromIndex is passed, the str string is searched starting from this index

**indexOf() Return Value**

* **returns the index** of the first occurrence of the specified character/string
* **returns -1** if the specified character/string is not found.

**Example 1: Java String indexOf()**

// Java String indexOf() with only one parameter

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

int result;

// getting index of character 'J'

result = str1.indexOf('J');

System.out.println(result); // 6

// the first occurrence of 'a' is returned

result = str1.indexOf('a');

System.out.println(result); // 2

// character not in the string

result = str1.indexOf('j');

System.out.println(result); // -1

// getting the index of "ava"

result = str1.indexOf("ava");

System.out.println(result); // 7

// substring not in the string

result = str1.indexOf("java");

System.out.println(result); // -1

// index of empty string in the string

result = str1.indexOf("");

System.out.println(result); // 0

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Notes:**

* The character 'a' occurs multiple times in the "Learn Java" string. The indexOf() method returns the index of the first occurrence of 'a' (which is 2).
* If the empty string is passed, indexOf() returns 0 (found at the first position. It is because the empty string is a subset of every substring.

**Example 2: indexOf() With fromIndex Parameter**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java programming";

int result;

// getting the index of character 'a'

// search starts at index 4

result = str1.indexOf('a', 4);

System.out.println(result); // 7

// getting the index of "Java"

// search starts at index 8

result = str1.indexOf("Java", 8);

System.out.println(result); // -1

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Notes:**

* The first occurrence of 'a' in the "Learn Java programming" string is at index 2. However, the index of second 'a' is returned when str1.indexOf('a', 4) is used. It is because the search starts at index 4.
* The "Java" string is in the "Learn Java programming" string. However, str1.indexOf("Java", 8) returns -1 (string not found). It is because the search starts at index 8 and there is no "Java" in "va programming".

**Java String compareTo()**

The compareTo() method compares two strings lexicographically (in the dictionary order). The comparison is based on the Unicode value of each character in the strings.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "Learn Kolin";

int result;

// comparing str1 with str2

result = str1.compareTo(str2);

System.out.println(result);

}

}

// Output: -1

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of compareTo()**

The syntax of the compareTo() method is:

string.compareTo(String str)

Here, string is an object of the String class.

**compareTo() Parameters**

The compareTo() method takes a single parameter.

* **str** - the string to be compared

**compareTo() Return Value**

* **returns 0** if the strings are equal
* **returns a negative integer** if the string comes before the str argument in the dictionary order
* **returns a positive integer** if the string comes after the str argument in the dictionary order

**Example: Java String compareTo()**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "Learn Java";

String str3 = "Learn Kolin";

int result;

// comparing str1 with str2

result = str1.compareTo(str2);

System.out.println(result); // 0

// comparing str1 with str3

result = str1.compareTo(str3);

System.out.println(result); // -1

// comparing str3 with str1

result = str3.compareTo(str1);

System.out.println(result); // 1

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here,

* str1 and str2 are equal. Hence, str1.compareTo(str2) returns 0.
* str1 comes before str3 in the dictionary order. Hence, str1.compareTo(str3) returns negative, and str3.compareTo(str1) returns positive.

**Example 2: Check if Two Strings are Equal**

class Main {

public static void main(String[] args) {

String str1 = "Learn Python";

String str2 = "Learn Java";

// if str1 and str2 are equal, the result is 0

if (str1.compareTo(str2) == 0) {

System.out.println("str1 and str2 are equal");

}

else {

System.out.println("str1 and str2 are not equal");

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

str1 and str2 are not equal

**Example 3: compareTo() With Case**

The compareTo() method takes the letter case (uppercase and lowercase) into consideration.

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "learn Java";

int result;

// comparing str1 with str2

result = str1.compareTo(str2);

System.out.println(result); // -32

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

When "Learn Java" is compared to "learn Java", we do not get **0**. It is because compareTo() takes the letter case into consideration.

**Notes:**

* If you need to compare two strings ignoring case differences, use the [Java String compareToIgnoreCase()](https://www.programiz.com/java-programming/library/string/comparetoignorecase) method.
* If you pass null to the compareTo() method, you will get an error.

**Java String compareToIgnoreCase()**

The syntax of the string compareToIgnoreCase() method is:

string.compareToIgnoreCase(String str)

Here, string is an object of the String class.

**compareToIgnoreCase() Parameters**

The string compareToIgnoreCase() method takes a single parameter.

* **str** - the string to be compared

**compareToIgnoreCase() Return Value**

* **returns 0** if the strings are equal, ignoring case considerations
* **returns a negative integer** if string comes before the str argument in the dictionary order
* **returns a positive integer** if string comes after the str argument in the dictionary order

**Example: Java String compareToIgnoreCase()**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "learn java";

String str3 = "Learn Kolin";

int result;

// comparing str1 with str2

result = str1.compareToIgnoreCase(str2);

System.out.println(result); // 0

// comparing str1 with str3

result = str1.compareToIgnoreCase(str3);

System.out.println(result); // -1

// comparing str3 with str1

result = str3.compareToIgnoreCase(str1);

System.out.println(result); // 1

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here,

* str1 and str2 are equal if you do not consider the case differences. Hence, str1.compareToIgnoreCase(str2) returns 0.
* str1 comes before str3 in the dictionary order. Hence, str1.compareToIgnoreCase(str3) returns negative, and str3.compareToIgnoreCase(str1) returns positive.

**Example 2: Check if Two Strings are Equal**

class Main {

public static void main(String[] args) {

String str1 = "LEARN JAVA";

String str2 = "Learn Java";

// if str1 and str2 are equal (ignoring case differences),

// the result is 0

if (str1.compareToIgnoreCase(str2) == 0) {

System.out.println("str1 and str2 are equal");

}

else {

System.out.println("str1 and str2 are not equal");

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

str1 and str2 are equal

If you need to compare two strings with case differences taken into consideration, use either

* [Java String CompareTo()](https://www.programiz.com/java-programming/library/string/compareto)
* [Java String equals()](https://www.programiz.com/java-programming/library/string/equals)

**Java String trim()**

The trim() method removes any leading (starting) and trailing (ending) whitespaces from the specified string.

**Example**

class Main {

public static void main(String[] args) {

String str1 = " Learn Java Programming ";

System.out.println(str1.trim());

}

}

// Output: Learn Java Programming

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of trim()**

The syntax of the string trim() method is:

string.trim()

Here, string is an object of the String class.

**trim() Parameters**

* the trim() method doesn't take any parameters

**trim() Return Value**

* returns a string with leading and trailing whitespace removed
* returns the original string if there is no whitespace in the start or the end of the string

**Note:** In programming, whitespace is any character or series of characters that represent horizontal or vertical space. For example: space, newline \n, tab \t, vertical tab \v etc.

**Example: Java String trim()**

class Main {

public static void main(String[] args) {

String str1 = " Learn Java Programming ";

String str2 = "Learn\nJava Programming\n\n ";

System.out.println(str1.trim());

System.out.println(str2.trim());

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Learn Java Programming

Learn

Java Programming

Here, str1.trim() returns

"Learn Java Programming"

Similarly, str2.trim() returns

"Learn\nJava Programming"

As you can see from the above example, the trim() method only removes the leading and trailing whitespace. It doesn't remove whitespace that appears in the middle.

**Remove All Whitespace Characters**

If you need to **remove all whitespace characters from a string**, you can use the [String replaceAll() method](https://www.programiz.com/java-programming/library/string/replaceall) with proper regex.

class Main {

public static void main(String[] args) {

String str1 = "Learn\nJava \n\n ";

String result;

// replace all whitespace characters with empty string

result = str1.replaceAll("\\s", "");

System.out.println(result); // LearnJava

}

}

[Run](https://www.programiz.com/java-programming/online-compiler)

**Java String format()**

The format() method returns a formatted string based on the argument passed.

**Example**

class Main {

public static void main(String[] args) {

String str = "Java";

// format string

String formatStr = String.format("Language: %s", str);

System.out.println(formatStr);

}

}

// Output: Language: Java

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**format() Syntax**

The syntax of the String format() method is:

String.format(String str, Object... args)

Here,

* format() is a static method. We call the format() method using the class name String.
* str is a string that is to be formatted
* ... in the above code signifies you can pass more than one object to format().

**format() Parameters**

The format() method takes two parameters.

* **format** - a format string
* **args** - 0 or more arguments

**format() Return Value**

* returns a formatted string

**Example 1: Java String format()**

class Main {

public static void main(String[] args) {

String language = "Java";

int number = 30;

String result;

// format object as a string

result = String.format("Language: %s", language);

System.out.println(result); // Language: Java

// format number as a hexadecimal number

result = String.format("Hexadecimal Number: %x", number); // 1e

System.out.println(result); // Hexadecimal Number: 1e

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

In the above program, notice the code

result = String.format("Language: %s", language);

Here, "Language: %s" is a **format string**.

%s in the format string is replaced with the content of language. %s is a format specifier.

Similarly, %x is replaced with the hexadecimal value of number in String.format("Number: %x", number).

**Format Specifiers**

Here are the commonly used format specifiers:

|  |  |
| --- | --- |
| Specifier | Description |
| %b, %B | "true" or "false" based on the argument |
| %s, %S | a string |
| %c, %C | a Unicode character |
| %d | a decimal integer (used for integers only) |
| %o | an octal integer (used for integers only) |
| %x, %X | a hexadecimal integer (used for integers only) |
| %e, %E | for scientific notation (used for floating-point numbers) |
| %f | for decimal numbers (used for floating-point numbers) |

**Example 2: String Formatting of Numbers**

class Main {

public static void main(String[] args) {

int n1 = 47;

float n2 = 35.864f;

double n3 = 44534345.76d;

// format as an octal number

System.out.println(String.format("n1 in octal: %o", n1)); // 57

// format as hexadecimal numbers

System.out.println(String.format("n1 in hexadecimal: %x", n1)); // 2f

System.out.println(String.format("n1 in hexadecimal: %X", n1)); // 2F

// format as strings

System.out.println(String.format("n1 as string: %s", n1)); // 47

System.out.println(String.format("n2 as string: %s", n2)); // 35.864

// format in scientific notation

System.out.println(String.format("n3 in scientific notation: %g", n3)); // 4.45343e+07

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

n1 in octal: 57

n1 in hexadecimal: 2f

n1 in hexadecimal: 2F

n1 as string: 47

n2 as string: 35.864

n3 in scientific notation: 4.45343e+07

**Example 3: String format with multiple format specifiers**

You can use more than one format specifier in the format string.

// using more than one format specifiers

// in a format string

class Main {

public static void main(String[] args) {

int n1 = 47;

String text = "Result";

System.out.println(String.format("%s\nhexadecimal: %x", text, n1));

}

}

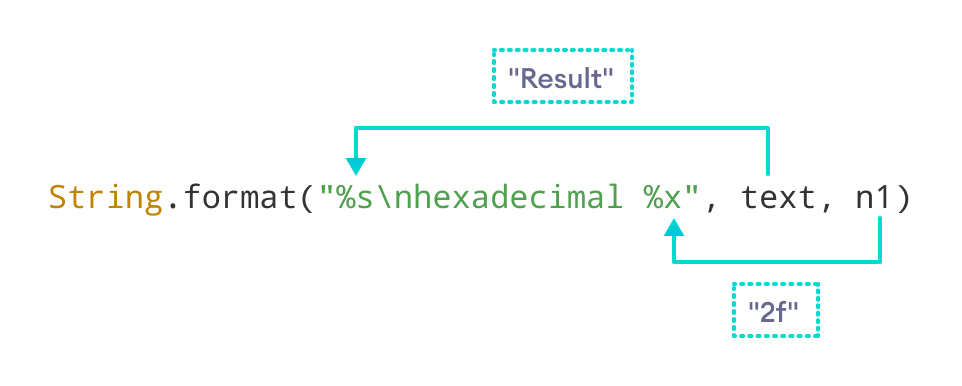
[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Result

hexadecimal: 2f

Here, %s is replaced with the value of text. Similarly, %o is replaced with the hexadecimal value of n1.

Working of Java String format()

**Example 4: Formatting of Decimal Numbers**

class Main {

public static void main(String[] args) {

float n1 = -452.534f;

double n2 = -345.766d;

// format floating-point as it is

System.out.println(String.format("n1 = %f", n1)); // -452.533997

System.out.println(String.format("n2 = %f", n2)); // -345.766000

// show up to two decimal places

System.out.println(String.format("n1 = %.2f", n1)); // -452.53

System.out.println(String.format("n2 = %.2f", n2)); // -345.77

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

n1 = -452.533997

n2 = -345.766000

n1 = -452.53

n2 = -345.77

**Note:** When we format **-452.534** using %f, we are getting **-452.533997**. It is not because of the format() method. Java doesn't return the exact [representation of floating-point numbers](https://stackoverflow.com/questions/588004/is-floating-point-math-broken).

When %.2f format specifier is used, format() gives two numbers after the decimal point.

**Example 5: Padding Numbers With Spaces and 0**

// using more than one format specifiers

// in a format string

class Main {

public static void main(String[] args) {

int n1 = 46, n2 = -46;

String result;

// padding number with spaces

// the length of the string will be 5

result = String.format("|%5d|", n1); // | 46|

System.out.println(result);

// padding number with numbers 0

// the length of the string will be 5

result = String.format("|%05d|", n1); // |00046|

System.out.println(result);

// using signs before numbers

result = String.format("%+d", n1); // +46

System.out.println(result);

result = String.format("%+d", n2); // -46

System.out.println(result);

// enclose negative number within parenthesis

// and removing the sign

result = String.format("%(d", n2); // (46)

System.out.println(result);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Example 6: Using 0x and 0 before Hexadecimal and Octal**

// using 0x before hexadecimal

// using 0 before octal

class Main {

public static void main(String[] args) {

int n = 46;

System.out.println(String.format("%#o", n)); // 056

System.out.println(String.format("%#x", n)); // 0x2e

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Java String format() with Locale**

The String format() method also has another syntax if you have to work with the specified [locale](https://docs.oracle.com/javase/8/docs/api/java/util/Locale.html).

String.format(Locale l,

String format,

Object... args)

**Example 7: Using GERMAN Locale in format()**

// to use Locale

import java.util.Locale;

class Main {

public static void main(String[] args) {

int number = 8652145;

String result;

// using the current locale

result = String.format("Number: %,d", number);

System.out.println(result);

// using the GERMAN locale as the first argument

result = String.format(Locale.GERMAN, "Number in German: %,d", number);

System.out.println(result);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Number: 8,652,145

Number in German: 8.652.145

**Note:** In Germany, integers are separated by . instead of ,.

**Java String split()**

The split() method divides the string at the specified regex and returns an array of substrings.

**Example**

class Main {

public static void main(String[] args) {

String text = "Java is a fun programming language";

// split string from space

String[] result = text.split(" ");

System.out.print("result = ");

for (String str : result) {

System.out.print(str + ", ");

}

}

}

// Output: result = Java, is, a, fun, programming, language,

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of String split()**

The syntax of the string split() method is:

string.split(String regex, int limit)

Here, string is an object of the String class.

**split() Parameters**

The string split() method can take two parameters:

* **regex** - the string is divided at this regex (can be strings)
* **limit** (optional) - controls the number of resulting substrings

If the limit parameter is not passed, split() returns all possible substrings.

**split() Return Value**

* returns an array of substrings

**Note:** If the regular expression passed to split() is invalid, the split() method raises PatternSyntaxExpression exception.

**Example 1: split() Without limit Parameter**

// importing Arrays to convert array to string

// used for printing arrays

import java.util.Arrays;

class Main {

public static void main(String[] args) {

String vowels = "a::b::c::d:e";

// splitting the string at "::"

// storing the result in an array of strings

String[] result = vowels.split("::");

// converting array to string and printing it

System.out.println("result = " + Arrays.toString(result));

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

result = [a, b, c, d:e]

Here, we split the string at ::. Since the limit parameter is not passed, the returned array contains all the substrings.

**split() With limit Parameter**

* If the limit parameter is 0 or negative, split() returns an array containing all substrings.
* If the limit parameter is positive (let's say n), split() returns the maximum of n substrings.

**Example 2: split() With limit Parameter**

// importing Arrays to convert array to string

import java.util.Arrays;

class Main {

public static void main(String[] args) {

String vowels = "a:bc:de:fg:h";

// splitting array at ":"

// limit is -2; array contains all substrings

String[] result = vowels.split(":", -2);

System.out.println("result when limit is -2 = " + Arrays.toString(result));

// limit is 0; array contains all substrings

result = vowels.split(":", 0);

System.out.println("result when limit is 0 = " + Arrays.toString(result));

// limit is 2; array contains a maximum of 2 substrings

result = vowels.split(":", 2);

System.out.println("result when limit is 2 = " + Arrays.toString(result));

// limit is 4; array contains a maximum of 4 substrings

result = vowels.split(":", 4);

System.out.println("result when limit is 4 = " + Arrays.toString(result));

// limit is 10; array contains a maximum of 10 substrings

result = vowels.split(":", 10);

System.out.println("result when limit is 10 = " + Arrays.toString(result));

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

result when limit is -2 = [a, bc, de, fg, h]

result when limit is 0 = [a, bc, de, fg, h]

result when limit is 2 = [a, bc:de:fg:h]

result when limit is 4 = [a, bc, de, fg:h]

result when limit is 10 = [a, bc, de, fg, h]

**Note:** The **split()** method takes regex as the first argument. If you need to use special characters such as: \, |, ^, \*, + etc, you need to escape these characters. For example, we need to use \\+ to split at +.

**Example 3: split() at the + character**

// importing Arrays to convert array to string

// used for printing arrays

import java.util.Arrays;

class Main {

public static void main(String[] args) {

String vowels = "a+e+f";

// splitting the string at "+"

String[] result = vowels.split("\\+");

// converting array to string and printing it

System.out.println("result = " + Arrays.toString(result));

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

result = [a, e, f]

Here, to split a string at +, we have used \\+. It's because + is a special character (has a special meaning in regular expressions).

**Java String toLowerCase()**

The toLowerCase() method converts all characters in the string to lowercase characters.

**Example**

class Main {

public static void main(String[] args) {

String str1 = "JAVA PROGRAMMING";

// convert to lower case letters

System.out.println(str1.toLowerCase());

}

}

// Output: java programming

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of toLowerCase()**

The syntax of the string toLowerCase() method is:

string.toLowerCase()

Here, string is an object of the String class.

**toLowerCase() Parameters**

The toLowerCase() method does not take any parameters.

**toLowerCase() Return Value**

* returns a string with all upper case letters converted to lowercase letters

**Example: Java toLowerCase()**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "Java123";

// convert to lowercase letters

System.out.println(str1.toLowerCase()); // "learn java"

System.out.println(str2.toLowerCase()); // "java123"

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

As you can see from the above example, toLowerCase() converts all uppercase letters to lower case letters.

**toLowerCase() With Locale Parameter**

The toLowerCase() method can also take a locale as an argument. This allows you to convert characters in a string to lowercase using the given [Locale](https://docs.oracle.com/javase/8/docs/api/java/util/Locale.html) (such as: Turkish, Lithuanian etc.) rules.

Its syntax is:

string.toLowerCase(Locale locale)

If you do not pass the locale parameter, the default locale, Locale.getDefault(), is used.

To learn more, visit [Java toLowerCase() With Locale](https://javapapers.com/core-java/javas-tolowercase-has-got-a-surprise-for-you/).

To convert all characters in a string to upper case characters, use the [Java String toUpperCase() method](https://www.programiz.com/java-programming/library/string/touppercase).

**Java String toUpperCase()**

The syntax of the string toUpperCase() method is:

string.toUpperCase()

**toUpperCase() Parameters**

* doesn't take any parameters

**toUpperCase() Return Value**

* returns a string with all lower case letters converted to upper case

**Example: Java toUpperCase()**

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "Java123";

// convert to upper case letters

System.out.println(str1.toUpperCase()); // "LEARN JAVA"

System.out.println(str2.toUpperCase()); // "JAVA123"

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

As you can see from the above example, toUpperCase() converts all lower case letters to upper case letters.

**toUpperCase() With Locale Parameter**

The toUpperCase() method can also take a locale as an argument. This allows you to convert all characters in a string to upper case using the given [Locale](https://docs.oracle.com/javase/8/docs/api/java/util/Locale.html) (such as: Turkish, Lithuanian etc.) rules.

Its syntax is:

string.toUpperCase(Locale locale)

If you do not pass the locale parameter, the default locale, Locale.getDefault(), is used.

To learn more, visit [Java toUpperCase() With Locale](https://stackoverflow.com/questions/11063102/using-locales-with-javas-tolowercase-and-touppercase).

To convert all characters in a string to lower case letters, use the [Java String toLowerCase() method](https://www.programiz.com/java-programming/library/string/tolowercase).

**Java String valueOf()**

The valueOf() method returns the string representation of the argument passed.

**Example**

class Main {

public static void main(String[] args) {

double interest = 923.234d;

// convert double to string

System.out.println(String.valueOf(interest));

}

}

// Output: 923.234

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of valueOf()**

The syntax of the String valueOf() method for different data types is:

String.valueOf(boolean b)

String.valueOf(char c)

String.valueOf(char[] data)

String.valueOf(double d)

String.valueOf(float f)

String.valueOf(int b)

String.valueOf(long l)

String.valueOf(Object o)

Here, valueOf() is a static method. We call the valueof() method using the class name like this: String.valueOf(b);

**valueOf() Parameters**

The valueOf() method takes a single parameter.

* data that is to be converted to a string

**valueOf() Return Value**

* returns the string representation of the argument passed

**Example: Java String valueOf() for Numbers**

class Main {

public static void main(String[] args) {

int a = 5;

long l = -2343834L;

float f = 23.4f;

double d = 923.234d;

// convert numbers to strings

System.out.println(String.valueOf(a)); // "5"

System.out.println(String.valueOf(l)); // "-2343834"

System.out.println(String.valueOf(f)); // "23.4"

System.out.println(String.valueOf(d)); // "923.234"

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Example 2: Convert char and char array to String**

In Java, you can also use the + operator to concatenate two strings. For example,

class Main {

public static void main(String[] args) {

char c = 'J';

char ch[] = {'J', 'a', 'v', 'a'};

// convert char to string

System.out.println(String.valueOf(c)); // "J"

// convert char array to string

System.out.println(String.valueOf(ch)); // "Java"

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Convert subarray of the char Array to String**

You can also convert a subarray of a character array to string. For this, we use this syntax.

valueOf(char[] data, int offset, int length)

Here,

* **data** - the character array
* **offset** - initial offset of the subarray
* **count** - the length of the subarray

**Example 3: Subarray of a char Array to String**

class Main {

public static void main(String[] args) {

char ch[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm'};

int offset = 2;

int length = 4;

String result;

// subarray {'o', 'g', 'r', 'm'} is converted to string

result = String.valueOf(ch, offset, length);

System.out.println(result); // "ogrm"

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Example 4: Convert Object to String**

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

ArrayList<String> languages = new ArrayList<String>();

languages.add("Java");

languages.add("Python");

languages.add("Kotlin");

String result;

// Output: "[Java, Python, Kotlin]"

result = String.valueOf(languages);

System.out.println(result);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, an ArrayList object, languages, is converted to a string.

In Java, there is another method named copyValueOf() which is equivalent to the valueOf() method.

**Note:** You can also use the object.toString() method to convert an object to a string. To learn more, visit: [Java Object toString()](https://www.programiz.com/java-programming/library/object/tostring) method.

**Java String toCharArray()**

The syntax of the toCharArray() method is:

string.toCharArray()

Here, string is an object of the String class.

**toCharArray() Parameters**

The toCharArray() method doesn't take any parameters.

**toCharArray() Return Value**

* returns a char array

**Example: Java String toCharArray()**

class Main {

public static void main(String[] args) {

String str = "Java Programming";

// creating a char array

char[] result;

result = str.toCharArray();

System.out.println(result); // Java Programming

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, the length of the char array result will be equal to the length of the string str.

**Java String matches()**

The matches() method checks whether the string matches the given regular expression or not.

**Example**

class Main {

public static void main(String[] args) {

// a regex pattern for

// four letter string that starts with 'J' and end with 'a'

String regex = "^J..a$";

System.out.println("Java".matches(regex));

}

}

// Output: true

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of matches()**

The syntax of the string matches() method is:

string.matches(String regex)

Here, string is an object of the String class.

**matches() Parameters**

The matches() method takes a single parameter.

* **regex** - a regular expression

**matches() Return Value**

* **returns true** if the regex matches the string
* **returns false** if the regex doesn't match the string

**Example 1: Java matches()**

class Main {

public static void main(String[] args) {

// a regex pattern for

// five letter string that starts with 'a' and end with 's'

String regex = "^a...s$";

System.out.println("abs".matches(regex)); // false

System.out.println("alias".matches(regex)); // true

System.out.println("an abacus".matches(regex)); // false

System.out.println("abyss".matches(regex)); // true

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, "^a...s$" is a regex, which means a 5 letter string that starts with a and ends with s.

**Example 2: Check for Numbers**

// check whether a string contains only numbers

class Main {

public static void main(String[] args) {

// a search pattern for only numbers

String regex = "^[0-9]+$";

System.out.println("123a".matches(regex)); // false

System.out.println("98416".matches(regex)); // true

System.out.println("98 41".matches(regex)); // false

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, "^[0-9]+$" is a regex, which means only digits.

To learn more about regex, visit *Java Regex*.

**Java String startsWith()**

The startsWith() method checks whether the string begins with the specified string or not.

**Example**

class Main {

public static void main(String[] args) {

String str = "JavaScript";

// checks if "JavaScript" starts with "Java"

System.out.println(str.startsWith("Java"));

}

}

// Output: true

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Syntax of startsWith()**

The syntax of the string startsWith() method is:

string.startsWith(String str, int offset)

Here, string is an object of the String class.

**startsWith() Parameters**

The startsWith() method can take two parameters.

* **str** - check whether string starts with str or not
* **offset** (optional) - checks in a substring of string starting from this index.

**startsWith() Return Value**

* **returns true** if the string begins with the given string
* **returns false** if the string doesn't begin with the given string

**Example 1: Java startsWith() Without Offset Parameter**

class Main {

public static void main(String[] args) {

String str = "Java Programming";

System.out.println(str.startsWith("Java")); // true

System.out.println(str.startsWith("J")); // true

System.out.println(str.startsWith("Java Program")); // true

System.out.println(str.startsWith("java")); // false

System.out.println(str.startsWith("ava")); // false

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

As you can see from the above example, startsWith() takes case (lowercase and uppercase) into consideration.

**Example 2: Java startsWith() With Offset Parameter**

class Main {

public static void main(String[] args) {

String str = "Java Programming";

// checks in substring "a Programming"

System.out.println(str.startsWith("Java", 3)); // false

System.out.println(str.startsWith("a Pr", 3)); // true

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, we have passed 3 as an offset. Hence, in the above program, startsWith() checks whether "a Programming" begins with the specified string.

If you need to check whether the string ends with the specified string or not, use the [Java String endsWith()](https://www.programiz.com/java-programming/library/string/endswith) method.

**Java String endsWith()**

The syntax of the string endsWith() method is:

string.endsWith(String str)

Here, string is an object of the String class.

**endsWith() Parameters**

The endsWith() method takes a single parameter.

* **str** - check whether string ends with str or not

**endsWith() Return Value**

* **returns true** if the string ends with the given string
* **returns false** if the string doesn't end with the given string

**Example: Java endsWith() Without Offset Parameter**

class Main {

public static void main(String[] args) {

String str = "Java Programming";

System.out.println(str.endsWith("mming")); // true

System.out.println(str.endsWith("g")); // true

System.out.println(str.endsWith("a Programming")); // true

System.out.println(str.endsWith("programming")); // false

System.out.println(str.endsWith("Java")); // false

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

As you can see from the above example, endsWith() takes case (lower case and upper case) into consideration.

If you need to check whether the string begins with the specified string or not, use the [Java String startsWith()](https://www.programiz.com/java-programming/library/string/startswith) method.

**Java String isEmpty()**

The syntax of the string isEmpty() method is:

string.isEmpty()

Here, string is an object of the String class.

**isEmpty() Parameters**

The isEmpty() method does not take any parameters.

**isEmpty() Return Value**

* **returns true** if the string is empty (length is 0)
* **returns false** if the string is not empty

**Example: Java String isEmpty()**

class Main {

public static void main(String[] args) {

String str1 = "Java Programming";

String str2 = "";

System.out.println(str1.isEmpty()); // false

System.out.println(str2.isEmpty()); // true

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Note:** A non-initialized string is not an empty string. If you use isEmpty() on a string that is not initialized, you will get an error.

**Java String intern()**

The syntax of the string intern() method is:

string.intern()

Here, string is an object of the String class.

**intern() Parameters**

The intern() method does not take any parameters.

**intern() Return Value**

* returns a canonical representation of the string

**What is Java String Interning?**

The String interning ensures that all strings having the same contents use the same memory.

Suppose, we these two strings:

String str1 = "xyz";

String str2 = "xyz";

Since both str1 and str2 have the same contents, both these strings will share the same memory. Java automatically interns the string literals.

However, if you create strings with using the new keyword, these strings won't share the same memory. For example,

class Main {

public static void main(String[] args) {

String str1 = new String("xyz");

String str2 = new String("xyz");

System.out.println(str1 == str2); // false

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

As you can see from this example, both str1 and str2 have the same content. However, they are not equal because they don't share the same memory.

In this case, you can manually use the intern() method so that the same memory is used for strings having the same content.

**Example: Java String intern()**

class Main {

public static void main(String[] args) {

String str1 = new String("xyz");

String str2 = new String("xyz");

// str1 and str2 doesn't share the same memory pool

System.out.println(str1 == str2); // false

// using the intern() method

// now both str1 and str2 share the same memory pool

str1 = str1.intern();

str2 = str2.intern();

System.out.println(str1 == str2); // true

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

As you can see, both str1 and str2 have the same content, but they are not equal initially.

We then use the intern() method so that str1 and str2 use the same memory pool. After we use intern(), str1 and str2 are equal.

**Java String contentEquals()**

The syntax of the string contentEquals() method is:

string.contentEquals(StringBuffer sb)

string.contentEquals(charSequence cs)

Here, string is an object of the String class.

**contentEquals() Parameters**

The contentEquals() method takes a single parameter.

* either StringBuffer or charSequence

**Note:** You can pass any class that implements charSequence to the contentEquals() method. For example: String, StringBuffer, CharBuffer etc.

**contentEquals() Return Value**

* Returns true if the string contains the same sequence of characters as the specified parameter. If not, returns false.

**Example: Java String contentEquals()**

class Main {

public static void main(String[] args) {

String str = "Java";

String str1 = "Java";

StringBuffer sb1 = new StringBuffer("Java");

CharSequence cs1 = "Java";

String str2 = "JavA";

StringBuffer sb2 = new StringBuffer("JavA");

CharSequence cs2 = "JavA";

System.out.println(str.contentEquals(str1)); // true

System.out.println(str.contentEquals(sb1)); // true

System.out.println(str.contentEquals(cs1)); // true

System.out.println(str.contentEquals(str2)); // false

System.out.println(str.contentEquals(sb2)); // false

System.out.println(str.contentEquals(cs2)); // false

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Java String equals() Vs contentEquals()**

The Java String equals() method not only compares the content, but also checks if the other object is an instance of String. However, contentEquals() only compares the content. For example,

class Main {

public static void main(String[] args) {

String str1 = "Java";

StringBuffer sb1 = new StringBuffer("Java");

System.out.println(str1.equals(sb1)); // false

System.out.println(str1.contentEquals(sb1)); // true

}

}

Here, both str1 and sb1 have the same content but they are instance of different objects. Hence, str1.equals(sb1) returns false and str1.contentEquals(sb1) returns true.

**Java String hashCode()**

A hashcode is a number (object's memory address) generated from any object, not just strings. This number is used to store/retrieve objects quickly in a hashtable.

The syntax of the string hashCode() method is:

string.hashCode()

Here, string is an object of the String class.

**hashCode() Parameters**

The matches() method doesn't take any parameters.

**hashCode() Return Value**

* returns the hashcode, which is an int value, of the string

The hash code is computed using formula:

s[0]\*31(n-1) + s[1]\*31(n-2) + ... + s[n-1]

where,

* s[0] is the first element of string s, s[1] is the second element and so on.
* n is the length of the string

**Example: Java String hashCode()**

class Main {

public static void main(String[] args) {

String str1 = "Java";

String str2 = "Java Programming";

String str3 = "";

System.out.println(str1.hashCode()); // 2301506

System.out.println(str2.hashCode()); // 1377009627

// hash code of empty string is 0

System.out.println(str3.hashCode()); // 0

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Note**: For two strings to be equal, their hash code also must be equal.

**Java String subSequence()**

The syntax of the subSequence() method is:

string.subSequence(int startIndex, int endIndex)

Here, string is an object of the String class.

**subSequence() Parameters**

The subSequence() method takes two parameters.

* **startIndex** - the starting index
* **endIndex**- the ending index

**subSequence() Return Value**

* The subSequence() method returns a CharSequence.

**Example: Java String subSequence()**

class Main {

public static void main(String[] args) {

String str = "Java Programming";

System.out.println(str.subSequence(3, 8)); // a Pro

}

}