**Java String Handling: String Constructors, Special string operations, Character Extraction, String Comparisons, Modifying a string, String Buffer.**

**String** is a sequence of characters, for e.g. “Hello” is a string of 5 characters. In java, string is an immutable object which means it is constant and can cannot be changed once it is created.

**Creating a String**

There are two ways to create a String in Java

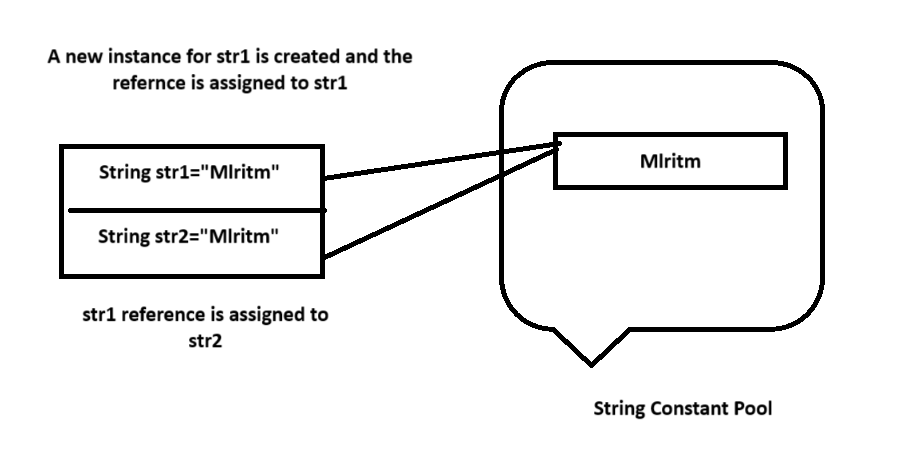
1. String literal
2. Using new keyword

**1. String literal**

A string literal is a sequence of characters enclosed in **double quotation marks (” “)**. In java, Strings can be created by assigning a String literal to a String instance:

String str1 = "Mlritm";

String str2 = "Mlritm";



**The problem with this approach**: As I stated in the beginning that String is an object in Java. However we have not created any string object using new keyword in the above statements.

The compiler does this internally and looks for the string in the memory (this memory is often referred as **string constant pool**). If the string is not found, it creates an object with the string value, which is “Mlritm” in this example and assign a reference to this string.

In our example, a reference to string “Mlritm” is copied to the string str1, however for str2, the compiler finds the string in string constant pool and doesn’t create the new object, rather assigns the same old reference to the string str2.

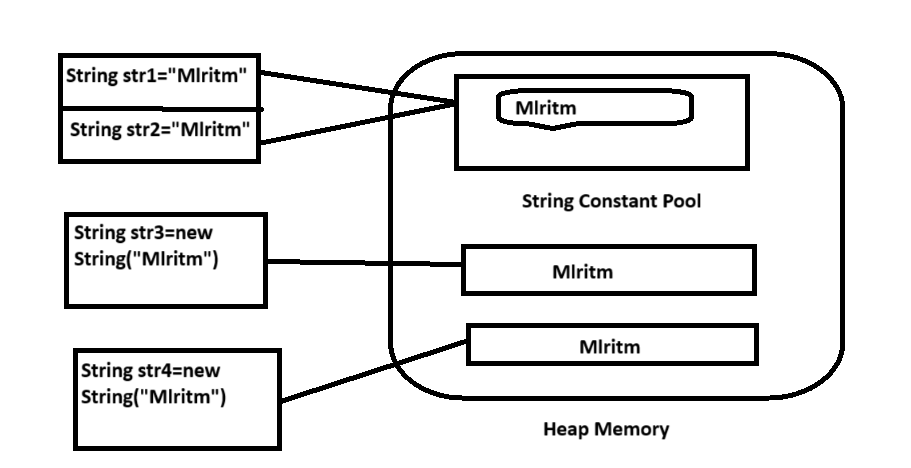
What if we want to have two different object with the same string? For that we would need to create strings using **new keyword**.

### 2. Using New Keyword

To create a new instance of a string, we use new keyword. When we create a string using new keyword, it gets created in heap memory rather than string constant pool as shown in the following diagram. When we create a string using new keyword, it always create a new string irrespective of whether the string is already present or not in the heap memory.

String str3 = new String("Mlritm");

String str4 = new String("Mlritm");



In this case compiler would create two different object in heap memory with the same string.

## Example 1: Java String vs new String

Let’s write a java string program to understand the difference of two different ways of creating a string in java.

class StrExample1

{

public static void main(String args[])

{

//creating string using string literal

String s1 = "Mlritm";

String s2 = "Mlritm";

if(s1 == s2)

{

System.out.println("String s1 and s2 are equal");

}

else

{

System.out.println("String s1 and s2 are NOT equal");

}

}

}

## Example 2: A Simple Java String Example

public class StrExample2

{

public static void main(String args[])

{

String str = "Mlritm";

//declaring a char array

char arrCh[]={'h','e','l','l','o'};

//converting char array arrCh[] to string str2

String str2 = new String(arrCh);

//creating another java string str3 by using new keyword

String str3 = new String("Java String Example");

//Displaying all the three strings

System.out.println(str);

System.out.println(str2);

System.out.println(str3);

}

}

**String constructors**

In Java, string class supports several types of constructors to create and initialize string objects based on the various types of arguments. This is an example of polymorphism.

The most commonly used constructors of the String class are as follows:

* String()
* String(String str)
* String(char chars[ ])
* String(char chars[ ], int startIndex, int count)
* String(byte byteArr[ ])
* String(byte byteArr[ ], int startIndex, int count)

1. **String():** To create an empty string, we will call the default constructor. The general syntax to create an empty string in Java program is as follows:

String s = new String();

It will create a string object in the heap area with no value.

1. **String(String str):** This is the most common string constructor that we generally use in the Java program. This constructor will create a new string object in the heap area and stores the given value in it. The general syntax to construct a string object with the specified string str is as follows:

String st = new String(String str);

For example:

String s2 = new String("Hello Java");

1. **String(char chars[ ]):** This string constructor creates a string object and stores the array of characters in it. The general syntax to create a string object with a specified array of characters is as follows:

String str = new String(char char[])

For example:

char chars[] = { 'a', 'b', 'c', 'd' };

String s3 = new String(chars);

**4. String(char chars[ ], int startIndex, int count):** This constructor creates and initializes a string object with a subrange of a character array.

The argument startIndex specifies the index at which the subrange begins and count specifies the number of characters to be copied. The general syntax to create a string object with the specified subrange of character array is as follows:

String str = new String(char chars[ ], int startIndex, int count);

For example:

char chars[] = { 'w', 'i', 'n', 'd', 'o', 'w', 's'  };

String str = new String(chars, 2, 3);

The object str contains the address of the value ”ndo” stored in the heap area because the starting index is 2 and the total number of characters to be copied is 3.

**5.String(byte byteArr[ ]):** This type of string constructor constructs a new string object by decoding the given array of bytes (i.e., by decoding ASCII values into the characters) according to the system’s default character set. Let’s take an example program based on this constructor.

public class ByteArray

{

public static void main(String[] args)

{

byte b[] = { 97, 98, 99, 100 }; // Range of bytes: -128 to 127. These byte values will be converted into corresponding characters.

String s = new String(b);

System.out.println(s);

}

}

1. 97 is the Unicode value of a, 98 ➨ b, 99 ➨ c, 100 ➨ d.  
2. 65 is the Unicode value of A.

**6. String(byte byteArr[ ], int startIndex, int count):** This constructor also creates a new string object by decoding the ASCII values using the system’s default character set.

public class ByteArray

{

public static void main(String[] args)

{

byte b[] = { 65, 66, 67, 68, 69, 70 };

// Range of bytes: -128 to 127.

String s = new String(b, 2, 4); // CDEF

System.out.println(s);

}

}

### Special String Operations

1. String Literals
2. String Concatenation
3. String Concatenation with Other Data Types
4. String Conversion and toString()

## 1. String Literals

A string literal is a sequence of characters enclosed in double quotes. In Java, string literals are used to create instances of the String class.

public class StringLiteralsExample

{

public static void main(String[] args)

{

String str1 = "Hello, World!";

String str2 = "Java Programming";

System.out.println(str1);

System.out.println(str2);

}

}

## 2. String Concatenation

String concatenation is the operation of joining two or more strings together. Java provides several ways to concatenate strings, including using the + operator and the concat() method.

public class StringConcatenationExample

{

public static void main(String[] args)

{

String str1 = "Hello";

String str2 = "World";

// Using the + operator

String result = str1 + ", " + str2 + "!";

System.out.println(result);

// Using the concat() method

String result1 = str1.concat(", ").concat(str2).concat("!");

System.out.println(result);

}

}

## 3. String Concatenation with Other Data Types

Java allows concatenation of strings with other data types using the + operator. The non-string operands are converted to their string representation before concatenation.

public class StringConcatenationWithTypesExample

{

public static void main(String[] args)

{

String str = "The answer is ";

int number = 42;

double pi = 3.14159;

// Concatenating strings with other data types

String result1 = str + number;

String result2 = "Pi is approximately " + pi;

System.out.println(result1);

System.out.println(result2);

}}

## 4. String Conversion and toString()

Java provides the toString() method to convert an object to its string representation. This method is defined in the Object class and can be overridden by custom classes.

public class StringConversionExample

{

public static void main(String[] args)

{

int number = 42;

double pi = 3.14159;

boolean bool = true;

// Converting different data types to string using String.valueOf()

String strNumber = String.valueOf(number);

String strPi = String.valueOf(pi);

String strBool = String.valueOf(bool);

System.out.println("String representation of number: " + strNumber);

System.out.println("String representation of pi: " + strPi);

System.out.println("String representation of boolean: " + strBool);

// Custom class conversion using toString()

Person person = new Person("Alice", 30);

System.out.println(person);

}

}

class Person

{

private String name;

private int age;

public Person(String name, int age)

{

this.name = name;

this.age = age;

}

@Override

public String toString() {

return "Person{name='" + name + "', age=" + age + '}';

}

}

* The String.valueOf() method is used to convert various data types to their string representation.
* The Person class overrides the toString() method to provide a custom string representation.

### Character Extraction

Java provides several methods to extract characters from a string. These methods are useful for manipulating and analyzing strings at the character level.

1. charAt()
2. getChars()
3. getBytes()
4. toCharArray()

**charAt( )**

* To extract **a character**from a string we can **directly refer**it using this method

**char charAt(int where)**

* where is the index to specify where the character is specified in a String.
* It will return a character at the specified index.

**getChars( )**

* We can **extract more than one character** at a time by using this method with the following general form:

**void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)**

* **sourceStart**specifies the index of the beginning of the substring.
* **sourceEnd**specifies an index that is one past the end of the desired substring.
* The array that will receive the characters is specified by**target.**
* The **index within target**at**which the substring will be copied**is passed in **targetStart.**

**getBytes( )**

* It stores the **characters in an array of bytes**.
* it uses the **default character-to-byte conversions**provided by the platform. The general form is:

**byte[ ] getBytes( )**

* It is most useful when we are **exporting a String value into an environment**that **does not support 16-bit Unicode characters**.

**toCharArray( )**

* It is used to convert **all the characters in a String object into a character array.**
* It returns **an array of characters**for the entire string with the following general form:

**char[ ] toCharArray( )**

public class CharacterExtraction

{

public static void main(String args[])

{

String str="Hello World";

char ch1 = str.charAt(3);

char ch2= "Bhuvana".charAt(2);

System.out.println("charAt() method output is::"+ch1);

System.out.println("charAt() method output is::"+ch2);

int start = 2;

int end = 6;

char buf[] = new char[end - start];

str.getChars(end start,buf,0);

System.out.println("getChars() method ouptput is::");

System.out.println(buf);

System.out.println();

byte barr[] = str.getBytes();

for(int i=0;i<str.length();i++)

System.out.print(barr[i]);

System.out.println();

char chararray[] = str.toCharArray();

System.out.println(chararray);

}}

**Output:**

**charAt() method output is::l  
charAt() method output is::u  
getChars() method ouptput is::  
llo**

**721011081081113287111114108100  
Hello World**

**String comparison**

Below are 5 ways to compare two [Strings](https://www.geeksforgeeks.org/string-class-in-java/) in Java:

1. Using user-defined function
2. Using String.equals()
3. Using String.equalsIgnoreCase()
4. Using Objects.equals()
5. Using String.compareTo()

**1.Using user-defined function:**

Define a function to compare values with the following conditions :

1. if (string1 > string2) it returns a **positive value**.
2. if both the strings are equal lexicographically i.e.(string1 == string2) it returns **0**.
3. if (string1 < string2) it returns a **negative value**.

**2. Using String.equals() :**

In Java, string equals() method compares the two given strings based on the data/content of the string. If all the contents of both the strings are same then it returns true. If any character does not match, then it returns false.

**Syntax:**

str1.equals(str2);

Here str1 and str2 both are the strings that are to be compared.

public class Main

{

public static void main(String[] arg)

{

String str1 = "java";

String str2 = "java";

System.out.println(str1.equals(str2));

}

}

**3. Using String.equalsIgnoreCase() :**

The [String.equalsIgnoreCase()](https://www.geeksforgeeks.org/equalsignorecase-in-java/) method compares two strings irrespective of the case (lower or upper) of the string. This method returns true if the argument is not null and the contents of both the Strings are same ignoring case, else false. **Syntax:**

str2.equalsIgnoreCase(str1);

Here str1 and str2 both are the strings which are to be compared.

public class Main

{

public static void main(String[] arg)

{

String str1 = "java";

String str2 = "Java";

System.out.println(str1.equalsIgnoreCase(str2));

}

}

**4.Using Objects.equals() :**

[Object.equals(Object a, Object b)](https://www.geeksforgeeks.org/java-util-objects-class-java/) method returns true if the arguments are equal to each other and false otherwise. Consequently, if both arguments are null, true is returned and if exactly one argument is null, false is returned. Otherwise, equality is determined by using the equals() method of the first argument.

**Syntax:**

public static boolean equals(Object a, Object b)

Here a and b both are the string objects which are to be compared.

**5. Using String.compareTo() for Comparing Two Strings**

**Syntax:**

int str1.compareTo(String str2)

**Working:** It compares and returns the following values as follows:

1. if (string1 > string2) it returns a **positive value**.
2. if both the strings are equal lexicographically i.e.(string1 == string2) it returns **0**.
3. if (string1 < string2) it returns a **negative value**.

public class CompareTo

{

public static void main(String[] arg)

{

String str1 = "aaa";

String str2 = "bbb";

String str3 = "ccc";

String str4 = "bbb";

System.out.println(str2.compareTo(str3));

System.out.println(str2.compareTo(str1));

System.out.println(str2.compareTo(str4));

}

}

**Modifying a string**

Java provides several methods to modify strings. These methods allow you to create new strings by modifying the content of existing strings, which are immutable in Java.

1. substring()
2. concat()
3. replace()
4. replaceAll()
5. replaceFirst()
6. toUpperCase()
7. toLowerCase()
8. trim()
9. split()

## 1. substring()

The substring() method returns a new string that is a substring of the original string. It takes one or two arguments: the start index and optionally the end index.

### Syntax:

public String substring(int beginIndex)

public String substring(int beginIndex, int endIndex) 5,8

**Example:**

public class SubstringExample {

public static void main(String[] args) {

String str = "Hello, World!";

String substr1 = str.substring(7);

String substr2 = str.substring(0, 5);

System.out.println("Substring from index 7: " + substr1);

System.out.println("Substring from index 0 to 5: " + substr2);

}

}

**Output:**

Substring from index 7: World!

Substring from index 0 to 5: Hello

**2. concat()**

The concat() method concatenates the specified string to the end of the original string.

**Syntax:**

public String concat(String str)

**Example:**

public class ConcatExample {

public static void main(String[] args) {

String str1 = "Hello";

String str2 = "World";

String result = str1.concat(", ").concat(str2).concat("!");

System.out.println("Concatenated string: " + result);

}

}

**Output:**

Concatenated string: Hello, World!

**3. replace()**

The replace() method replaces all occurrences of a specified character or substring with a new character or substring.

**Syntax:**

public String replace(char oldChar, char newChar)

public String replace(CharSequence target, CharSequence replacement)

**Example:**

public class ReplaceExample {

public static void main(String[] args) {

String str = "Hello, World!";

String result1 = str.replace('o', '0');

String result2 = str.replace("World", "Java");

System.out.println("Replaced 'o' with '0': " + result1);

System.out.println("Replaced 'World' with 'Java': " + result2);

}

}

**Output:**

Replaced 'o' with '0': Hell0, W0rld!

Replaced 'World' with 'Java': Hello, Java!

**4. replaceAll()**

The replaceAll() method replaces each substring that matches the given regular expression with the given replacement.

**Syntax:**

public String replaceAll(String regex, String replacement)

**Example:**

public class ReplaceAllExample {

public static void main(String[] args) {

String str = "Hello123World";

String result = str.replaceAll("[0-9]", "#");

System.out.println("Replaced all digits with '#': " + result);

}

}

**Output:**

Replaced all digits with '#': Hello###World

**5. replaceFirst()**

The replaceFirst() method replaces the first substring that matches the given regular expression with the given replacement.

**Syntax:**

public String replaceFirst(String regex, String replacement)

**Example:**

public class ReplaceFirstExample {

public static void main(String[] args) {

String str = "Hello123World123";

String result = str.replaceFirst("[0-9]", "#");

System.out.println("Replaced first digit with '#': " + result);

}

}

**Output:**

Replaced first digit with '#': Hello#23World123

**6. toUpperCase()**

The toUpperCase() method converts all characters in the string to uppercase.

**Syntax:**

public String toUpperCase()

**Example:**

public class ToUpperCaseExample {

public static void main(String[] args) {

String str = "Hello, World!";

String result = str.toUpperCase();

System.out.println("Uppercase string: " + result);

}

}

**Output:**

Uppercase string: HELLO, WORLD!

**7. toLowerCase()**

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

**Syntax:**

public String toLowerCase()

**Example:**

public class ToLowerCaseExample {

public static void main(String[] args) {

String str = "Hello, World!";

String result = str.toLowerCase();

System.out.println("Lowercase string: " + result);

}

}

**Output:**

Lowercase string: hello, world!

**8. trim()**

The trim() method removes whitespace from both ends of the string.

**Syntax:**

public String trim()

**Example:**

public class TrimExample {

public static void main(String[] args) {

String str = " Hello, World! ";

String result = str.trim();

System.out.println("Trimmed string: '" + result + "'");

}

}

**Output:**

Trimmed string: 'Hello, World!'

**9. split()**

The split() method splits the string into an array of substrings based on the specified delimiter.

**Syntax:**

public String[] split(String regex)

public String[] split(String regex, int limit)

**Example:**

import java.util.Arrays;

public class SplitExample {

public static void main(String[] args) {

String str = "apple,banana,cherry";

String[] result = str.split(",");

System.out.println("Split string: " + Arrays.toString(result));

}

}

**Output:**

Split string: [apple, banana, cherry]

**StringBuffer class**

StringBuffer is a class in Java that represents a mutable sequence of characters. It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.

**Here are some important features and methods of the StringBuffer class:**

* StringBuffer objects are mutable, meaning that you can change the contents of the buffer without creating a new object.
* The initial capacity of a StringBuffer can be specified when it is created, or it can be set later with the ensureCapacity() method.
* The append() method is used to add characters, strings, or other objects to the end of the buffer.
* The insert() method is used to insert characters, strings, or other objects at a specified position in the buffer.
* The delete() method is used to remove characters from the buffer.
* The reverse() method is used to reverse the order of the characters in the buffer.

public class StringBufferExample {

public static void main(String[] args)

{

StringBuffer sb = new StringBuffer();

sb.append("Hello");

sb.append(" ");

sb.append("world");

String message = sb.toString();

System.out.println(message);

}

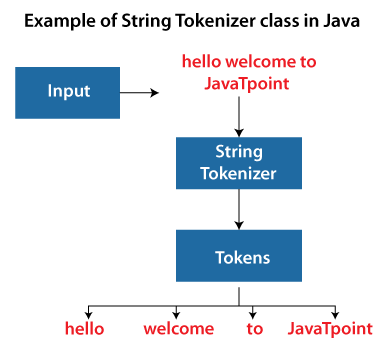
}

**Java StringTokenizer class**

The java.util.StringTokenizer class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

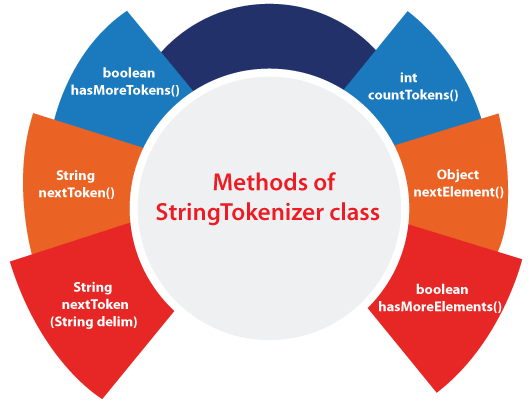
In the StringTokenizer class, the delimiters can be provided at the time of creation or one by one to the tokens.



|  |  |
| --- | --- |
| Constructor | Description |
| StringTokenizer(String str) | It creates StringTokenizer with specified string. |
| StringTokenizer(String str, String delim) | It creates StringTokenizer with specified string and delimiter. |
| StringTokenizer(String str, String delim, boolean returnValue) | It creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens. |

Methods of the StringTokenizer Class

The six useful methods of the StringTokenizer class are as follows:



|  |  |
| --- | --- |
| Methods | Description |
| boolean hasMoreTokens() | It checks if there is more tokens available. |
| String nextToken() | It returns the next token from the StringTokenizer object. |
| String nextToken(String delim) | It returns the next token based on the delimiter. |
| boolean hasMoreElements() | It is the same as hasMoreTokens() method. |
| Object nextElement() | It is the same as nextToken() but its return type is Object. |
| int countTokens() | It returns the total number of tokens. |