**Java String**

In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

char[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=new String(ch);

is same as:

String s="javatpoint";

Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.



The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.

**CharSequence Interface**

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.

The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

There are two ways to create String object:

1) String Literal

Java String literal is created by using double quotes. For Example:

String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

String s1="Welcome";

String s2="Welcome"; //will not create new instance

In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as string constant pool.

Why java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool)

2) By new keyword

String s=**new** String("Welcome"); //creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap (non pool).

Java String Example

**public** **class** StringExample {

**public** **static** **void** main(String args[]) {

String s1 = "java";// creating string by java string literal

**char** ch[] = { 's', 't', 'r', 'i', 'n', 'g', 's' };

String s2 = **new** String(ch);// converting char array to string

String s3 = **new** String("example");// creating java string by new keyword

System.***out***.println(s1);

System.***out***.println(s2);

System.***out***.println(s3);

}

}

**Output:**

java

strings

example

**Java String class methods**

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | char charAt(int index) | returns char value for the particular index |
| 2 | int length() | returns string length |
| 3 | static String format(String format, Object... args) | returns formatted string |
| 4 | static String format(Locale l, String format, Object... args) | returns formatted string with given locale |
| 5 | String substring(int beginIndex) | returns substring for given begin index |
| 6 | String substring(int beginIndex, int endIndex) | returns substring for given begin index and end index |
| 7 | boolean contains(CharSequence s) | returns true or false after matching the sequence of char value |
| 8 | static String join(CharSequence delimiter, CharSequence... elements) | returns a joined string |
| 9 | static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) | returns a joined string |
| 10 | boolean equals(Object another) | checks the equality of string with object |
| 11 | boolean isEmpty() | checks if string is empty |
| 12 | String concat(String str) | concatinates specified string |
| 13 | String replace(char old, char new) | replaces all occurrences of specified char value |
| 14 | String replace(CharSequence old, CharSequence new) | replaces all occurrences of specified CharSequence |
| 15 | static String equalsIgnoreCase(String another) | compares another string. It doesn't check case. |
| 16 | String[] split(String regex) | returns splitted string matching regex |
| 17 | String[] split(String regex, int limit) | returns splitted string matching regex and limit |
| 18 | String intern() | returns interned string |
| 19 | int indexOf(int ch) | returns specified char value index |
| 20 | int indexOf(int ch, int fromIndex) | returns specified char value index starting with given index |
| 21 | int indexOf(String substring) | returns specified substring index |
| 22 | int indexOf(String substring, int fromIndex) | returns specified substring index starting with given index |
| 23 | String toLowerCase() | returns string in lowercase. |
| 24 | String toLowerCase(Locale l) | returns string in lowercase using specified locale. |
| 25 | String toUpperCase() | returns string in uppercase. |
| 26 | String toUpperCase(Locale l) | returns string in uppercase using specified locale. |
| 27 | String trim() | removes beginning and ending spaces of this string. |
| 28 | static String valueOf(int value) | converts given type into string. It is overloaded. |

**Immutable String in Java**

In java, string objects are immutable. Immutable simply means unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

**class** Testimmutablestring {

**public** **static** **void** main(String args[]) {

String s = "Sachin";

s.concat(" Tendulkar");// concat() method appends the string at the end

System.***out***.println(s);// will print Sachin because strings are immutable objects

}

}

Output:

Sachin

**class** Testimmutablestring1 {

**public** **static** **void** main(String args[]) {

String s = "Sachin";

s = s.concat(" Tendulkar");

System.***out***.println(s);

}

}

Output:

Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

**Why string objects are immutable in java?**

Because java uses the concept of string literal. Suppose there are 5 reference variables, all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

**Java String compare**

We can compare string in java on the basis of content and reference. There are three ways to compare string in java:

* By equals() method
* By = = operator
* By compareTo() method

**1) String compare by equals() method**

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

public boolean equals(Object another) compares this string to the specified object.

public boolean equalsIgnoreCase(String another) compares this String to another string, ignoring case.

**class** Teststringcomparison1 {

**public** **static** **void** main(String args[]) {

String s1 = "Sachin";

String s2 = "Sachin";

String s3 = **new** String("Sachin");

String s4 = "Saurav";

System.***out***.println(s1.equals(s2));// true

System.***out***.println(s1.equals(s3));// true

System.***out***.println(s1.equals(s4));// false

}

}

**Output:**

true

true

false

**class** Teststringcomparison2 {

**public** **static** **void** main(String args[]) {

String s1 = "Sachin";

String s2 = "SACHIN";

System.***out***.println(s1.equals(s2));// false

System.***out***.println(s1.equalsIgnoreCase(s2));// true

}

}

**Output:**

false

true

**2) String compare by == operator**

The = = operator compares references not values.

**class** Teststringcomparison3 {

**public** **static** **void** main(String args[]) {

String s1 = "Sachin";

String s2 = "Sachin";

String s3 = **new** String("Sachin");

System.***out***.println(s1 == s2);// true (because both refer to same instance)

System.***out***.println(s1 == s3);// false(because s3 refers to instance created in nonpool)

}

}

Output:

true

false

**3) String compare by compareTo() method**

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

s1 == s2 :0

s1 > s2 :positive value

s1 < s2 :negative value

**class** Teststringcomparison4 {

**public** **static** **void** main(String args[]) {

String s1 = "Sachin";

String s2 = "Sachin";

String s3 = "Ratan";

System.***out***.println(s1.compareTo(s2));// 0

System.***out***.println(s1.compareTo(s3));// 1(because s1>s3)

System.***out***.println(s3.compareTo(s1));// -1(because s3 < s1 )

}

}

Output:

0

1

-1

**String Concatenation in Java**

In java, string concatenation forms a new string that is the combination of multiple strings. There are two ways to concat string in java:

1) String Concatenation by + (string concatenation) operator

Java string concatenation operator (+) is used to add strings. For Example:

**class** TestStringConcatenation1 {

**public** **static** **void** main(String args[]) {

String s = "Sachin" + " Tendulkar";

System.***out***.println(s);// Sachin Tendulkar

}

}

**Output:**

Sachin Tendulkar

The Java compiler transforms above code to this:

String s = (**new** StringBuilder()).append("Sachin").append("Tendulkar”).toString();

**Substring in Java**

A part of string is called substring. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.

Note: Index starts from 0.

You can get substring from the given string object by one of the two methods:

public String substring(int startIndex): This method returns new String object containing the substring of the given string from specified startIndex (inclusive).

public String substring(int startIndex, int endIndex): This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

In case of string:

startIndex: inclusive

endIndex: exclusive

Let's understand the startIndex and endIndex by the code given below.

String s="hello";

System.out.println(s.substring(0,2));//he

In the above substring, 0 points to h but 2 points to e (because end index is exclusive).

Example of java substring

**public** **class** TestSubstring {

**public** **static** **void** main(String args[]) {

String s = "SachinTendulkar";

System.***out***.println(s.substring(6));// Tendulkar

System.***out***.println(s.substring(0, 6));// Sachin

}

}

**Output:**

Tendulkar

Sachin

**Java String class methods**

Let's see the important methods of String class.

Java String toUpperCase() and toLowerCase() method

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

**public** **class** string1 {

**public** **static** **void** main(String args[]) {

String s="Sachin";

System.***out***.println(s.toUpperCase());//SACHIN

System.***out***.println(s.toLowerCase());//sachin

System.***out***.println(s);//Sachin(no change in original)

}

}

**Output:**

SACHIN

sachin

Sachin

**Java String getChars()**

The java string getChars() method copies the content of this string into specified char array. There are 4 arguments passed in getChars() method. The signature of getChars() method is given below:

**public** **void** getChars(**int** srcBeginIndex, **int** srcEndIndex, **char**[] destination, **int** dstBeginIndex)

**Returns**

It doesn't return any value.

**Throws**

It throws StringIndexOutOfBoundsException if beginIndex is greater than endIndex.

Java String getChars() method example

**public** **class** StringGetCharsExample {

**public** **static** **void** main(String args[]) {

String str = **new** String("hello javatpoint how r u");

**char**[] ch = **new** **char**[15];

**try** {

str.getChars(6, 16, ch, 2);

System.***out***.println(ch);

} **catch** (Exception ex) {

System.***out***.println(ex);

}

}

}

**Output:**

javatpoint

**Java StringBuffer class**

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

Important Constructors of StringBuffer class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| StringBuffer() | creates an empty string buffer with the initial capacity of 16. |
| StringBuffer(String str) | creates a string buffer with the specified string. |
| StringBuffer(int capacity) | creates an empty string buffer with the specified capacity as length. |

**What is mutable string?**

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

**1) StringBuffer append() method**

The append() method concatenates the given argument with this string.

**class** StringBufferExample {

**public** **static** **void** main(String args[]) {

StringBuffer sb = **new** StringBuffer("Hello ");

sb.append("Java");// now original string is changed

System.***out***.println(sb);// prints Hello Java

}

}

Output:

Hello Java

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

**class** StringBufferExample2 {

**public** **static** **void** main(String args[]) {

StringBuffer sb = **new** StringBuffer("Hello ");

sb.insert(1, "Java");// now original string is changed

System.***out***.println(sb);// prints HJavaello

}

}

**Output:**

HJavaello

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

**class** StringBufferExample3 {

**public** **static** **void** main(String args[]) {

StringBuffer sb = **new** StringBuffer("Hello");

sb.replace(1, 3, "Java");

System.***out***.println(sb);// prints HJavalo

}

}

Output:

HJavalo

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

**class** StringBufferExample4 {

**public** **static** **void** main(String args[]) {

StringBuffer sb = **new** StringBuffer("Hello");

sb.delete(1, 3);

System.***out***.println(sb);// prints Hlo

}

}

Output:

Hlo

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

**class** StringBufferExample5 {

**public** **static** **void** main(String args[]) {

StringBuffer sb = **new** StringBuffer("Hello");

sb.reverse();

System.***out***.println(sb);// prints olleH

}

}

Output:

olleH

**Java StringBuilder class**

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Important Constructors of StringBuilder class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| StringBuilder() | creates an empty string Builder with the initial capacity of 16. |
| StringBuilder(String str) | creates a string Builder with the specified string. |
| StringBuilder(int length) | creates an empty string Builder with the specified capacity as length. |

**Important methods of StringBuilder class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public StringBuilder append(String s) | is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc. |
| public StringBuilder insert(int offset, String s) | is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc. |
| public StringBuilder replace(int startIndex, int endIndex, String str) | is used to replace the string from specified startIndex and endIndex. |
| public StringBuilder delete(int startIndex, int endIndex) | is used to delete the string from specified startIndex and endIndex. |
| public StringBuilder reverse() | is used to reverse the string. |
| public int capacity() | is used to return the current capacity. |
| public void ensureCapacity(int minimumCapacity) | is used to ensure the capacity at least equal to the given minimum. |
| public char charAt(int index) | is used to return the character at the specified position. |
| public int length() | is used to return the length of the string i.e. total number of characters. |
| public String substring(int beginIndex) | is used to return the substring from the specified beginIndex. |
| public String substring(int beginIndex, int endIndex) | is used to return the substring from the specified beginIndex and endIndex. |

**Difference between String and StringBuffer**

|  |  |  |
| --- | --- | --- |
| **No.** | **String** | **StringBuffer** |
| 1) | String class is immutable. | StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

**Performance Test of String and StringBuffer**

**public** **class** ConcatTest {

**public** **static** String concatWithString() {

String t = "Java";

**for** (**int** i = 0; i < 10000; i++) {

t = t + "Tpoint";

}

**return** t;

}

**public** **static** String concatWithStringBuffer() {

StringBuffer sb = **new** StringBuffer("Java");

**for** (**int** i = 0; i < 10000; i++) {

sb.append("Tpoint");

}

**return** sb.toString();

}

**public** **static** **void** main(String[] args) {

**long** startTime = System.*currentTimeMillis*();

*concatWithString*();

System.***out***.println("Time taken by Concating with String: " + (System.*currentTimeMillis*() - startTime) + "ms");

startTime = System.*currentTimeMillis*();

*concatWithStringBuffer*();

System.***out***.println(

"Time taken by Concating with StringBuffer: " + (System.*currentTimeMillis*() - startTime) + "ms");

}

}

**Output:**

Time taken by Concating with String: 180ms

Time taken by Concating with StringBuffer: 1ms

**String and StringBuffer HashCode Test**

As you can see in the program given below, String returns new hashcode value when you concat string but StringBuffer returns same.

**public** **class** InstanceTest {

**public** **static** **void** main(String args[]) {

System.***out***.println("Hashcode test of String:");

String str = "java";

System.***out***.println(str.hashCode());

str = str + "tpoint";

System.***out***.println(str.hashCode());

System.***out***.println("Hashcode test of StringBuffer:");

StringBuffer sb = **new** StringBuffer("java");

System.***out***.println(sb.hashCode());

sb.append("tpoint");

System.***out***.println(sb.hashCode());

}

}

**Output:**

Hashcode test of String:

3254818

229541438

Hashcode test of StringBuffer:

1476011703

1476011703

The hash code for a String object is computed as −

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

Using int arithmetic, where s[i] is the ith character of the string, n is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

**package** package1;

**public** **class** HashCode {

**public** **static** **void** main(String args[]) {

String Str = **new** String("Jii");

**char** Str\_array[] = Str.toCharArray();

**int** Str\_length = Str.length();

**int** sum = 0;

**for** (**int** i = 0; i < Str.length(); i++) {

sum = (int) (sum + ((int) Str\_array[i])\*(java.lang.Math.pow(31, Str\_length-(i+1))));

}

System.***out***.println("Hashcode for Str :" + Str.hashCode() );

System.***out***.println("Calculated HashCode = " + sum);

}

}

**Difference between StringBuffer and StringBuilder**

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is less efficient than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |

StringBuffer Example

**public** **class** BufferTest{

**public** **static** **void** main(String[] args){

StringBuffer buffer=**new** StringBuffer("hello");

buffer.append("java");

System.***out***.println(buffer);

}

}

Output:

Hellojava

Java String FAQs or Interview Questions

1) How many objects will be created in the following code?

String s1="javatpoint";

String s2="javatpoint";

Answer: Only one.

2) What is the difference between equals() method and == operator?

The equals() method matches content of the strings whereas == operator matches object or reference of the strings.

3) Is String class final?

Answer: Yes.

4) How to reverse String in java?

Input:

this is javatpoint

Output:

tnioptavaj si siht

5) How to check Palindrome String in java?

Input:

nitin

Output:

true

Input:

jatin

Output:

false

6) Write a java program to capitalize each word in string?

Input:

this is javatpoint

Output:

This Is Javatpoint

7) Write a java program to reverse each word in string?

Input:

this is javatpoint

Output:

siht si tnioptavaj

8) Write a java program to tOGGLE each word in string?

Input:

this is javatpoint

Output:

tHIS iS jAVATPOINT

9) Write a java program reverse tOGGLE each word in string?

Input:

this is javatpoint

Output:

sIHT sI tNIOPTAVAJ

10) What is the difference between String and StringBuffer in java?

11) What is the difference between StringBuffer and StringBuilder in java?

12) What does intern() method in java?

13) How to convert String to int in java?

14) How to convert int to String in java?

15) How to convert String to Date in java?

16) How to Optimize Java String Creation?

17) Java Program to check whether two Strings are anagram or not

18) Java program to find the percentage of uppercase, lowercase, digits and special characters in a String

19) How to convert String to Integer and Integer to String in Java

20) Java Program to find duplicate characters in a String

21) Java Program to prove that strings are immutable in java

22) Java Program to remove all white spaces from a String

23) Java Program to check whether one String is a rotation of another

24) Java Program to count the number of words in a String

25) Java Program to reverse a given String with preserving the position of space

26) How to swap two String variables without third variable

27) How to remove a particular character from a String