### **Strings in JAVA:**

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

### **How to create a string object?**

### There are two ways to create String object:

### **1) String Literal**

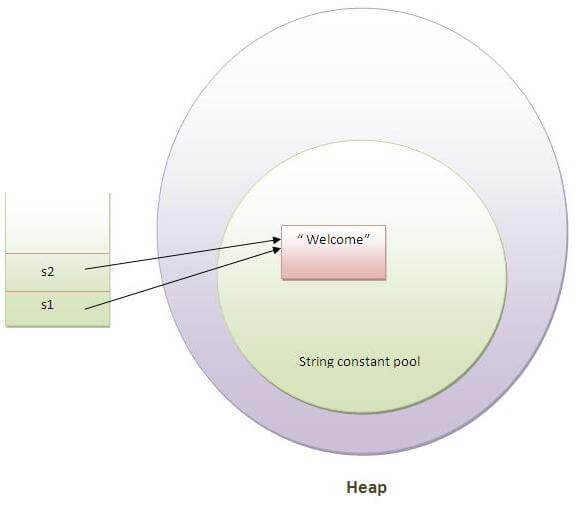
Java String literal is created by using double quotes.

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 the 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";//It doesn't create a 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, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

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

### **Why Java uses the concept of String literal?**

To make Java more memory efficient (because no new objects are created if it exists already in the 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 a heap (non-pool).

**Example:**

**package** jAVASTRINGS;

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

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

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

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

**char**[] ch = {'S','c','o','n','d',' ','S','t','r','i','n','g'};

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

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

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

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

//we can declare strings in above 3 ways

}

}

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 the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.



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

We will discuss immutable string later. Let's first understand what is String in Java and how to create the String object.

# Immutable String in Java

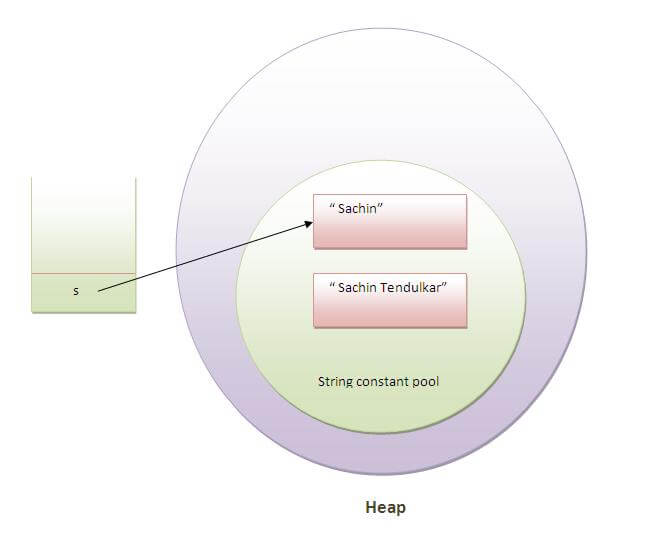
In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

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

Example:

1. **class** Testimmutablestring{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s.concat(" Tendulkar");//concat() method appends the string at the end
5. System.out.println(s);//will print Sachin because strings are immutable objects
6. }
7. }

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with “Sachin Tendulkar”. That is why string is known as immutable.



As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

String s1 = " Sachin ";

s1 = s1.concat(" Tendulkar ");

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

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.  **package** jAVASTRINGS;  **public** **class** ImmutableString {    **public** **static** **void** main(String[] args) {    String s = "Sachin";  s.concat("Tendulkar");  System.***out***.println(s);      String s1 = " Sachin ";  s1 = s1.concat(" Tendulkar ");  System.***out***.println(s1);  }  } |

# Java String compare

We can compare string in java on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

There are three ways to compare string in java:

1. By equals() method
2. By = = operator
3. 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.

1. **class** Teststringcomparison1{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3=**new** String("Sachin");
6. String s4="Saurav";
7. System.out.println(s1.equals(s2));//true
8. System.out.println(s1.equals(s3));//true
9. System.out.println(s1.equals(s4));//false
10. }
11. }

Output:true

true

false

1. **class** Teststringcomparison2{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="SACHIN";
6. System.out.println(s1.equals(s2));//false
7. System.out.println(s1.equalsIgnoreCase(s2));//true
8. }
9. }

Output:

false

true

## 2) String compare by == operator

The = = operator compares references not values.

1. **class** Teststringcomparison3{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3=**new** String("Sachin");
6. System.out.println(s1==s2);//true (because both refer to same instance)
7. System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
8. }
9. }

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

1. **class** Teststringcomparison4{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3="Ratan";
6. System.out.println(s1.compareTo(s2));//0
7. System.out.println(s1.compareTo(s3));//1(because s1>s3)
8. System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
9. }
10. }

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. By + (string concatenation) operator
2. By concat() method

## 1) String Concatenation by + (string concatenation) operator

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

1. **class** TestStringConcatenation1{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin"+" Tendulkar";
4. System.out.println(s);//Sachin Tendulkar
5. }
6. }

Output:Sachin Tendulkar

The **Java compiler transforms** above code to this:

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

In java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and its append method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also. For Example:

1. **class** TestStringConcatenation2{
2. **public** **static** **void** main(String args[]){
3. String s=50+30+"Sachin"+40+40;
4. System.out.println(s);//80Sachin4040
5. }
6. }

o/p : 80Sachin4040

#### **Note: After a string literal, all the + will be treated as string concatenation operator.**

### **2) String Concatenation by concat() method**

The String concat() method concatenates the specified string to the end of current string. Syntax:

1. **public** String concat(String another)

Let's see the example of String concat() method.

1. **class** TestStringConcatenation3{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin ";
4. String s2="Tendulkar";
5. String s3=s1.concat(s2);
6. System.out.println(s3);//Sachin Tendulkar
7. }
8. }

Sachin Tendulkar

# 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.

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

1. **public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
2. **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.

1. String s="hello";
2. 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

1. **public** **class** TestSubstring{
2. **public** **static** **void** main(String args[]){
3. String s="SachinTendulkar";
4. System.out.println(s.substring(6));//Tendulkar
5. System.out.println(s.substring(0,6));//Sachin
6. }
7. }

o/p:Tendulkar

Sachin

# Java String class methods

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

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.

1. String s="Sachin";
2. System.out.println(s.toUpperCase());//SACHIN
3. System.out.println(s.toLowerCase());//sachin
4. System.out.println(s);//Sachin(no change in original)

SACHIN

sachin

Sachin

### **Java String trim() method**

The string trim() method eliminates white spaces before and after string.

1. String s="  Sachin  ";
2. System.out.println(s);//  Sachin
3. System.out.println(s.trim());//Sachin

Sachin

Sachin

### **Java String startsWith() and endsWith() method**

1. String s="Sachin";
2. System.out.println(s.startsWith("Sa"));//true
3. System.out.println(s.endsWith("n"));//true

true

true

### **Java String charAt() method**

The string charAt() method returns a character at specified index.

1. String s="Sachin";
2. System.out.println(s.charAt(0));//S
3. System.out.println(s.charAt(3));//h

S

h

### **Java String length() method**

The string length() method returns length of the string.

1. String s="Sachin";
2. System.out.println(s.length());//6

6

### **Java String intern() method**

A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

1. String s=**new** String("Sachin");
2. String s2=s.intern();
3. System.out.println(s2);//Sachin

Sachin

### **Java String valueOf() method**

The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

1. **int** a=10;
2. String s=String.valueOf(a);
3. System.out.println(s+10);

Output:

1010

### **Java String replace() method**

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

1. String s1="Java is a programming language. Java is a platform. Java is an Island.";
2. String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
3. System.out.println(replaceString);

Output:

Kava is a programming language. Kava is a platform. Kava is an Island.

# 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.

# 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.

#### **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. |
|  |  |

### **Important methods of StringBuffer class**

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| public synchronized StringBuffer | 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 synchronized StringBuffer | 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 synchronized StringBuffer | replace(int startIndex, int endIndex, String str) | is used to replace the string from specified startIndex and endIndex. |
| public synchronized StringBuffer | delete(int startIndex, int endIndex) | is used to delete the string from specified startIndex and endIndex. |
| public synchronized StringBuffer | 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. |

### **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.

1. class StringBufferExample{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello ");
4. sb.append("Java");//now original string is changed
5. System.out.println(sb);//prints Hello Java
6. }
7. }

### **2) StringBuffer insert() method**

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

1. class StringBufferExample2{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello ");
4. sb.insert(1,"Java");//now original string is changed
5. System.out.println(sb);//prints HJavaello
6. }
7. }

### **3) StringBuffer replace() method**

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

1. class StringBufferExample3{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.replace(1,3,"Java");
5. System.out.println(sb);//prints HJavalo
6. }
7. }

### **4) StringBuffer delete() method**

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

1. class StringBufferExample4{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.delete(1,3);
5. System.out.println(sb);//prints Hlo
6. }

}

**5) StringBuffer reverse() method**

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

1. class StringBufferExample5{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.reverse();
5. System.out.println(sb);//prints olleH
6. }

}

**6) StringBuffer capacity() method**

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

1. class StringBufferExample6{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2
9. }  }

### **7) StringBuffer ensureCapacity() method**

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

1. class StringBufferExample7{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2
9. sb.ensureCapacity(10);//now no change
10. System.out.println(sb.capacity());//now 34
11. sb.ensureCapacity(50);//now (34\*2)+2
12. System.out.println(sb.capacity());//now 70
13. }

}

Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

|  |  |  |
| --- | --- | --- |
| **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. |

[**Java Swing**](https://www.javatpoint.com/java-swing) [**JavaFX**](https://www.javatpoint.com/javafx-tutorial) [**Java Applet**](https://www.javatpoint.com/java-applet) [**Java Reflection**](https://www.javatpoint.com/java-reflection) [**Java Date**](https://www.javatpoint.com/java-date) [**Java Conversion**](https://www.javatpoint.com/java-string-to-int) [**Java Collection**](https://www.javatpoint.com/collections-in-java) [**Java JDBC**](https://www.javatpoint.com/java-jdbc) [**Java New Features**](https://www.javatpoint.com/New-features-in-java) [**RMI**](https://www.javatpoint.com/RMI) [**Internationalization**](https://www.javatpoint.com/internationalization-in-java) [**Interview Questions**](https://www.javatpoint.com/corejava-interview-questions)

JavaTpoint

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**next →**](https://www.javatpoint.com/difference-between-stringbuffer-and-stringbuilder) [**← prev**](https://www.javatpoint.com/StringBuilder-class) Difference between String and StringBuffer There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:   |  |  |  | | --- | --- | --- | | **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. |   String vs StringBuffer Performance Test of String and StringBuffer  1. **public** **class** ConcatTest{ 2. **public** **static** String concatWithString()    { 3. String t = "Java"; 4. **for** (**int** i=0; i<10000; i++){ 5. t = t + "Tpoint"; 6. } 7. **return** t; 8. } 9. **public** **static** String concatWithStringBuffer(){ 10. StringBuffer sb = **new** StringBuffer("Java"); 11. **for** (**int** i=0; i<10000; i++){ 12. sb.append("Tpoint"); 13. } 14. **return** sb.toString(); 15. } 16. **public** **static** **void** main(String[] args){ 17. **long** startTime = System.currentTimeMillis(); 18. concatWithString(); 19. System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms"); 20. startTime = System.currentTimeMillis(); 21. concatWithStringBuffer(); 22. System.out.println("Time taken by Concating with  StringBuffer: "+(System.currentTimeMillis()-startTime)+"ms"); 23. } 24. }   OutPut:  Time taken by Concating with String: 578ms  Time taken by Concating with StringBuffer: 0ms 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.   1. **public** **class** InstanceTest{ 2. **public** **static** **void** main(String args[]){ 3. System.out.println("Hashcode test of String:"); 4. String str="java"; 5. System.out.println(str.hashCode()); 6. str=str+"tpoint"; 7. System.out.println(str.hashCode()); 9. System.out.println("Hashcode test of StringBuffer:"); 10. StringBuffer sb=**new** StringBuffer("java"); 11. System.out.println(sb.hashCode()); 12. sb.append("tpoint"); 13. System.out.println(sb.hashCode()); 14. } 15. }   O/P:  Hashcode test of String:  3254818  229541438  Hashcode test of StringBuffer:  118352462  118352462 |

Difference between StringBuffer and StringBuilder

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder. The StringBuilder class is introduced since JDK 1.5.

A list of differences between StringBuffer and StringBuilder are given below:

|  |  |  |
| --- | --- | --- |
| **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

1. //Java Program to demonstrate the use of StringBuffer class.
2. **public** **class** BufferTest{
3. **public** **static** **void** main(String[] args){
4. StringBuffer buffer=**new** StringBuffer("hello");
5. buffer.append("java");
6. System.out.println(buffer);
7. }
8. }

hellojava

## StringBuilder Example

1. //Java Program to demonstrate the use of StringBuilder class.
2. **public** **class** BuilderTest{
3. **public** **static** **void** main(String[] args){
4. StringBuilder builder=**new** StringBuilder("hello");
5. builder.append("java");
6. System.out.println(builder);
7. }
8. }

hellojava

## Performance Test of StringBuffer and StringBuilder

Let's see the code to check the performance of StringBuffer and StringBuilder classes.

1. //Java Program to demonstrate the performance of StringBuffer and StringBuilder classes.
2. **public** **class** ConcatTest{
3. **public** **static** **void** main(String[] args){
4. **long** startTime = System.currentTimeMillis();
5. StringBuffer sb = **new** StringBuffer("Java");
6. **for** (**int** i=0; i<10000; i++){
7. sb.append("Tpoint");
8. }
9. System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");
10. startTime = System.currentTimeMillis();
11. StringBuilder sb2 = **new** StringBuilder("Java");
12. **for** (**int** i=0; i<10000; i++){
13. sb2.append("Tpoint");
14. }
15. System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");
16. }
17. }

Time taken by StringBuffer: 16ms

Time taken by StringBuilder: 0ms

# How to create Immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

### **Example to create Immutable class**

|  |
| --- |
| In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method. |

1. **public** **final** **class** Employee{
2. **final** String pancardNumber;
4. **public** Employee(String pancardNumber){
5. **this**.pancardNumber=pancardNumber;
6. }
8. **public** String getPancardNumber(){
9. **return** pancardNumber;
10. }
12. }

The above class is immutable because:

* The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
* The class is final so we cannot create the subclass.
* There is no setter methods i.e. we have no option to change the value of the instance variable. These points makes this class as immutable.

# Java toString() method

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

## Advantage of Java toString() method

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

### **Understanding problem without toString() method**

Let's see the simple code that prints reference.

1. **class** Student{
2. **int** rollno;
3. String name;
4. String city;
6. Student(**int** rollno, String name, String city){
7. **this**.rollno=rollno;
8. **this**.name=name;
9. **this**.city=city;
10. }
12. **public** **static** **void** main(String args[]){
13. Student s1=**new** Student(101,"Raj","lucknow");
14. Student s2=**new** Student(102,"Vijay","ghaziabad");
16. System.out.println(s1);//compiler writes here s1.toString()
17. System.out.println(s2);//compiler writes here s2.toString()
18. }
19. }

Output:Student@1fee6fc

Student@1eed786

|  |  |
| --- | --- |
| As you can see in the above example, printing s1 and s2 prints the hashcode values of the objects but I want to print the values of these objects. Since java compiler internally calls toString() method, overriding this method will return the specified values. Let's understand it with the example given below: |  |

## Example of Java toString() method

Now let's see the real example of toString() method.

1. **class** Student{
2. **int** rollno;
3. String name;
4. String city;
6. Student(**int** rollno, String name, String city){
7. **this**.rollno=rollno;
8. **this**.name=name;
9. **this**.city=city;
10. }
12. **public** String toString(){//overriding the toString() method
13. **return** rollno+" "+name+" "+city;
14. }
15. **public** **static** **void** main(String args[]){
16. Student s1=**new** Student(101,"Raj","lucknow");
17. Student s2=**new** Student(102,"Vijay","ghaziabad");
19. System.out.println(s1);//compiler writes here s1.toString()
20. System.out.println(s2);//compiler writes here s2.toString()
21. }
22. }

Output:101 Raj lucknow

102 Vijay Ghaziabad

# Java String charAt()

The **java string charAt()** method returns a char value at the given index number.

The index number starts from 0 and goes to n-1, where n is length of the string. It returns **StringIndexOutOfBoundsException** if given index number is greater than or equal to this string length or a negative number.

### **Internal implementation**

1. public char charAt(int index) {
2. if ((index < 0) || (index >= value.length)) {
3. throw new StringIndexOutOfBoundsException(index);
4. }
5. return value[index];
6. }

### **Signature**

The signature of string charAt() method is given below:

1. public char charAt(int index)

### **Parameter**

**index** : index number, starts with 0

### **Returns**

**A char value**

### **Specified by**

**CharSequence** interface, located inside java.lang package.

### **Throws**

**StringIndexOutOfBoundsException** : if index is negative value or greater than this string length.

## Java String charAt() method example

1. public class CharAtExample{
2. public static void main(String args[]){
3. String name="javatpoint";
4. char ch=name.charAt(4);//returns the char value at the 4th index
5. System.out.println(ch);
6. }}

Output:

t

## StringIndexOutOfBoundsException with charAt()

Let's see the example of charAt() method where we are passing greater index value. In such case, it throws StringIndexOutOfBoundsException at run time.

1. public class CharAtExample{
2. public static void main(String args[]){
3. String name="javatpoint";
4. char ch=name.charAt(10);//returns the char value at the 10th index
5. System.out.println(ch);
6. }}

Output:

Exception in thread "main" java.lang.StringIndexOutOfBoundsException:

String index out of range: 10

at java.lang.String.charAt(String.java:658)

at CharAtExample.main(CharAtExample.java:4)

## Java String charAt() Example 3

Let's see a simple example where we are accessing first and last character from the provided string.

1. public class CharAtExample3 {
2. public static void main(String[] args) {
3. String str = "Welcome to Javatpoint portal";
4. int strLength = str.length();
5. // Fetching first character
6. System.out.println("Character at 0 index is: "+ str.charAt(0));
7. // The last Character is present at the string length-1 index
8. System.out.println("Character at last index is: "+ str.charAt(strLength-1));
9. }
10. }

Output:

Character at 0 index is: W

Character at last index is: l

## Java String charAt() Example 4

Let's see an example where we are accessing all the elements present at odd index.

1. public class CharAtExample4 {
2. public static void main(String[] args) {
3. String str = "Welcome to Javatpoint portal";
4. for (int i=0; i<=str.length()-1; i++) {
5. if(i%2!=0) {
6. System.out.println("Char at "+i+" place "+str.charAt(i));
7. }
8. }
9. }
10. }

Output:

Char at 1 place e

Char at 3 place c

Char at 5 place m

Char at 7 place

Char at 9 place o

Char at 11 place J

Char at 13 place v

Char at 15 place t

Char at 17 place o

Char at 19 place n

Char at 21 place

Char at 23 place o

Char at 25 place t

Char at 27 place l

## Java String charAt() Example 5

Let's see an example where we are counting frequency of a character in the string.

1. public class CharAtExample5 {
2. public static void main(String[] args) {
3. String str = "Welcome to Javatpoint portal";
4. int count = 0;
5. for (int i=0; i<=str.length()-1; i++) {
6. if(str.charAt(i) == 't') {
7. count++;
8. }
9. }
10. System.out.println("Frequency of t is: "+count);
11. }
12. }

Frequency of t is: 4

# Java String compareTo()

The **java string compareTo()** method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

It compares strings on the basis of Unicode value of each character in the strings.

If first string is lexicographically greater than second string, it returns positive number (difference of character value). If first string is less than second string lexicographically, it returns negative number and if first string is lexicographically equal to second string, it returns 0.

1. **if** s1 > s2, it returns positive number
2. **if** s1 < s2, it returns negative number
3. **if** s1 == s2, it returns 0

### **Internal implementation**

1. **public** **boolean** equals(Object anObject) {
2. **if** (**this** == anObject) {
3. **return** **true**;
4. }
5. **if** (anObject **instanceof** String) {
6. String anotherString = (String) anObject;
7. **int** n = value.length;
8. **if** (n == anotherString.value.length) {
9. **char** v1[] = value;
10. **char** v2[] = anotherString.value;
11. **int** i = 0;
12. **while** (n-- != 0) {
13. **if** (v1[i] != v2[i])
14. **return** **false**;
15. i++;
16. }
17. **return** **true**;
18. }
19. }
20. **return** **false**;
21. }

### **Signature**

1. **public** **int** compareTo(String anotherString)

### **Parameters**

**anotherString**: represents string that is to be compared with current string

### **Returns**

an integer value

## Java String compareTo() method example

1. **public** **class** CompareToExample{
2. **public** **static** **void** main(String args[]){
3. String s1="hello";
4. String s2="hello";
5. String s3="meklo";
6. String s4="hemlo";
7. String s5="flag";
8. System.out.println(s1.compareTo(s2));//0 because both are equal
9. System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"
10. System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"
11. System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"
12. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=CompareToExample)

Output:

0

-5

-1

2

## Java String compareTo(): empty string

If you compare string with blank or empty string, it returns length of the string. If second string is empty, result would be positive. If first string is empty, result would be negative.

1. **public** **class** CompareToExample2{
2. **public** **static** **void** main(String args[]){
3. String s1="hello";
4. String s2="";
5. String s3="me";
6. System.out.println(s1.compareTo(s2));
7. System.out.println(s2.compareTo(s3));
8. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=CompareToExample2)

Output:

5

-2

# Java String concat

The **java string concat()** method combines specified string at the end of this string. It returns combined string. It is like appending another string.

### **Internal implementation**

1. **public** String concat(String str) {
2. **int** otherLen = str.length();
3. **if** (otherLen == 0) {
4. **return** **this**;
5. }
6. **int** len = value.length;
7. **char** buf[] = Arrays.copyOf(value, len + otherLen);
8. str.getChars(buf, len);
9. **return** **new** String(buf, **true**);
10. }

### **Signature**

The signature of string concat() method is given below:

1. **public** String concat(String anotherString)

### **Parameter**

**anotherString** : another string i.e. to be combined at the end of this string.

### **Returns**

combined string

## Java String concat() method example

1. **public** **class** ConcatExample{
2. **public** **static** **void** main(String args[]){
3. String s1="java string";
4. s1.concat("is immutable");
5. System.out.println(s1);
6. s1=s1.concat(" is immutable so assign it explicitly");
7. System.out.println(s1);
8. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=ConcatExample)

java string

java string is immutable so assign it explicitly

## Java String concat() Method Example 2

Let's see an example where we are concatenating multiple string objects.

1. **public** **class** ConcatExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str1 = "Hello";
4. String str2 = "Javatpoint";
5. String str3 = "Reader";
6. // Concatenating one string
7. String str4 = str1.concat(str2);
8. System.out.println(str4);
9. // Concatenating multiple strings
10. String str5 = str1.concat(str2).concat(str3);
11. System.out.println(str5);
12. }
13. }

Output:

HelloJavatpoint

HelloJavatpointReader

## Java String concat() Method Example 3

Let's see an example where we are concatenating spaces and special chars to the string object.

1. **public** **class** ConcatExample3 {
2. **public** **static** **void** main(String[] args) {
3. String str1 = "Hello";
4. String str2 = "Javatpoint";
5. String str3 = "Reader";
6. // Concatenating Space among strings
7. String str4 = str1.concat(" ").concat(str2).concat(" ").concat(str3);
8. System.out.println(str4);
9. // Concatenating Special Chars
10. String str5 = str1.concat("!!!");
11. System.out.println(str5);
12. String str6 = str1.concat("@").concat(str2);
13. System.out.println(str6);
14. }
15. }

Output:

Hello Javatpoint Reader

Hello!!!

Hello@Javatpoint

# Java String contains()

The **java string contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*.

### **Internal implementation**

1. **public** **boolean** contains(CharSequence s) {
2. **return** indexOf(s.toString()) > -1;
3. }

### **Signature**

The signature of string contains() method is given below:

1. **public** **boolean** contains(CharSequence sequence)

### **Parameter**

**sequence** : specifies the sequence of characters to be searched.

### **Returns**

**true** if sequence of char value exists, otherwise **false**.

### **Throws**

**NullPointerException** : if sequence is null.

## Java String contains() method example

1. **class** ContainsExample{
2. **public** **static** **void** main(String args[]){
3. String name="what do you know about me";
4. System.out.println(name.contains("do you know"));
5. System.out.println(name.contains("about"));
6. System.out.println(name.contains("hello"));
7. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=ContainsExample)

true

true

false

## Java String contains() Method Example 2

The contains() method searches case sensitive char sequence. If the argument is not case sensitive, it returns false. Let's see an example below.

1. **public** **class** ContainsExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str = "Hello Javatpoint readers";
4. **boolean** isContains = str.contains("Javatpoint");
5. System.out.println(isContains);
6. // Case Sensitive
7. System.out.println(str.contains("javatpoint")); // false
8. }
9. }

true

false

## Java String contains() Method Example 3

The contains() method is helpful to find a char-sequence in the string. We can use it in control structure to produce search based result. Let us see an example below.

1. **public** **class** ContainsExample3 {
2. **public** **static** **void** main(String[] args) {
3. String str = "To learn Java visit Javatpoint.com";
4. **if**(str.contains("Javatpoint.com")) {
5. System.out.println("This string contains javatpoint.com");
6. }**else**
7. System.out.println("Result not found");
8. }
9. }

Output:

This string contains javatpoint.com

# Java String startsWith()

The **java string startsWith()** method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false.

### **Internal implementation**

1. **public** **boolean** startsWith(String prefix, **int** toffset) {
2. **char** ta[] = value;
3. **int** to = toffset;
4. **char** pa[] = prefix.value;
5. **int** po = 0;
6. **int** pc = prefix.value.length;
7. // Note: toffset might be near -1>>>1.
8. **if** ((toffset < 0) || (toffset > value.length - pc)) {
9. **return** **false**;
10. }
11. **while** (--pc >= 0) {
12. **if** (ta[to++] != pa[po++]) {
13. **return** **false**;
14. }
15. }
16. **return** **true**;
17. }

### **Signature**

The syntax or signature of startWith() method is given below.

1. **public** **boolean** startsWith(String prefix)
2. **public** **boolean** startsWith(String prefix, **int** offset)

### **Parameter**

**prefix** : Sequence of character

### **Returns**

true or false

## Java String startsWith() method example

1. **public** **class** StartsWithExample{
2. **public** **static** **void** main(String args[]){
3. String s1="java string split method by javatpoint";
4. System.out.println(s1.startsWith("ja"));
5. System.out.println(s1.startsWith("java string"));
6. }}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=StartsWithExample)

Output:

true

true

## Java String startsWith(String prefix, int offset) Method Example

This is overloaded method of startWith() method which is used to pass one extra argument (offset) to the function. This method works from the passed offset. Let's see an example.

1. **public** **class** StartsWithExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str = "Javatpoint";
4. System.out.println(str.startsWith("J")); // True
5. System.out.println(str.startsWith("a")); // False
6. System.out.println(str.startsWith("a",1)); // True
7. }
8. }

Output:

true

false

true

# Java String endsWith()

The **java string endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.

### **Internal implementation**

1. **public** **boolean** endsWith(String suffix) {
2. **return** startsWith(suffix, value.length - suffix.value.length);
3. }

### **Signature**

The syntax or signature of endsWith() method is given below.

1. **public** **boolean** endsWith(String suffix)

### **Parameter**

**suffix** : Sequence of character

### **Returns**

true or false

## Java String endsWith() method example

1. **public** **class** EndsWithExample{
2. **public** **static** **void** main(String args[]){
3. String s1="java by javatpoint";
4. System.out.println(s1.endsWith("t"));
5. System.out.println(s1.endsWith("point"));
6. }}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=EndsWithExample)

Output:

true

true

## Java String endsWith() Method Example 2

1. **public** **class** EndsWithExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str = "Welcome to Javatpoint.com";
4. System.out.println(str.endsWith("point"));
5. **if**(str.endsWith(".com")) {
6. System.out.println("String ends with .com");
7. }**else** System.out.println("It does not end with .com");
8. }
9. }

Output:

false

String ends with .com

# Java String split()

The **java string split()** method splits this string against given regular expression and returns a char array.

### **Internal implementation**

1. public String[] split(String regex, int limit) {
2. /\* fastpath if the regex is a
3. (1)one-char String and this character is not one of the
4. RegEx's meta characters ".$|()[{^?\*+\\", or
5. (2)two-char String and the first char is the backslash and
6. the second is not the ascii digit or ascii letter.
7. \*/
8. char ch = 0;
9. if (((regex.value.length == 1 &&
10. ".$|()[{^?\*+\\".indexOf(ch = regex.charAt(0)) == -1) ||
11. (regex.length() == 2 &&
12. regex.charAt(0) == '\\' &&
13. (((ch = regex.charAt(1))-'0')|('9'-ch)) < 0 &&
14. ((ch-'a')|('z'-ch)) < 0 &&
15. ((ch-'A')|('Z'-ch)) < 0)) &&
16. (ch < Character.MIN\_HIGH\_SURROGATE ||
17. ch > Character.MAX\_LOW\_SURROGATE))
18. {
19. int off = 0;
20. int next = 0;
21. boolean limited = limit > 0;
22. ArrayList<String> list = new ArrayList<>();
23. while ((next = indexOf(ch, off)) != -1) {
24. if (!limited || list.size() < limit - 1) {
25. list.add(substring(off, next));
26. off = next + 1;
27. } else {    // last one
28. //assert (list.size() == limit - 1);
29. list.add(substring(off, value.length));
30. off = value.length;
31. break;
32. }
33. }
34. // If no match was found, return this
35. if (off == 0)
36. return new String[]{this};
38. // Add remaining segment
39. if (!limited || list.size() < limit)
40. list.add(substring(off, value.length));
42. // Construct result
43. int resultSize = list.size();
44. if (limit == 0)
45. while (resultSize > 0 && list.get(resultSize - 1).length() == 0)
46. resultSize--;
47. String[] result = new String[resultSize];
48. return list.subList(0, resultSize).toArray(result);
49. }
50. return Pattern.compile(regex).split(this, limit);
51. }



### **Signature**

There are two signature for split() method in java string.

1. public String split(String regex)
2. and,
3. public String split(String regex, int limit)



### **Parameter**

**regex** : regular expression to be applied on string.

**limit** : limit for the number of strings in array. If it is zero, it will returns all the strings matching regex.

### **Returns**

array of strings

### **Throws**

**PatternSyntaxException** if pattern for regular expression is invalid

### **Since**

1.4

## Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

1. public class SplitExample{
2. public static void main(String args[]){
3. String s1="java string split method by javatpoint";
4. String[] words=s1.split("\\s");//splits the string based on whitespace
5. //using java foreach loop to print elements of string array
6. for(String w:words){
7. System.out.println(w);
8. }
9. }}



[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=SplitExample)

java

string

split

method

by

javatpoint

## Java String split() method with regex and length example

1. public class SplitExample2{
2. public static void main(String args[]){
3. String s1="welcome to split world";
4. System.out.println("returning words:");
5. for(String w:s1.split("\\s",0)){
6. System.out.println(w);
7. }
8. System.out.println("returning words:");
9. for(String w:s1.split("\\s",1)){
10. System.out.println(w);
11. }
12. System.out.println("returning words:");
13. for(String w:s1.split("\\s",2)){
14. System.out.println(w);
15. }
17. }}



[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=SplitExample2)

returning words:

welcome

to

split

world

returning words:

welcome to split world

returning words:

welcome

to split world

## Java String split() method with regex and length example 2

Here, we are passing split limit as a second argument to this function. This limits the number of splitted strings.

1. public class SplitExample3 {
2. public static void main(String[] args) {
3. String str = "Javatpointtt";
4. System.out.println("Returning words:");
5. String[] arr = str.split("t", 0);
6. for (String w : arr) {
7. System.out.println(w);
8. }
9. System.out.println("Split array length: "+arr.length);
10. }
11. }



Returning words:

Java

poin

Split array length: 2