**Strings**, which are widely used in **Java** programming, are a sequence of characters. In the **Java**programming language, **strings** are objects. The **Java**platform provides the **String** class to create and manipulate **strings**.

Manipulate:

handle or control (a tool, mechanism, information, etc.) in a skilful manner

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'};

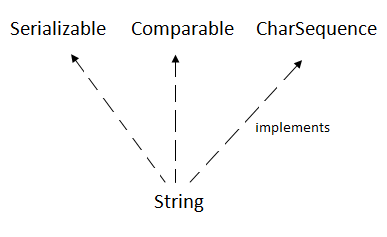
String s=**new** String(ch);

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.



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.

### What is String in java

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.

### How to create String object?

|  |
| --- |
| There are two ways to create String object:   1. By string literal 2. By new keyword |

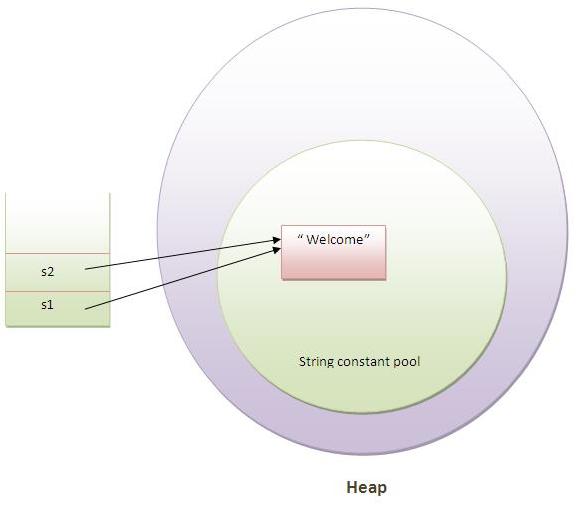
### 1) String Literal

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

1. 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:

1. String s1="Welcome";
2. 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 class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

**1 of 28 methods check in it.**

# Immutable String in Java

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

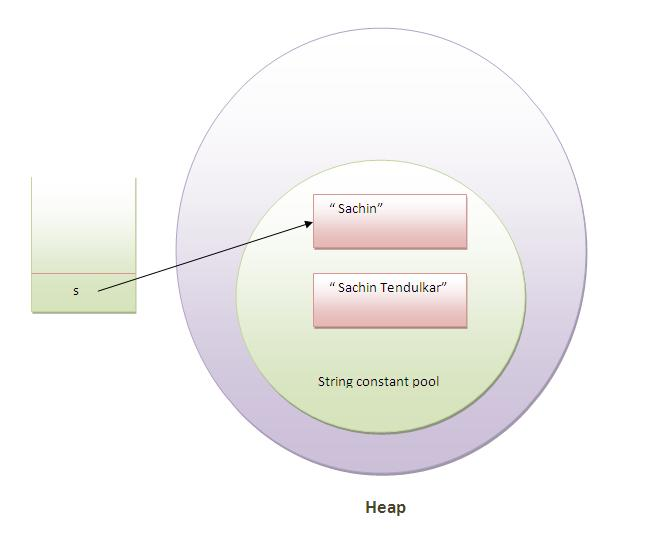
Once string object is created its data or state can't be changed but a new string object is created.

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

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. }

**Output:Sachin**

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. 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:

1. **class** Testimmutablestring1{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s=s.concat(" Tendulkar");
5. System.out.println(s);
6. }
7. }

Output:Sachin Tendulkar

### Why string objects are immutable in java?

|  |
| --- |
| Because java uses the concept of string literal.  Suppose there are 5 reference variables, all refers 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 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. 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. }} |

**O/P:** java string

java string is immutable so assign it explicitly

# The String Constant Pool

**public class StringConstantPool {**

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

**String s = "prasad";**

**String s2 = "prasad";**

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

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

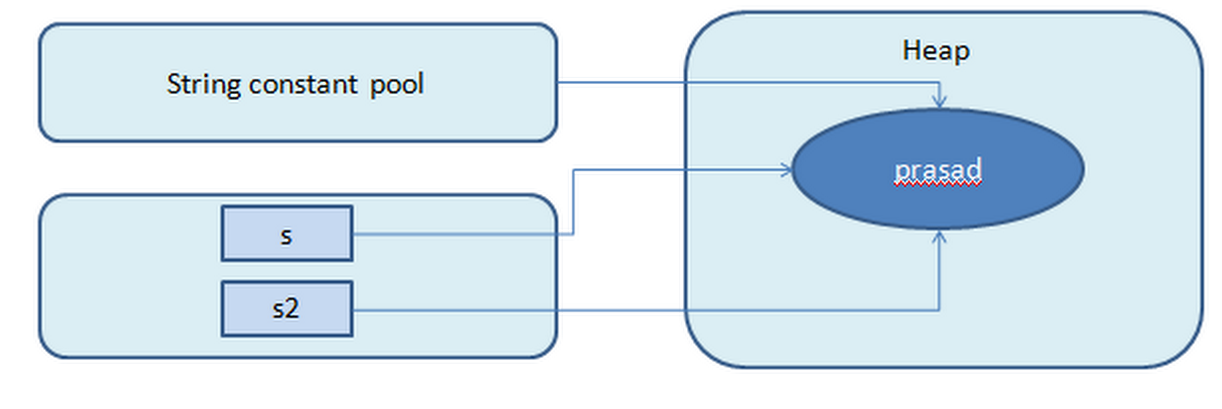
**}**

**}**

Now lets know what happens here step by step

* The class is loaded when JVM is invoked.
* JVM will look for all the string literals in the program
* First, it finds the variable s which refers to the  literal “prasad” and it will be created in the memory
* A reference for the literal “prasad” will be placed in the string constant pool memory.
* Then it finds another variable s2 which is referring to the same string literal “prasad“.
* Now that JVM has already found a string literal “prasad“, both the variables s and s2 wil refer to the same object i.e. “prasad“.

The diagram below demonstrates this

****

String literals referred

Now we have looked into the case when string literals are created without using the new operator. What happens if String s2 = new ("prasad");

As we are invoking the new keyword, **The object “prasad” will be created when the new String(“prasad”) is invoked. This is unlike the string literal “prasad” which is created when class is loaded.**

Now the values of objects referenced by variable s and variable s2 are the same i.e. “prasad” but those are not the same objects. They refer to different objects. We will verify this with a program.

public class StringConstantPool {

public static void main(String[] args) {

String s = "prasad";

String s2 = new String("prasad");

System.out.println(s.equals(s2));

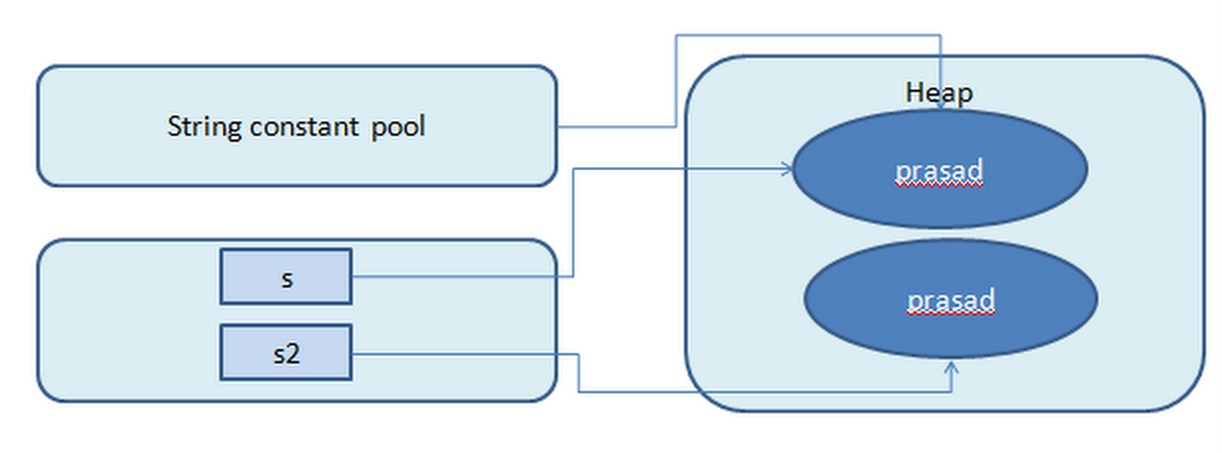
System.out.println(s == s2);

}

}

* The contents of both objects are the same so equals method returns true
* The objects referred by both variables are different so == operator returns false

This is elaborated in this diagram



String with new keyword

Points to be remembered

* String literals with same values will always refer to the same String object
* String objects created using new operator will be different from literals

# How to Create immutable Class in java?

Today we will learn how to create immutable class in java. **Immutable objects** are instances whose state doesn’t change after it has been initialized. For example, String is an immutable class and once instantiated its value never changes.

Other benefit of immutable class is that it is inherently [**thread-safe**](https://www.journaldev.com/1061/thread-safety-in-java), so you don’t need to worry about thread safety in case of multi-threaded environment.

To create immutable class in java, you have to do following steps.

1. Declare the class as final so it can’t be extended.
2. Make all fields private so that direct access is not allowed.
3. Don’t provide setter methods for variables
4. Make all **mutable fields final** so that it’s value can be assigned only once.
5. Initialize all the fields via a constructor performing deep copy.
6. Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

**Stackoverflow:::**

1. No. It's not the same using StringBuilder than doing "a" + "b".
2. In Java, String instances are immutable.
3. So, if you do:
4. String c = "a" + "b";
5. You are creating new Strings every time you concatenate.
6. On the other hand, StringBuilder is like a buffer that can grow as it needs when appending new Strings.
7. StringBuilder c = new StringBuilder();
8. c.append("a");
9. c.append("b"); // c is only created once and appended "a" and "b".
10. Rule of the thumb is (changed thanks to the comments I got):
11. If you are going to concatenate a lot (i.e., concatenate inside a loop, or generating a big XML formed by several string concatenated variables), do use StringBuilder. Otherwise, simple concatenation (using + operator) will be just fine.
12. Compiler optimizations also play a huge role when compiling this kind of code.
13. [Here's](http://chaoticjava.com/posts/stringbuilder-vs-string/) further explanation on the topic. And more StackOVerflow questions on the issue:
14. [One](https://stackoverflow.com/questions/1532461/stringbuilder-vs-string-concatenation-in-tostring-in-java), [Two](https://stackoverflow.com/questions/63150/whats-the-best-way-to-build-a-string-of-delimited-items-in-java), [Three](https://stackoverflow.com/questions/242438/java-performance-of-stringbuilder-in-a-loop).

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.

### 1) StringBuilder append() method

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

### 2) StringBuilder insert() method

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

### 3) StringBuilder replace() method

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

### 4) StringBuilder delete() method

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

### 5) StringBuilder reverse() method

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

### 6) StringBuilder capacity() method

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder 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.

### 7) StringBuilder ensureCapacity() method

The ensureCapacity() method of StringBuilder 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.

# 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.

### 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.

### 2) StringBuffer insert() method

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

### 3) StringBuffer replace() method

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

### 4) StringBuffer delete() method

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

### 5) StringBuffer reverse() method

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

### 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.

### 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.