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:

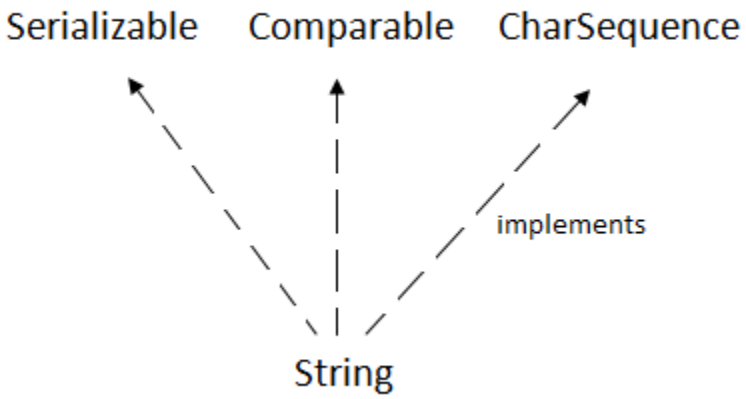
```
1. char[] ch={'j','a','v','a'};
2. String s=new String(ch);
```

is same as:

```
1. String s="java";
```

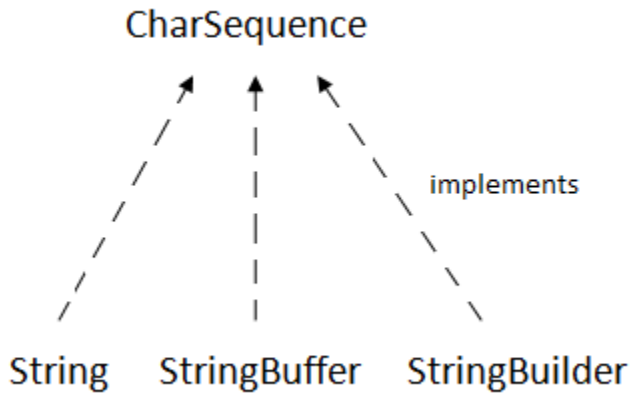
Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

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 String in Java is and how to create the String object.

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 a string object.

How to create a 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 the

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";//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).

By new keyword

1. 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).

Interfaces and Classes in Strings in Java

In Java, both **String** and **CharBuffer** interact with sequences of characters, but they are designed with different use cases and underlying mechanisms in mind. Here's an exploration of their interfaces, classes, and how they fit into the Java ecosystem.

String

The **String** class is one of the most fundamental types in Java, designed to represent immutable sequences of characters. Here's a closer look at its characteristics and the interfaces it implements:

- **Immutability:** Once instantiated, a **String** object cannot be modified. This immutable design is a deliberate choice to ensure thread safety, consistency, and efficiency, especially regarding the String pool mechanism.
- **String Pool:** Java maintains a pool of string literals to help save memory. When a new string literal is created, Java checks the Pool for a matching string. If found, the new variable references the pooled string. If not, the new string is added to the Pool.
- **Implemented Interfaces:** The **String** class implements several interfaces, including:
    - **Serializable:** Allows string objects to be serialized into byte streams, facilitating their transmission or storage.
    - **Comparable<String>:** Enables lexical comparison between two strings, supporting natural ordering within collections.
    - **CharSequence:** Provides a unified read-only interface for different kinds of char sequences, allowing **String** objects to be manipulated and accessed generically.

CharBuffer

CharBuffer, on the other hand, is part of the **java.nio** package, which provides a set of classes for non-blocking I/O operations. **CharBuffer** is a mutable sequence of characters with more flexibility for manipulation. Here's more about **CharBuffer:**