String Handling:

In the java.lang package we have below three classes to store and perform different operations on a string of characters.

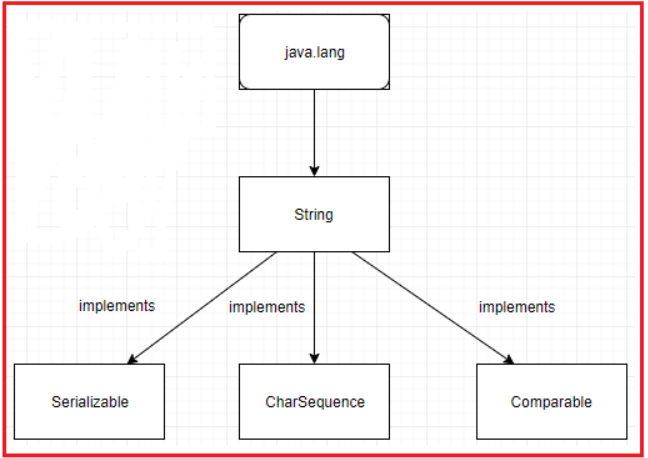
1. **String**
2. **StringBuffer**
3. **StringBuilder**

**String in Java:**

The string is a predefined class available in **java.lang.package** which is used to store a group of characters. Strings are immutable. Whenever a change to a String is made, an entirely new String is created. If we want to store a group of characters we can use a char array.

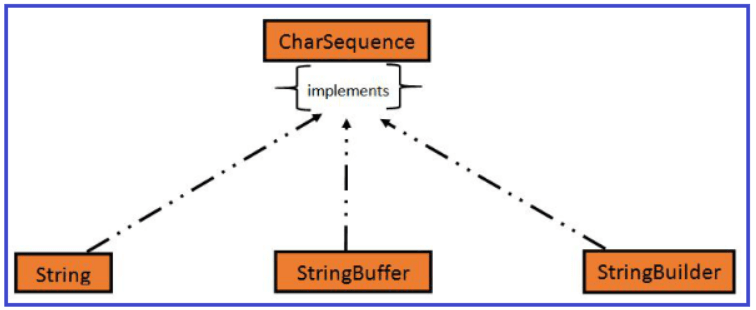
For example: **char name[] = new char[10];**

But here size is fixed, with no predefined methods. If we want to use the String class we have to create an object for the string class. The **java.lang.String** class implements a Serializable, Comparable, and CharSequence interface. For better understanding please have a look at the following image.



**CharSequence interface in Java**

A CharSequence is a readable sequence of char values. This interface provides uniform, read-only access to many different kinds of char sequences. This interface does not refine the general contracts of the equals and hashCode methods. The result of comparing two objects that implement CharSequence is, therefore, in general, undefined. Each object may be implemented by a different class, and there is no guarantee that each class will be capable of testing its instances for equality with those of the other. It is therefore inappropriate to use arbitrary CharSequence instances as elements in a set or as keys in a map. String, StringBuffer, and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.



**How to create an object for String in Java?**

To create an object for String class we have the following two ways:

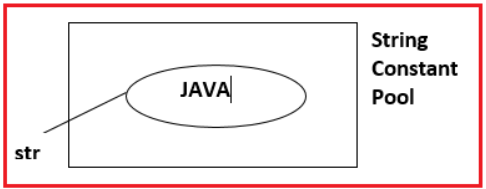
1. **Using string literal directly**
2. **Using a new keyword**

**Using String Literal to create a String object in Java**

In java, Strings can be created by assigning a string literal to a String instance:

**Example:** **String str =”Java”;**

When we create String objects using a string literal directly then the string object is created inside a special memory area called String Constant Pool.



**The disadvantage of this approach:**

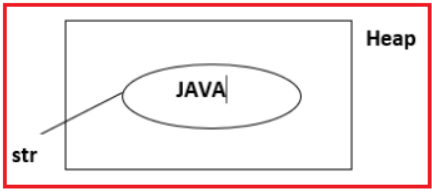
As the string is an object in Java. However, if we have not created any string object sing new keyword then the compiler will automatically create a string object having the string literal and will assign it to the provided string instances.

But if the object already exists in the memory it does not create a new Object rather it assigns the same old object to the new instance, that means even though we have two string instances, the compiler will create only one string object having the same value and will assign the same to both the instances.

**Using new Operator to create string object in Java**

As we saw above that when we tried to assign the same string object to two different literals, the compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings by using the new keyword.

**Example**: **String str = new String{“Java”);**When we create a String object using a new operator then a string object is created inside Heap memory.



**Simple Java String Program:**

**package** *Demo*;

**public** **class** StringDemo

**{**

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

**{**

//creating a string by java string literal

**String** str = "Hello World";

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

//converting char array arrch[] to string str2

**String** str2 = new **String** **(**arrch**)**;

//creating another java string str3 by using new keyword

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

//Displaying all the three strings

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

System.out.println **(**str2**)**;

System.out.println **(**str3**)**;

**}**

**}**

**Output:**

**Hello World**  
**hello**  
**Java String Example**

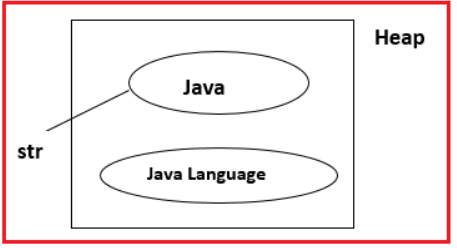
**What is Java String Pool?**

Java String pool refers to a collection of Strings that are stored in heap memory. In this, whenever a new object is created, the String pool first checks whether the object is already present in the pool or not. If it is present, then the same reference is returned to the variable else a new object will be created in the String pool and the respective reference will be returned.

**Immutable Objects in Java**

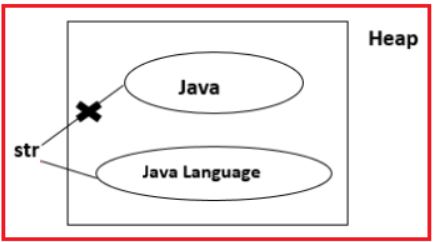
String Objects can be created either by using a new operator or by using string Literal directly. In both, cases String objects are created as Immutable Objects.

**Example:**  
**String str = new Strinh(“Java”);**  
**System.out.println(str); //Java**  
**System.out.println(str.concat(“Language”)); // Java Language**  
**System.out.println(str); //Java**



Objects are created immutable means once the string is created its content never be changed instead of this it will create the new string object with the changed content. But string reference variables can replace their objects with new objects.

**Example:**  
**String str = new String(“Java”);**  
**System.out.println(str); //Java**  
**str = str.concat(“Language”);**  
**System.out.println(str); //Java Language**



**Difference Between creating String Object using New Operator and String Literal directly**

**Using new operator**

When we create a String Object using a new operator then the string Object is created inside the Heap Memory. While storing string Object into Heap it never checks whether the newly created object has already existed or not simply it will create a new object every time.

**Example:**  
**String str1 = new String(“Hi”);**  
**str1 = new String(“welcome”);**  
**String str2 = new String(“friends”);**  
**String str3 = new String(“java world”);**  
**String str4 = new String(“friends”);**  
**String str5 = str3;**

Here, Hi Object any reference which is called an unused object or an unreferenced object that is ready for garbage collection. When there is any unused object available in the heap then it is the responsibility of the garbage collector program to delete those unused objects.

**Using string Literal directly**

When we create a String Object using String Literal directly then a string Object is created inside string Constant Pool. While storing string objects into String constant Pool it always checks whether the newly created object has already existed or not. If the object is already existed then reused the same object otherwise if not available it will create new objects so that memory can be utilized efficiently.

**Example:**  
**String str1 = new String(“Hi”);**  
**str1 = new String(“welcome”);**  
**String str2 = new String(“friends”);**  
**String str3 = new String(“java world”);**  
**String str4 = new String(“friends”);**  
**String str5 = str3;**

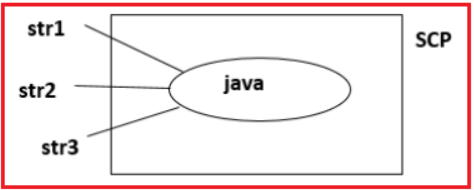
Here, the “Hi” Object does not have any reference which is called an unused object or an unreferenced object. When there is any unused object is available in String constant pool then it is the responsibility of SCP itself to delete those unused objects.

**Note:** In Java, a reference variable can refer to only one object at any time. But one object can be referred to by any number of reference variables.

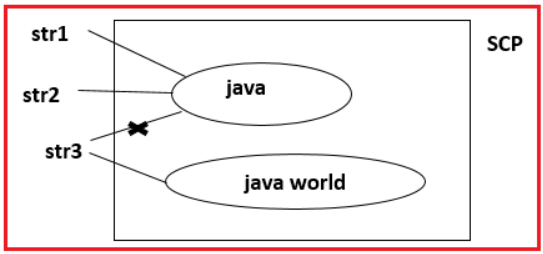
**Why String Objects are given as Immutable Objects in Java?**

Sometimes an object can be referred to by multiple reference variables in this case if string objects are mutable objects then if we change the content of the object then automatically other references get also modified so that string Objects are given as Immutable Objects, it means whenever any operation is done on Strings it will create a new object.

**Example:**  
**String str1 = “java”;**  
**String str2 = “java”;**  
**String str3 = “java”;**

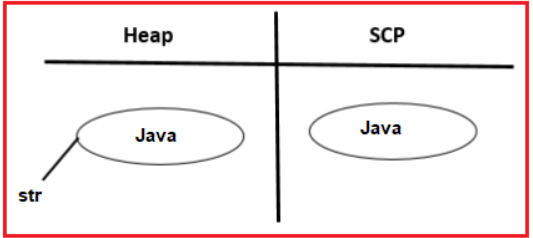


**str3 = str3+” world”;**

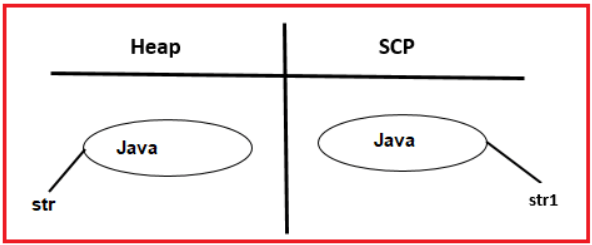


When we create a String Object using the new operator then a String object is created inside the Heap memory and the same copy of the object is also created in SCP for re-usability.

**Example: String str = new String(“Java”);**



**String str1 = “Java”;**



**How to create an Immutable class in Java?**

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 an immutable class by creating a final class that has final data members.

**Example to create Immutable Class in Java:**

**public** **final** **class** Employee

**{**

**final** **String** pancardNumber;

**public** Employee **(String** pancardNumber**)**

**{**

this.pancardNumber = pancardNumber;

**}**

**public** **String** getPancardNumber **()**

**{**

**return** pancardNumber;

**}**

**}**

The above class is immutable because:

1. The instance variable of the class is final i.e. we cannot change its value of it after creating an object.
2. The class is final so we cannot create the subclass.
3. There are no setter methods i.e. we have no option to change the value of the instance variable.

These points make this class immutable.

**String Comparison in Java**

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

1. **By equals() method:** It is used in the authentication
2. **By = = operator:** It is used for reference matching
3. **By compareTo() method:** It is used for sorting

**equals() Method:**

equals() method always check the equality of the content of two given strings. It is used for Content Comparison. The String equals() method compares the original content of the string. It compares the values of string for equality.

**== Operator:**

The = = operator compares references not values. It is always used to compare the address of two references whether they are referring to the same object or not.

**compareTo() Method:**

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

Suppose s1 and s2 are two string variables. If:  
s1 == s2 : 0  
s1 > s2   : positive value  
s1 < s2   : negative value

**Sample Program for String Comparison in Java**

**package** *Demo*;

**public** **class** StringComparison **{**

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

**String** s1="Java";

**String** s2="Java";

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

**String** s4="JavaLanguage";

**String** s5="JAVA";

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

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

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

System.out.println**(**s1.equals**(**s5**))**;//false

System.out.println**(**s1.equalsIgnoreCase**(**s5**))**;//true

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:**

**Rules for String Comparison:**

We never use comparison operators like <,>,<=,>= to compare two Strings directly. But if we want to achieve this requirement we have to use the compareTo() method of the String class. But we can use comparison operators or equality operators like ==, != to compare two strings.

**Example :**  
**String str1 = “java”;**  
**String str2 = “java”;**

**String Concatenation in Java**

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

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

**+ Operator:**

If we use + Operator between numerical data type values then it will perform an arithmetical addition operation. But if we use + operator between two strings or between strings or any primitive then it will do Concatenation Operation.

**concat() Method:**

The String Concat() method concatenates the specified string to the end of the current string.

**Example: Program for String Concatenation in Java**

**package** *Demo*;

**public** **class** StringConcatenation **{**

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

**String** s1="Hello ";

**String** s2="World";

**String** s3=s1.concat**(**s2**)**;

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

**String** str = new **String(**"Hello "**)**;

System.out.println**(**str+123**)**;

**}**

**}**

**Output:**  
**Hello World**  
**Hello 123**

**String Formatting in Java**

Java String format() method is used for formatting the String. The java string format() method returns a formatted string using the given locale, specified format string, and arguments. We can concatenate the strings using this method and at the same time, we can format the output concatenated string.

**Syntax: public static String format(Locale loc, String form, Object… args)**

**Example: Program to create a formatted string in Java**

**package** *Demo*;

**public** **class** StringFormat

**{**

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

**{**

**String** str = "Java is an Object-Oriented Programming Language";

//concatenating string using format

**String** formattedString = **String**.format **(**"My String is %s", str**)**;

/\*formatting the value passed and concatenating at the same time

\* %.6f is for having 6 digits in the fractional part

\*/

**String** formattedString2 = **String**.format **(**"My String is %.6f", 12.121**)**;

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

System.out.println **(**formattedString2**)**;

**}**

**}**

**Output:**

**My String is Java is an Object-Oriented Programming Language**  
**My String is 12.121000**

**String Methods in Java**

Following is the list of methods supported by the String class –

**int length():**

This method returns the number of chars available in the given String.

**Example :**  
**String str = ne wString(“Java Language”);**  
**System.out.println(str); //Java Language**  
**System.out.println(str.length()); //13**

**char charAt(int index):**

This method returns the character at the specified index position available in the given string. This method is especially used to access the string char by char.

**Example : Accessing the string char by char**  
**String str = new String(“Java Language”);**  
**for(int i=0; i<str.length(); i++);{**  
**char ch = str.charat(i);**  
**System.out.println(ch);**  
**}**

**String concat(String) :**

This method is used to append the given two Strings.

**Example :**  
**String str = new String(“Java”);**  
**System.out.println(str); //Java**  
**System.out.println(str.concat(” Language”)); //Java Language**  
**System.out.println(str); //Java**

**String toLowerCase() :**

This method returns String in lower case or Small Letters.  
**Example :**  
**String str = new String(“Java Language”);**  
**System.out.println(str); //Java Language**  
**System.out.println(str.toLowerCase()); //java language**

**String toUpperCase() :**

**Example :**  
**String str = new String(“Java Language”);**  
**System.out.println(str); //Java Language**  
**System.out.println(str.toUpperCase()); //JAVA LANGUAGE**

**boolean equals(String) :**

This method is used to compare the equality of the two strings and it returns true if two strings are equal otherwise if not equal it returns false. This method considers the case of the Strings.  
**Example :**  
**String str = new string(“java”);**  
**System.out.println(str.equals(“java”)); //true**  
**System.out.println(str.equals(“Java”)); //false**

**boolean equalsIgnoreCase(String) :**

This method is used to compare the equality of the two strings same as the equals() method but it will not consider the case of the strings.  
**Example :**  
**String str = new string(“java”);**  
**System.out.println(str.equals(“java”)); //true**  
**System.out.println(str.equalsIgnoreCase(“Java”)); //true**  
**System.out.println(str.equalsIgnoreCase(“Hello”)); //false**

**boolean startsWith(String) :**

This method returns true with a specified string otherwise it returns false (it always considers the case).  
**Example :**  
**String str = “Java Language”;**  
**System.out.println(str.startsWith(“Java”)); //true**  
**System.out.println(str.startsWith(“Javax”)); //false**

**boolean endsWith(String) :**

This method returns true when the given string ends with a specified string otherwise it returns false (it always considers the case).

**Example :**  
**String str = “Java Language”;**  
**System.out.println(str.endsWith(“age”)); //true**  
**System.out.println(str.endsWith(“agex”)); //false**

**int indexOf(char) or int indexOf(String) :**

This method returns the index of the first occurrence of the specified character. It always begins searching from index 0.  
**Example :**  
**String str = “Java Language”;**  
**System.out.println(str.indexOf(‘a’)); //1**  
**System.out.println(str.indexOf(‘p’)); //-1**

**int indexOf(char, int index) :**

This method returns the index of the specified character. It will begin searching the character from the specified index.  
**Example :**  
**String str = “Java Language”;**  
**System.out.println(str.indexOf(‘a’, str.indexOf(‘a’)+1)); //3**

**int lastIndexOf(char) or int lastIndexOf(String) :**

This method returns index of the last occurrence of specified character.  
**Example :**  
**String str = “Java Language”;**  
**System.out.println(str.lastIndexOf(‘a’)); //10**  
**System.out.println(str.lastIndexOf(‘q’)); //-1**  
**System.out.println(str.lastIndexOf(“Lan”)); //5**  
**System.out.println(str.lastIndexOf(“Lanx”)); //-1**

**Note:** All the above three methods are case-sensitive. If the specified String or character is not available all these three methods return -1.

**String substring(int index) :**

This method is used to capture the part of the string from the given string. It will capture from the specified index to the end of the string.  
**Example :**  
**String str = “Java language is Good”;**  
**System.out.println(str.substring(5)); //Language is Good**

**String substring(int index, int offset) :**

This method is also used to capture the part of the string from the given string. It will capture from the specified index to the specified offset. The index begins from 0 to size -1 and Offset will begin from 1 to size  
**Example :**  
**String str = “Java language is Good”;**  
**System.out.println(str.substring(0,4)); //Java**  
**System.out.println(str.substring(5,8)); //Lan**

**String replace(“oldstring”, “newstring”) :**

This method replaces the specified chars of the string with a newly specified string of characters if available otherwise it won’t replace anything and returns the same string.  
**Example :**  
**String str = “Java Language”;**  
**System.out.println(str.replace(“Java”, “VB”); //VB Language**

**String trim() :**

This method removes extra spaces given before the text and after the text.  
**Example :**  
**String str = ” Java Language “;**  
**System.out.println(“HI”+str+”OK”);**  
**System.out.println(“HI”+str.trim()+”OK”);**

**String intern() :**

This method is used to return the corresponding object of the string reference which is available in SCP.  
**Example :**  
**class stringTest{**  
**public static void main(String[] data){**  
**String str1 = new String(“Java”);**  
**String str2 = str1.intern();**  
**System.out.println(str1); //java**  
**System.out.println(str2); //java**  
**System.out.println(str1==str2); //false**  
**}**  
**}**

**Method Chaining in Java**

Method Chaining is a process of calling string methods continuously using the dot(.) operator. Method Chaining is the practice of calling different [methods](https://www.geeksforgeeks.org/methods-in-java/) in a single line instead of calling different methods with the same object reference separately. Under this procedure, we have to write the object reference once and then call the methods by separating them with a (dot.)

**Example: Program to demonstrate the working for Method Chaining in Java**

**package** *Demo*;

**class** Aa

**{**

**private** **int** a;

**private** **float** b;

Aa **()**

**{**

System.out.println **(**"Calling The Constructor"**)**;

**}**

**public** Aa setint **(int** a**)**

**{**

this.a = a;

**return** this;

**}**

**public** Aa setfloat **(float** b**)**

**{**

this.b = b;

**return** this;

**}**

**void** display **()**

**{**

System.out.println **(**"Display= " + a + " " + b**)**;

**}**

**}**

**public** **class** MethodChaining

**{**

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

**{**

// This is the "method chaining".

new Aa **()**.setint **(**10**)**.setfloat **(**20**)**.display **()**;

**}**

**}**

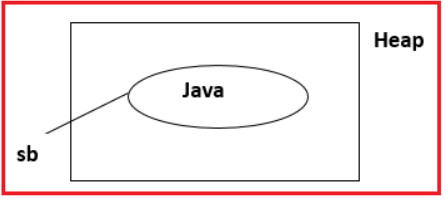
**Output:**

**Calling The Constructor**  
**Display= 10 20.0**

**StringBuffer in Java with Examples:**

StringBuffer is another predefined class available in java.lang package which is also used to store a group of characters like the String class. But String class objects are immutable objects whereas StringBuffer Objects are called mutable objects. StringBuffer objects are mutable objects means the content of StringBuffer can be changed any number of times. We have only one way to create a StringBuffer object that is using the new operator.

**Example:**  
**StringBuffer sb = new StringBuffer(“java”);**  
**System.out.println(sb); //java**



**Methods of StringBuffer in Java**

**int length() :**

This method returns the count of the number of chars available in the StringBuffer.

**StringBuffer append(XXXX) :**

This method is used to append the specified content with the existing content.  
**Example :**  
**StringBuffer sb = new StringBuffer(“java”);**  
**System.out.println(sb); //java**  
**System.out.println(sb.append(“Language”)); //java Language**

**StringBuffer insert(int index,xxxx) :**

This method inserts the specified content at the specified position in the String Buffer.  
**Example :**  
**StringBuffer sb = new StringBuffer(“java Language”);**  
**System.out.println(sb); //java Language**  
**System.out.println(sb.insert(4,”Is Good”)); //java is Good Language**

**StringBuffer deleteCharAt(int index):**

This method deletes the character at the specified index and returns the StringBuffer.  
**Example :**  
**StringBuffer sb = new StringBuffer(“java Language”);**  
**System.out.println(sb); //java Language**  
**System.out.println(sb.deleteCharAt(1)); //jva Language**

**StringBuffer delete(int index, int offset):**

This method deletes the character at the specified index to the specified offset and returns StringBuffer.  
**Example :**  
**StringBuffer sb = new StringBuffer(“java Language”);**  
**System.out.println(sb); //java Language**  
**System.out.println(sb.delete(4,9)); //jva age**

**String substring(int index) :**

This method captures the substring from the specified index to the end of the string in the given StringBuffer.  
**Example :**  
**StringBuffer sb = new StringBuffer(“java Language”);**  
**System.out.println(sb); //java Language**  
**System.out.println(sb.substring(5)); //Language**

**String substring(int index, int offset) :**

This method captures the substring from the specified index to the specified offset of the string in the given StringBuffer.  
**Example :**  
**StringBuffer sb = new StringBuffer(“java Language”);**  
**System.out.println(sb); //java Language**  
**System.out.println(sb.substring(5,8)); //Lan**

**StringBuffer replace(int index, int offset, String) :**

This method replaces the String in the StringBuffer at specified positions.  
**Example :**  
**StringBuffer sb = new StringBuffer(“java Language”);**  
**System.out.println(sb); //java Language**  
**System.out.println(sb.replace(0,4,”VB”)); //VB Language**  
**System.out.println(sb); //VB Language**

**StringBuffer reverse() :**

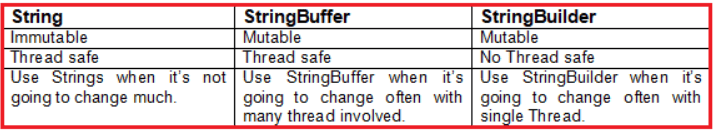
This method reverses the content of the given StringBuffer.  
**Example :**  
**StringBuffer sb = new StringBuffer(“java Language”);**  
**System.out.println(sb); //java Language**  
**System.out.println(sb.reverse()); //egaugnaL avaj**  
**System.out.println(sb); //egaugnaL avaj**

**String Builder in Java:**

StringBuilder is another predefined class that is also used to store a group of chars. StringBuilder is exactly similar to StringBuffer but the StringBuilder class is not a synchronized class whereas the StringBuffer class is a synchronized class. The StringBuilder in Java represents a mutable sequence of characters.

**Syntax : StringBuilder sb = new StringBuilder();**

**Difference Between String, StringBuffer, and StringBuilder**



**String Tokenizer in Java with Examples**

StringTokenizer class in Java is used to break a string into tokens. StringTokenizer is used to split a string by “space” and “comma” delimiter, iterate the StringTokenizer elements and print it out one by one. The set of delimiters (the characters that separate tokens) may be specified either at the creation time or on a per-token basis.

An instance of StringTokenizer behaves in one of two ways, depending on whether it was created with the returnTokens flag having the value true or false:

1. If the flag is false, delimiter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters.
2. If the flag is true, delimiter characters are considered to be tokens. A token is either one delimiter character or a maximal sequence of consecutive characters that are not delimiters.

**Example: Program for StringTokenizer in Java**

**package** *Demo*;

**import** *java.util.StringTokenizer*;

**public** **class** StringTokenizerDemo **{**

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

StringTokenizer st1 = new StringTokenizer**(**"Hi! I am good. How about you?"**)**;

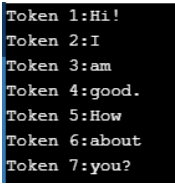
**for** **(int** i = 1; st1.hasMoreTokens**()**; i++**)**

System.out.println**(**"Token "+i+":"+st1.nextToken**())**;

**}**

**}**

**Output:**



**String Joiner in Java with Examples**

Java added a new final class StringJoiner in **java.util** package. The StringJoiner class in Java 8 constructs a sequence of characters. This sequence is separated by a delimiter and optionally starts with a supplied prefix and ends with a supplied suffix. It is used to construct a sequence of characters separated by a delimiter. Now, you can create a string by passing delimiters like comma(,), hyphen(-), etc.

**Example: Program for StringJoiner in Java**

**package** *Demo*;

**import** *java.util.StringJoiner*;

**public** **class** StringJoinerDemo **{**

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

StringJoiner strJoin = new StringJoiner**(**","**)**;

strJoin.add**(**"One"**)**;

strJoin.add**(**"Two"**)**;

strJoin.add**(**"Three"**)**;

strJoin.add**(**"Four"**)**;

strJoin.add**(**"Five"**)**;

strJoin.add**(**"Six"**)**;

strJoin.add**(**"Seven"**)**;

System.out.println**(**strJoin.toString**())**;

**}**

**}**

**Output:**  
**One,Two,Three,Four,Five,Six,Seven**

**Is String a Class or data type?**

The string is a class in java.lang package. But in java, all classes are also considered as data types. So we can also say String is a data type.

**Why string class is given when a char array is available to represent a sequence of characters?**

String class is given to store characters dynamically without size limitations and also to perform different common operations on string data.

**Explanations:**

1. The limitation of the array is size. Once the array is created with some size then its size can’t be modified. So if we use a character array to represent or store a sequence of characters then we can store only characters up to their size.
2. If we want to store new characters beyond their size, we must create an array with the new required size and should copy all old array characters to the new array, and then we have to store new values at the end. We should repeat the same process every time when the array is filled and want to add new characters.
3. Since this operation must be done by every developer, to avoid this repeat operation, SUN has given the String class in java.lang package.
4. Hence using the String class we can store a sequence of characters without size limitation.