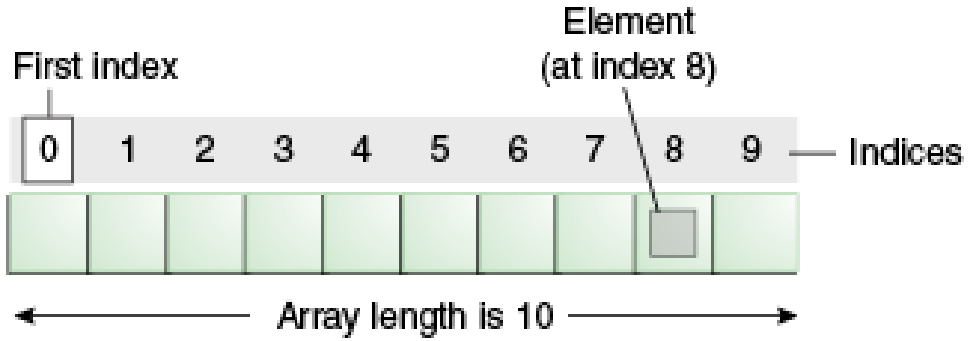
Array in Java



Syntax to Declare an Array in Java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in Java

1. arrayRefVar=new datatype[size];

```java
package HKG;
class Array1
{
public static void main(String args[])
{
int arr[]={20,12,14,16,3};
for(int i=0;i<arr.length;i++)//length is the property of array
System.out.print(arr[i]+" ");
}
}
```

```java
package HKG;
class Array_ForEach
{
public static void main(String[] args)
{
int arr[]={20,12,14,16,3};
for(int i:arr)
System.out.print(i+" ");
}
}
```

Anonymous Array in Java

```
package HKG;
public class AnonymousArr
{
    static void printArray(int arr[])
    {
       for(int i=0;i<arr.length;i++)
        System.out.print(arr[i]);
    }
public static void main(String args[])
    {
    //int[] arr=new int[] {1,3,5,7,9}; printArray(arr);// initializing and passing array
    printArray(new int[]{1,3,5,7,9});//passing anonymous array
    }
}
```

ArrayIndexOutOfBoundsException

The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

```
package HKG;
public class ArrayIndexOutOfBoundsException
{
public static void main(String args[])
{
int arr[]={50,60,70,80};
for(int i=0;i<=arr.length;i++)
{
System.out.println(arr[i]);
}
}
}
```

Multidimensional Array in Java

Syntax to Declare Multidimensional Array in Java

1.dataType[][] arrayRefVar; (or)
2.dataType [][]arrayRefVar; (or)
3.dataType arrayRefVar[][]; (or)
4.dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

```
1.int[][] arr=new int[3][3];//3 row and 3 column
```

```java
package HKG;


class Multidimensional
{
public static void main(String args[])
{
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
System.out.print(arr[i][j]+" ");
}
System.out.println();
}
}
}
```

```java
package HKG;
class JaggedArray
{
public static void main(String[] args)
{
//declaring a 2D array with odd columns
int arr[][] = new int[3][];
arr[0] = new int[]{1,2,3};
arr[1] = new int[]{1,4,2,6};
arr[2] = new int[]{6,5};
for(int i[]:arr)
{
for(int j:i)
{
System.out.print(j+" ");
}
System.out.println();//new line
}
}
}
```

String

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:

```
String s="welcome";
```

2) By new keyword

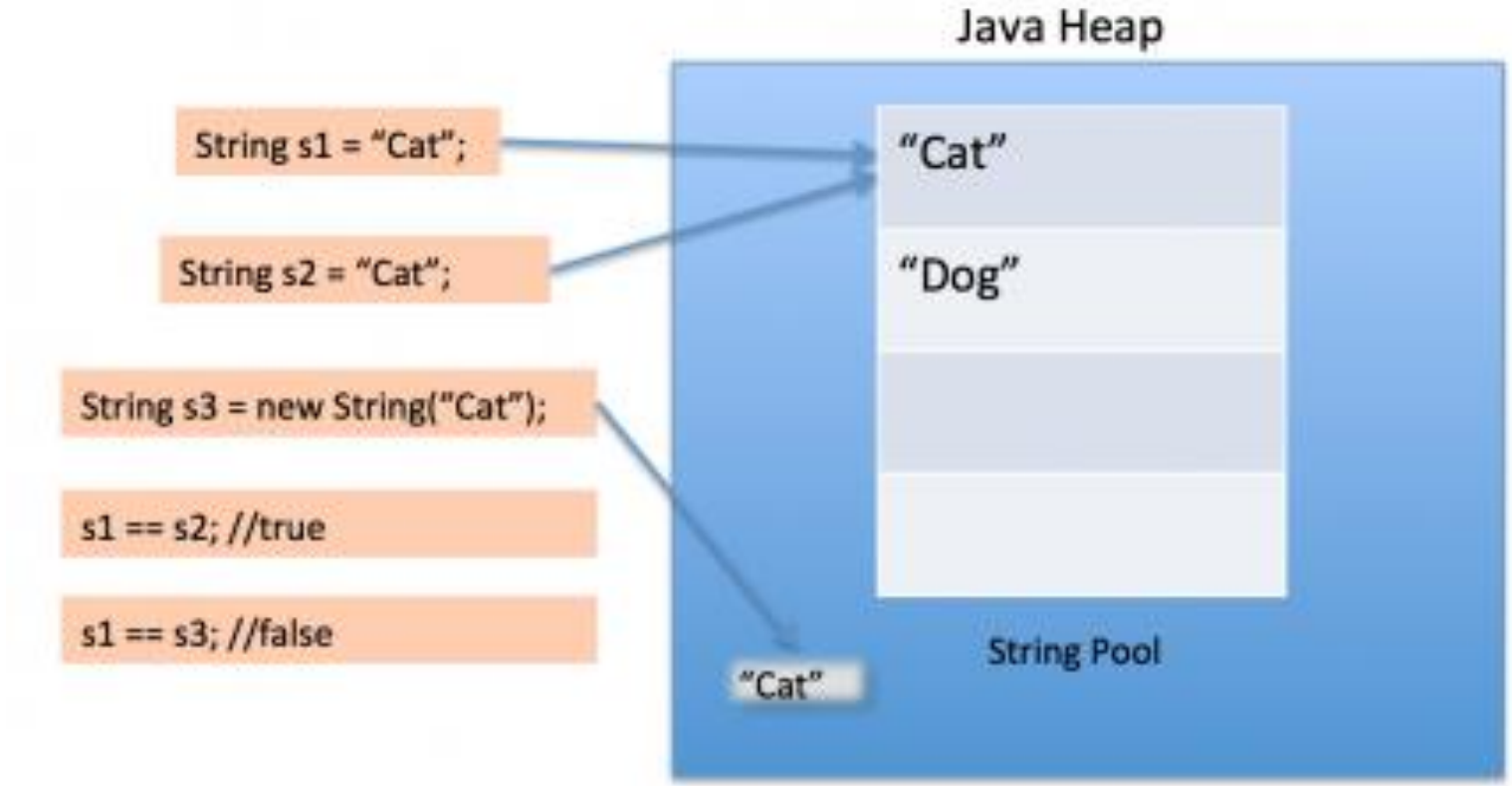
```
1.String s=new String("Welcome");//creates two objects and one reference variable
```

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).

String

```
package HKG;


public class String1{
public static void main(String args[]){
String s1="java";//creating string by Java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating Java string by new
keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```




```
class StrPool
{
    public static void main(String[] args) {
        String s1 = "Cat";
        String s2 = "Cat";
        String s3 = new String("Cat");

        System.out.println("s1 == s2 :" + (s1 == s2));
        System.out.println("s1 == s3 :" + (s1 == s3));
    }

}
```

Methods: String class

- ✓ str.length(): it returns the length of the string mentioned.
- ✓ str.toLowerCase(): it converts the given characters of string into lower case
- ✓ str.toUpperCase(): it converts the given characters of string into upper case.
- ✓ str.trim(): it is used to remove the leading and tailing spaces from the array if there are any.
- ✓ str.substring(int begin): it returns a new string by giving the part of a string from where the index is given.
- ✓ str.substring(int begin, int end): it works same as the above but the starting and ending index both can be
- ✓ given in the substring.
- ✓ str.replace(char old, char new): it replaces the old character with the new character.

- ✓ str.startsWith(string s): to find the particular starting word of the string/to find the initials.
- ✓ str.endsWith(string s): to find particular ending word of the string.
- ✓ str.charAt(int index): to find the particular character present on the index given.
- ✓ str.indexOf(String s): to find the index of the given character.
- ✓ str.lastIndexOf(string s): to find the index of the given character from the end of the array.
- ✓ str.equals(string s): to check whether the contents of two strings are equal or not.
- ✓ str.equalsIgnoreCase(string s): to check whether the contents of two strings are equal or not but it does not depend
- ✓ upon the case of characters.
- ✓ str.compareTo(string s): to compare two strings according to the dictionary order(in accordance with the
- ✓ ASCII codes for cases of characters).
- ✓ str.valueOf(int i): to convert different types of data into string data type.

Java Arrays

Normally, an array is a collection of similar type of elements which has contiguous memory location.

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

we can get the length of the array using the length member.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.

Single Dimensional Array in Java

Syntax to Declare an Array in Java

```
1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];
```

Instantiation of an Array in Java

```
1. arrayRefVar=new datatype[size];
```

Example of Java Array

```
1. //Java Program to illustrate how to declare, instantiate, initialize
2. //and traverse the Java array.
3.
```

```
4. class Testarray{
5. public static void main(String args[]){
6. int a[]=new int[5];//declaration and instantiation
7. a[0]=10;//initialization
8. a[1]=20;
9. a[2]=70;
10. a[3]=40;
11. a[4]=50;
12. //traversing array
13. for(int i=0;i<a.length;i++)//length is the property of array
14. System.out.println(a[i]);
15. }}
```

Declaration, Instantiation and Initialization of Java Array

```
1. //Java Program to illustrate the use of declaration, instantiation
2. //and initialization of Java array in a single line
3. class Testarray1{
4. public static void main(String args[]){
5. int a[]={33,3,4,5};//declaration, instantiation and initialization
6. //printing array
7. for(int i=0;i<a.length;i++)//length is the property of array
8. System.out.println(a[i]);
9. }}
```

For-each Loop for Java Array

We can also print the Java array using **for-each loop**. The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The syntax of the for-each loop is given below:

```
1. for(data_type variable:array){
2. //body of the loop
3. }
```

```
1. //Java Program to print the array elements using for-each loop
2. class Testarray1{
3. public static void main(String args[]){
4. int arr[]={33,3,4,5};
5. //printing array using for-each loop
6. for(int i:arr)
7. System.out.println(i);
8. }}
```

Passing Array to a Method in Java

We can pass the java array to method so that we can reuse the same logic on any array.

```
1. //Java Program to demonstrate the way of passing an array
2. //to method.
3. class Testarray2{
4. //creating a method which receives an array as a parameter
5. static void min(int arr[]){
6. int min=arr[0];
7. for(int i=1;i<arr.length;i++)
8.  if(min>arr[i])
9.   min=arr[i];
10.
11. System.out.println(min);
12. }
13.
14. public static void main(String args[]){
15. int a[]={33,3,4,5};//declaring and initializing an array
16. min(a);//passing array to method
17. }}
```

Anonymous Array in Java

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

```
1. //Java Program to demonstrate the way of passing an anonymous array
2. //to method.
3. public class TestAnonymousArray{
4. //creating a method which receives an array as a parameter
5. static void printArray(int arr[]){
6. for(int i=0;i<arr.length;i++)
7. System.out.println(arr[i]);
8. }
9.
10. public static void main(String args[]){
11. printArray(new int[]{10,22,44,66});//passing anonymous array to method
12. }}
```

Returning Array from the Method

We can also return an array from the method in Java.

```
1. //Java Program to return an array from the method
2. class TestReturnArray{
3. //creating method which returns an array
4. static int[] get(){
5. return new int[]{10,30,50,90,60};
6. }
7.
8. public static void main(String args[]){
9. //calling method which returns an array
10. int arr[]=get();
11. //printing the values of an array
12. for(int i=0;i<arr.length;i++)
13. System.out.println(arr[i]);
14. }}
```

ArrayIndexOutOfBoundsException

The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

```
1. //Java Program to demonstrate the case of
2. //ArrayIndexOutOfBoundsException in a Java Array.
3. public class TestArrayException{
4. public static void main(String args[]){
5. int arr[]={50,60,70,80};
6. for(int i=0;i<=arr.length;i++){
7. System.out.println(arr[i]);
8. }
9. }}
```

What is the class name of Java array?

In Java, an array is an object. For array object, a proxy class is created whose name can be obtained by getClass().getName() method on the object.

```
1. //Java Program to get the class name of array in Java
2. class Testarray4{
3. public static void main(String args[]){
4. //declaration and initialization of array
5. int arr[]={4,4,5};
6. //getting the class name of Java array
7. Class c=arr.getClass();
8. String name=c.getName();
9. //printing the class name of Java array
10. System.out.println(name);
11.
12. }}
```

Copying a Java Array

The two Object arguments specify the array to copy from and the array to copy to. The three int arguments specify the starting position in the source array, the starting position in the destination array, and the number of array elements to copy.

Syntax of arraycopy method

```
public static void arraycopy(
Object src, int srcPos,Object dest, int destPos, int length
)
```

Example of Copying an Array in Java

```
//Java Program to copy a source array into a destination array in Java
class TestArrayCopyDemo {
    public static void main(String[] args) {
        //declaring a source array
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                'i', 'n', 'a', 't', 'e', 'd' };
        //declaring a destination array
        char[] copyTo = new char[7];
        //copying array using System.arraycopy() method
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        //printing the destination array
        System.out.println(String.valueOf(copyTo));
    }
}
```

Cloning an Array in Java

Since, Java array implements the Cloneable interface, we can create the clone of the Java array. If we create the clone of a single-dimensional array, it creates the deep copy of the Java array. It means, it will copy the actual value. But, if we create the clone of a multidimensional array, it creates the shallow copy of the Java array which means it copies the references.

```
//Java Program to clone the array
class Testarray1{
```

```
3.  public static void main(String args[]){
4.  int arr[]={33,3,4,5};
5.  System.out.println("Printing original array:");
6.  for(int i:arr)
7.  System.out.println(i);
8.
9.  System.out.println("Printing clone of the array:");
10. int carr[]=arr.clone();
11. for(int i:carr)
12. System.out.println(i);
13.
14. System.out.println("Are both equal?");
15. System.out.println(arr==carr);
16.
17. }}
```

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:

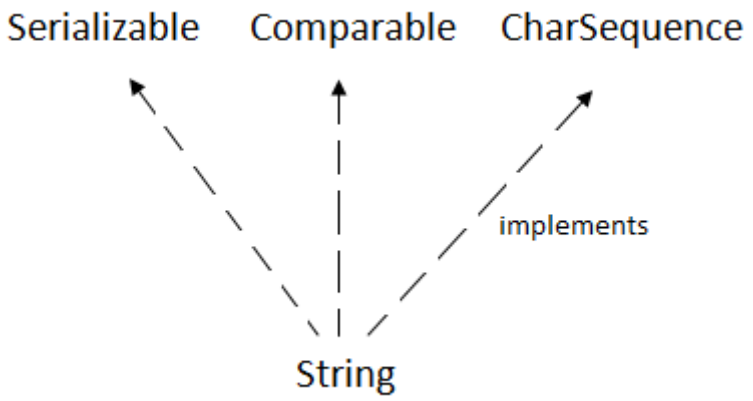
```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

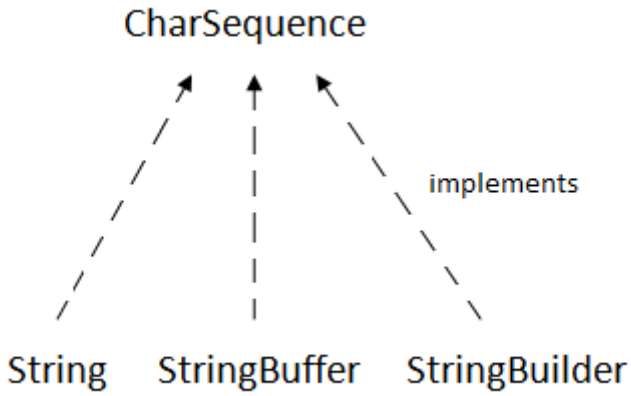
```
1. String s="javatpoint";
```

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.



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.

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

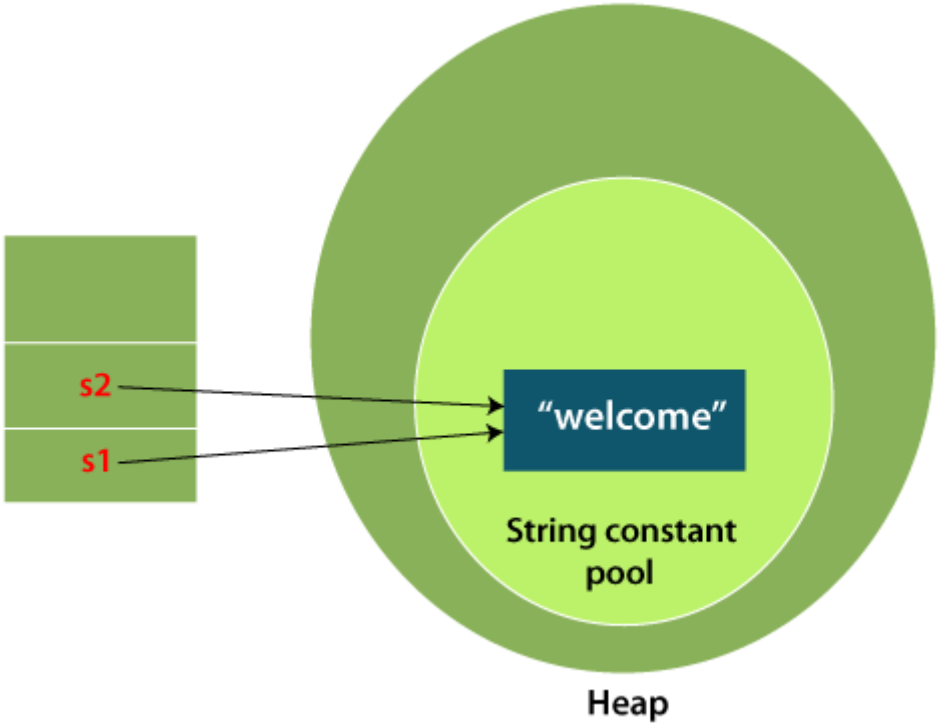
1) String Literal

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".***

2) 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).

Java String Example

StringExample.java

```
1. public class StringExample{
2. public static void main(String args[]){
3. String s1="java";//creating string by Java string literal
4. char ch[]={'s','t','r','i','n','g','s'};
5. String s2=new String(ch);//converting char array to string
6. String s3=new String("example");//creating Java string by new keyword
7. System.out.println(s1);
8. System.out.println(s2);
9. System.out.println(s3);
10. }}
```

Java String class methods

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

No.	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	String substring(int beginIndex) String substring(int beginIndex, int endIndex)	It returns substring

	public class SubstringExample{
	public static void main(String args[]){
	String s1="javatpoint";
	System.out.println(s1.substring(2,4));//returns va
	System.out.println(s1.substring(2));//returns vatpoint
	}}
	
	Output:
	va
	vatpoint
		for given begin index.

	public char[] toCharArray()
	1. **public class** StringToCharArrayExample{
	2. **public static void** main(String args[]){
	3. String s1="hello";
	4. **char**[] ch=s1.toCharArray();
	5. **for**(**int** i=0;i<ch.length;i++){
	6. System.out.println(ch[i]);
	7. }
	8. }}
	Output:
	h
	e
	l
	l
	o	

4	boolean contains(CharSequence s)
	
	public class ContainsExample2 {
	public static void main(String[] args) {
	String str = "Hello Javatpoint readers";
	boolean isContains = str.contains("Javatpoint");
	System.out.println(isContains);
	// Case Sensitive
	System.out.println(str.contains("javatpoint")); // false
	}
	}
		It returns true or false after matching the sequence of char value.

	Output:
	`true`
	`false`	

5	boolean equals(Object another)
	
	public class EqualsExample{
	public static void main(String args[]){
	String s1="javatpoint";
	String s2="javatpoint";
	String s3="JAVATPOINT";
	String s4="python";
	System.out.println(s1.equals(s2));//true because content and case is same
	System.out.println(s1.equals(s3));//false because case is not same
	System.out.println(s1.equals(s4));//false because content is not same
	}}
	
	Output:
	`true`
	`false`
	`false`
		It checks the equality of string with the given object.

6	boolean isEmpty()
	
	public class IsEmptyExample{
	public static void main(String args[]){
	String s1="";
	String s2="javatpoint";
	
	System.out.println(s1.isEmpty());
	System.out.println(s2.isEmpty());
	}}
	
	Output:
	`true`
	`false`
		It checks if string is empty.

7	String concat(String str)
	
	public class ConcatExample{
	public static void main(String args[]){
	String s1="java string";
	// The string s1 does not get changed, even though it is invoking th e method
	// concat(), as it is immutable. Therefore, the explicit assignment is r equired here.
	s1.concat("is immutable");
	System.out.println(s1);
	s1=s1.concat(" is immutable so assign it explicitly");
	System.out.println(s1);
	. }}
	
	Output:
	`java string`
	`java string is immutable so assign it explicitly`
		It concatena tes the specified string.

8	String replace(char old, char new)
	String replace(CharSequence old, CharSequence new)
	public class ReplaceExample1{
	public static void main(String args[]){
	String s1="javatpoint is a very good website";
	String replaceString=s1.replace('a','e');//replaces all occurrences of ' a' to 'e'
	String replaceString=s1.replace("is","was");//replaces all occurrences of "is" to "was"
	
	System.out.println(replaceString);
	}}
	
	Output:
	`jevetpoint is e very good website`
	`jevetpoint was e very good website`
		It replaces all occurrenc es of the specified char value.

9	static String equalsIgnoreCase(String another)
	public class EqualsIgnoreCaseExample{
	public static void main(String args[]){
	String s1="javatpoint";
		It compares another string. It doesn't

	String s2="javatpoint";
	String s3="JAVATPOINT";
	String s4="python";
	System.out.println(s1.equalsIgnoreCase(s2));//true because content and case both are same
	System.out.println(s1.equalsIgnoreCase(s3));//true because case is ignored
	System.out.println(s1.equalsIgnoreCase(s4));//false because content is not same
	. }}
	
	Output:
	true
	true
	false
		check case.

10	String intern()
	Java String intern() Method Example
	public class InternExample{
	public static void main(String args[]){
	String s1=**new** String("hello");
	String s2="hello";
	String s3=s1.intern();//returns string from pool, now it will be same as s2
	System.out.println(s1==s2);//false because reference variables are pointing to different instance
	System.out.println(s2==s3);//true because reference variables are pointing to same instance
	}}
	false
	true
		It returns an interned string.

11	int indexOf(int ch)
	int indexOf(int ch, int fromIndex)
	int indexOf(String substring)
	int indexOf(String substring, int fromIndex)
	
	Java String indexOf() Method Example
	public class IndexOfExample{
	public static void main(String args[]){
		It returns the specified char value index.

	String s1="this is index of example";
	//passing substring
	int index1=s1.indexOf("is");//returns the index of is substring
	int index2=s1.indexOf("index");//returns the index of index substrin
	g
	System.out.println(index1+" "+index2);//2 8
	
	//passing substring with from index
	. **int** index3=s1.indexOf("is",4);//returns the index of is substring after
	4th index
	. System.out.println(index3);//5 i.e. the index of another is
	.
	. //passing char value
	. **int** index4=s1.indexOf('s');//returns the index of s char value
	. System.out.println(index4);//3
	. }}
	
	Output:
	2 8
	5
	3	

12	String toLowerCase()
	public class StringLowerExample{
	public static void main(String args[]){
	String s1="JAVATPOINT HELLO stRIng";
	String s1lower=s1.toLowerCase();
	System.out.println(s1lower);
	}}
		It returns a string in lowercase.

13	String toUpperCase()
	
	public class StringUpperExample{
	public static void main(String args[]){
	String s1="hello string";
	String s1upper=s1.toUpperCase();
	System.out.println(s1upper);
	}}
		It returns a string in uppercase
		.

14	String trim()
	public class StringTrimExample{
	public static void main(String args[]){
	String s1=" hello string ";
	System.out.println(s1+"javatpoint");//without trim()
	System.out.println(s1.trim()+"javatpoint");//with trim()
	}}
	
	Output
	`hello string javatpoint`
	`hello stringjavatpoint`
		It removes beginning and ending spaces of this string.

15	static String valueOf(int value)
	
	public class StringValueOfExample{
	public static void main(String args[]){
	int value=30;
	String s1=String.valueOf(value);
	System.out.println(s1+10);//concatenating string with 10
	}}
	
	Output:
	`3010`
		It converts given type into string. It is an overloaded method.

Immutable String in Java

A String is an unavoidable type of variable while writing any application program. String references are used to store various attributes like username, password, etc. 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.

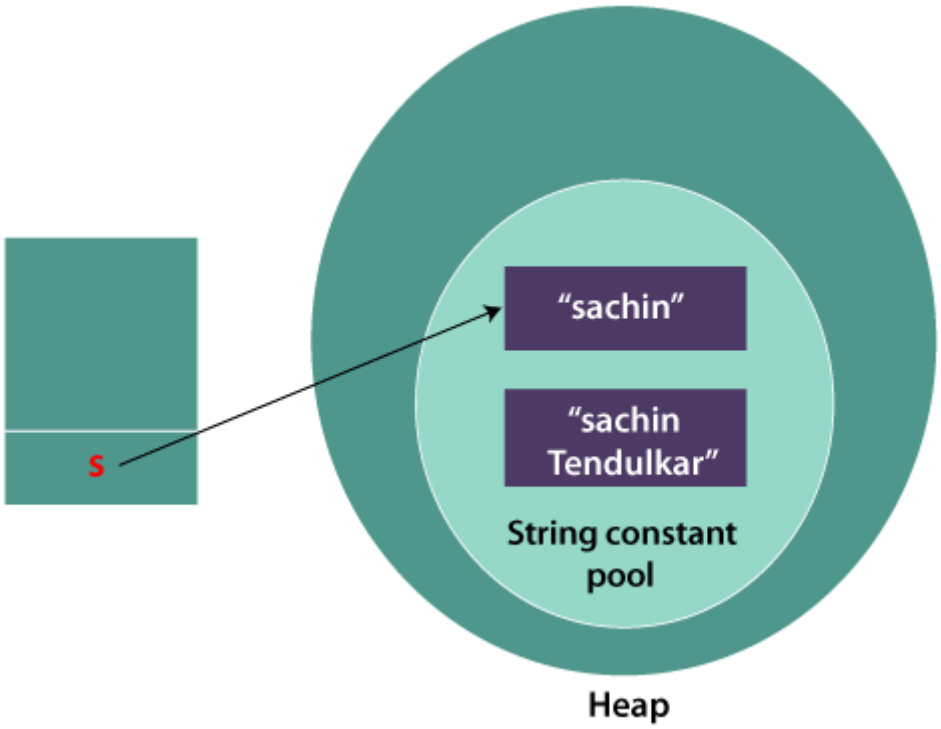
Let's try to understand the concept of immutability by the example given below:

```
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 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 explicitly 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
```

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

Why String objects are immutable in Java?

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.

Following are some features of String which makes String objects immutable.

1. ClassLoader:

A ClassLoader in Java uses a String object as an argument. Consider, if the String object is modifiable, the value might be changed and the class that is supposed to be loaded might be different.

To avoid this kind of misinterpretation, String is immutable.

2. Thread Safe:

As the String object is immutable we don't have to take care of the synchronization that is required while sharing an object across multiple threads.

3. ecurity:

As we have seen in class loading, immutable String objects avoid further errors by loading the correct class. This leads to making the application program more secure. Consider an example of banking software. The username and password cannot be modified by any intruder because String objects are immutable. This can make the application program more secure.

3. Heap Space:

The immutability of String helps to minimize the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the new object. This feature allows Java to use the heap space efficiently.

Why String class is Final in Java?

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

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 Using equals() Method
2. By Using == Operator
3. By compareTo() Method

By Using == operator

The == operator compares references not values.

Teststringcomparison3.java

```
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. }
```

By Using compareTo() method

The String class 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 objects. If:

- **s1 == s2** : The method returns 0.

- **s1 > s2** : The method returns a positive value.
- **s1 < s2** : The method returns a negative value.

```
class Teststringcomparison4{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3="Ratan";
   System.out.println(s1.compareTo(s2));//0
   System.out.println(s1.compareTo(s3));//1(because s1>s3)
   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
 }
}
```

Output:

```
0
1
-1
```

String Concatenation in Java

In Java, String concatenation forms a new String that is the combination of multiple strings. There are two ways to concatenate strings 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:

TestStringConcatenation1.java

```
class TestStringConcatenation1{
 public static void main(String args[]){
   String s="Sachin"+" Tendulkar";
   System.out.println(s);//Sachin Tendulkar
 }
```

```
6. }
```

Java StringBuffer Class

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

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

Important Constructors of StringBuffer Class

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

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)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t 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.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

Important methods of StringBuffer class

What is a mutable String?

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

1) StringBuffer Class 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. }
```

Output:

```
Hello Java
```

2) StringBuffer insert() Method

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

StringBufferExample2.java

```
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. }
```

Output:

```
HJavaello
```

3) StringBuffer replace() Method

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

StringBufferExample3.java

```
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. }
```

Output:

```
HJavalo
```

4) StringBuffer delete() Method

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

StringBufferExample4.java

```
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. }
7. }
```

Output:

```
Hlo
```

5) StringBuffer reverse() Method

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

StringBufferExample5.java

```
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. }
7. }
```

Output:

```
olleH
```

6) StringBuffer capacity() Method

The capacity() method of the 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.

StringBufferExample6.java

```
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. }
10. }
```

Output:

```
16
16
34
```

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of the 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.

StringBufferExample7.java

```
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. }
14. }
```

Output:

```
16
16
34
34
70
```

Java StringBuilder Class

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

Important Constructors of StringBuilder class

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

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 is 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.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

Java StringBuilder Examples

) StringBuilder append() method

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

StringBuilderExample.java

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

Output:

```
Hello Java
```

2) StringBuilder insert() method

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

StringBuilderExample2.java

```
1. class StringBuilderExample2{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello ");
4. sb.insert(1,"Java");//now original string is changed
5. System.out.println(sb);//prints HJavaello
6. }
7. }
```

3) StringBuilder replace() method

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

StringBuilderExample3.java

```
1. class StringBuilderExample3{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.replace(1,3,"Java");
5. System.out.println(sb);//prints HJavalo
6. }
7. }
```

Output:

```
HJavalo
```

StringBuilder delete() method

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

StringBuilderExample4.java

```
1. class StringBuilderExample4{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.delete(1,3);
5. System.out.println(sb);//prints Hlo
6. }
7. }
```

Output:

```
Hlo
```

5) StringBuilder reverse() method

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

StringBuilderExample5.java

```
1. class StringBuilderExample5{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.reverse();
5. System.out.println(sb);//prints olleH
6. }
7. }
```

Output:

```
olleH
```

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.

StringBuilderExample6.java

```
1. class StringBuilderExample6{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder();
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. }
10. }
```

```
16
16
34
```

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.

StringBuilderExample7.java

```
1. class StringBuilderExample7{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder();
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. }
14. }
```

Output:

```
16
16
34
34
70
```