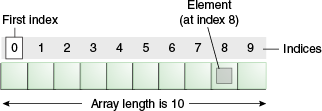
# Week-7 Arrays & Strings

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. we can also create single dimensional or multidimensional arrays in Java.



### Disadvantages

* **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

**Types of Array in java**

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

## Single Dimensional Array in Java

#### Syntax to Declare an Array in Java

dataType[] arr; (or) dataType []arr; (or)

dataType arr[];

#### Instantiation of an Array in Java

arrayRefVar=**new** datatype[size];

//Java Program to illustrate how to declare, instantiate, initialize

//and traverse the Java array.

**class** Testarray{

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

**int** a[]=**new int**[5];//declaration and instantiation a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//traversing array

**for**(**int** i=0;i<a.length;i++)//length is the property of array System.out.println(a[i]);

}

}

**Declaration, Instantiation and Initialization of Java Array**

**int** a[]={33,3,4,5};//declaration, instantiation and initialization

//Java Program to illustrate the use of declaration, instantiation and initialization of Java //array in a single line

**class** Testarray1{

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

**int** a[]={33,3,4,5};//declaration, instantiation and initialization

//printing array

**for**(**int** i=0;i<a.length;i++)//length is the property of array System.out.println(a[i]);

}

}

## For-each Loop for Java Array

We can also print the Java array using [**for-each loop**.](https://www.javatpoint.com/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:

**for**(data\_type variable:array){

//body of the loop

}

/Java Program to print the array elements using for-each loop

**class** Testarray1{

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

**int** arr[]={33,3,4,5};

//printing array using for-each loop

**for**(**int** i:arr) System.out.println(i);

}

}

## Multidimensional Array in Java

An array with more than one dimension is known as multidimensional array, in such case, data is stored in row and column based index (also known as matrix form).

#### Syntax to Declare Multidimensional Array in Java

dataType[][] arrayRefVar; (or) dataType [][]arrayRefVar; (or) dataType arrayRefVar[][]; (or) dataType []arrayRefVar[];

**Example to instantiate Multidimensional Array in Java int**[][] arr=**new int**[3][3];//3 row and 3 column

**Example of Multidimensional Java Array**

//Java Program to illustrate the use of multidimensional array

**class** Testarray3{

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

//declaring and initializing 2D array

**int** arr[][]={{1,2,3},{2,4,5},{4,4,5}};

//printing 2D array **for**(**int** i=0;i<3;i++){ **for**(**int** j=0;j<3;j++){

System.out.print(arr[i][j]+" ");

}

System.out.println();

}

}

}

//Java Program to demonstrate the addition of two matrices in Java

**class** Testarray5{

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

//creating two matrices

**int** a[][]={{1,3,4},{3,4,5}};

**int** b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices

**int** c[][]=**new int**[2][3];

//adding and printing addition of 2 matrices

**for**(**int** i=0;i<2;i++){ **for**(**int** j=0;j<3;j++){ c[i][j]=a[i][j]+b[i][j];

System.out.print(c[i][j]+" ");

}

System.out.println();//new line

}

}

}

## String in Java

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.

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.

### How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

## String Literal

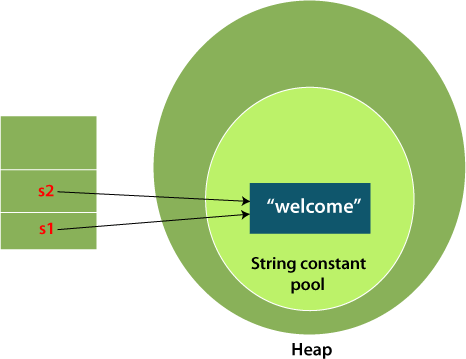
Java String literal is created by using double quotes. For Example: String s="welcome";

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).

## By new keyword

String s=**new** String("Welcome”) ;//creates two objects and one reference variable

In such case, [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) 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

**public class** StringExample{

**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);

}

}

## 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)](https://www.javatpoint.com/java-string-charat) | It returns char value for the particular index |
| 2 | [int length()](https://www.javatpoint.com/java-string-length) | It returns string length |
| 3 | [String substring(int beginIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index. |
| 4 | [String substring(int beginIndex, int](https://www.javatpoint.com/java-string-substring) [endIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index and end index. |
| 5 | [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains) | It returns true or false after matching the sequence of char value. |
| 6 | [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals) | It checks the equality of string with the given object. |
| 7 | [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty) | It checks if string is empty. |
| 8 | [String concat(String str)](https://www.javatpoint.com/java-string-concat) | It concatenates the specified string. |

|  |  |  |
| --- | --- | --- |
| 9 | [String replace(char old, char new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified char value. |
| 10 | [String replace(CharSequence old,](https://www.javatpoint.com/java-string-replace) [CharSequence new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified CharSequence. |
| 11 | [static String equalsIgnoreCase(String](https://www.javatpoint.com/java-string-equalsignorecase) [another)](https://www.javatpoint.com/java-string-equalsignorecase) | It compares another string. It doesn't check case. |
| 12 | [int indexOf(char ch)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index. |
| 13 | [int indexOf(char ch, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index starting with given index. |
| 14 | [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index. |
| 15 | [int indexOf(String substring, int](https://www.javatpoint.com/java-string-indexof) [fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index starting with given index. |
| 16 | [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase. |
| 17 | [String toLowerCase(Locale l)](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase using specified locale. |
| 18 | [String toUpperCase()](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase. |
| 19 | [String toUpperCase(Locale l)](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase using specified locale. |
| 20 | [String trim()](https://www.javatpoint.com/java-string-trim) | It removes beginning and ending spaces of this string. |

# 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:

**Testimmutablestring.java class** Testimmutablestring{

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

String s="Sachin";

s.concat(" Tendulkar");//concat() method appends the string at the end System.out.println(s);//will print Sachin because strings are immutable objects

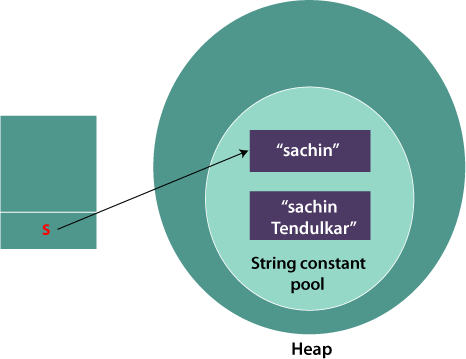
}

}

#### 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:

**Testimmutablestring1.java class** Testimmutablestring1{

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

String s="Sachin"; s=s.concat(" Tendulkar"); System.out.println(s);

}

}

#### 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.

# 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.

## 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. |

**Important methods of StringBuffer class**

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |

|  |  |  |
| --- | --- | --- |
| public synchronized StringBuffer | append(String s) | It 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) | It 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) | It is used to replace the string from specified startIndex and endIndex. |
| public synchronized StringBuffer | delete(int startIndex, int endIndex) | It is used to delete the string from specified startIndex and endIndex. |
| public synchronized StringBuffer | reverse() | is used to reverse the string. |
| public int | capacity() | It is used to return the current capacity. |
| public void | ensureCapacity(int minimumCapacity) | It is used to ensure the capacity at least equal to the given minimum. |
| public char | charAt(int index) | It is used to return the character at the specified position. |
| public int | length() | It is used to return the length of the string i.e. total number of characters. |
| public String | substring(int beginIndex) | It is used to return the substring from the specified beginIndex. |
| public String | substring(int beginIndex, int endIndex) | It is used to return the substring from the specified beginIndex and endIndex. |

## 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.

**class** StringBufferExample{

**public static void** main(String args[]){ StringBuffer sb=**new** StringBuffer("Hello "); sb.append("Java");//now original string is changed System.out.println(sb);//prints Hello Java

}

}