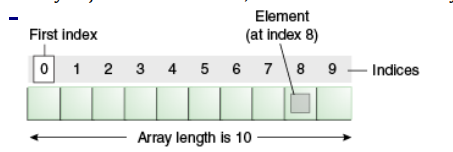
**Array & Strings**

**Java Array**

* Normally, array is a collection of similar type of elements that have contiguous memory location.
* Java array is an object the contains elements of similar data type. 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, first element of the array is stored at 0 index.



**Advantage of Java Array**

* **Code Optimization**: It makes the code optimized, we can retrieve or sort the data easily.
* **Random access**: We can get any data located at any index position

**Disadvantage of Java Array**

* **Size Limit**: We can store only 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.

**How to declare an array in Java?**

**dataType[] arrayName;**

**How to create Arrays**

**arrayRefVar = new dataType[arraySize];**

**How to Initialize Arrays in Java?**

**int[] age = {12, 4, 5, 2, 5};**

**Note**:

* Array indices always start from 0. That is, the first element of an array is at index 0.
* If the size of an array is n, then the last element of the array will be at index n-1.

**How to Access Elements of an Array in Java?**

* We can access the element of an array using the index number. Here is the syntax for accessing elements of an array,

**// access array elements**

**array[index]**

* Let's see an example of accessing array elements using index numbers.

**public** **class** AccessArrayElements {

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

// create an array

**int**[] age = {12, 4, 5, 2, 5};

// access each array elements

System.***out***.println("Accessing Elements of Array:");

System.***out***.println("First Element: " + age[0]);

System.***out***.println("Second Element: " + age[1]);

System.***out***.println("Third Element: " + age[2]);

System.***out***.println("Fourth Element: " + age[3]);

}}

**Types of Array in java**

There are two types of array.

1. Single Dimensional Array
2. Multidimensional Array

**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 single dimensional java arrayLet's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array

**public** **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;

//printing 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**

* We can declare, instantiate and initialize the java array together by:

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

* Let's see the simple example to print this array

**public** **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]);

}

}

**Passing Array to method in java**

* + - We can pass the java array to method so that we can reuse the same logic on any array.
    - Let's see the simple example to get minimum number of an array using method

**public** **class** PassingArray {

**static** **void** min(**int** arr[]){

**int** min=arr[0];

**for**(**int** i=1;i<arr.length;i++)

**if**(min>arr[i])

min=arr[i];

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

}

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

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

*min*(a);//passing array to method

}

}

**Multidimensional array in java**

* In such case, data is stored in row and column based index (also known as matrix form).

**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

**int[][] arr=new int[3][3];//3 row and 3 column**

* Example to initialize Multidimensional Array in java

**arr[0][0]=1;**

**arr[0][1]=2;**

**arr[0][2]=3;**

**arr[1][0]=4;**

* Example of Multidimensional java array Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

**public** **class** Multidiamensional {

**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();

}

}

}

**Copying a java array**

* We can copy an array to another by the arraycopy method of System class.

**Syntax of arraycopy method**

**public static void arraycopy(**

**Object src, int srcPos,Object dest, int destPos, int length**

**)**

* Example of arraycopy method

**public** **class** TestArrayCopyDemo {

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

**char**[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };

**char**[] copyTo = **new** **char**[7];

System.*arraycopy*(copyFrom, 2, copyTo, 0, 7);

System.***out***.println(**new** String(copyTo));

}

}

**Addition of 2 matrices in java**

* Let's see a simple example that adds two matrices

**public** **class** Addition2Matrices {

**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

}

}

}

**Strings**

* A string is a sequence of characters. For example, "hello" is a string containing a sequence ofcharacters 'h', 'e', 'l', 'l', and 'o'.
* Use double quotes to represent a string in Java.

**String Class**

How to create a string object?

There are two ways to create String object:

• By string literal e.g : String str1 = "Hello Java";

• By new keyword e.g : String str2 = new String("Hello Java");

Note: all string variables are instances of the String class (are not primitive types (like int, char, etc))

**String Class declaration in Java**

public final class String extends Object implements Serializable, Comparable<String>, CharSequence

{

}

**String Basics Example 1-**

**public** **class** StringMain {

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

// create strings

String s1 = "Java";

String s2 = "CPP";

String s3 = "Angular";

// print strings

System.***out***.println(s1);

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

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

}

}

**String Example 2-**

**public** **class** FirstStringProgram {

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

System.***out***.println("First String program");

// 1. way to create a string - literal way

String s1="Hello All";

System.***out***.println(s1);

//2. way to create a string - new operator

String s2=**new** String("Hello Java");

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

System.***out***.println("Length of S1 :" + s1.length());

System.***out***.println("Length of S2 :" + s2.length());

System.***out***.println("S1 toUpper:" +s1.toUpperCase());

System.***out***.println("S1 toLower:" +s1.toLowerCase())

// Each string is internally an array.

System.***out***.println(s1.charAt(0));

System.***out***.println(s1.charAt(6));

}

}

**Methods of Strings-**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [char charAt(int index)](https://www.javatpoint.com/java-string-charat) | It returns char value for the particular index |
| [int length()](https://www.javatpoint.com/java-string-length) | It returns string length |
| [static String format(String format, Object... args)](https://www.javatpoint.com/java-string-format) | It returns a formatted string. |
| [static String format(Locale l, String format, Object... args)](https://www.javatpoint.com/java-string-format) | It returns formatted string with given locale. |
| [String substring(int beginIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index. |
| [String substring(int beginIndex, int endIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index and end index. |
| [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains) | It returns true or false after matching the sequence of char value. |
| [static String join(CharSequence delimiter, CharSequence... elements)](https://www.javatpoint.com/java-string-join) | It returns a joined string. |
| [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](https://www.javatpoint.com/java-string-join) | It returns a joined string. |
| [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals) | It checks the equality of string with the given object. |
| [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty) | It checks if string is empty. |
| [String concat(String str)](https://www.javatpoint.com/java-string-concat) | It concatenates the specified string. |
| [String replace(char old, char new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified char value. |
| [String replace(CharSequence old, CharSequence new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified CharSequence. |
| [static String equalsIgnoreCase(String another)](https://www.javatpoint.com/java-string-equalsignorecase) | It compares another string. It doesn't check case. |
| [String[] split(String regex)](https://www.javatpoint.com/java-string-split) | It returns a split string matching regex. |
| [String[] split(String regex, int limit)](https://www.javatpoint.com/java-string-split) | It returns a split string matching regex and limit. |
| [String intern()](https://www.javatpoint.com/java-string-intern) | It returns an interned string. |
| [int indexOf(int ch)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index. |
| [int indexOf(int ch, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index starting with given index. |
| [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index. |
| [int indexOf(String substring, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index starting with given index. |
| [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase. |
| [String toLowerCase(Locale l)](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase using specified locale. |
| [String toUpperCase()](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase. |
| [String toUpperCase(Locale l)](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase using specified locale. |
| [String trim()](https://www.javatpoint.com/java-string-trim) | It removes beginning and ending spaces of this string. |
| [static String valueOf(int value)](https://www.javatpoint.com/java-string-valueof) | It converts given type into string. It is an overloaded method. |

**One of the most important characteristics of a string in Java is that they are immutable. In other words, oncecreated, the internal state of a string remains the same throughout the execution of the program. Thisimmutability is achieved through the use of a special string constant pool in the heap.**

**SCP: String constant pool or String literal Pool.**

* Literal: Things which doesn’t change. E.g 10, 120, 3.145, “Apple”, “Hello” etc.
* String literal in Java is created by using double quotes.

For example:

String s="Hello"

* The string literal is always created in the string constant pool. In Java, String constant pool is a special area thatis used for storing string objects.
* Internally, String class uses a string constant pool (SCP). This SCP area is part of the method area (Permgen)until Java 1.6 version.
* From Java 1.7 onwards, SCP area is moved in the heap memory because SCP is a fixed size in the methodarea but in the heap memory, SCP can be expandable.
* Therefore, the string constant pool has been moved to heap area for memory utilization only.
* Whenever we create a string literal in Java, JVM checks string constant pool first. If the string already exists in string constant pool, no new string object will be created in the string pool by JVM.

**Internal about the strings**

//creates two objects. One is in heap because of new keyword and another one is in SCP for future use.

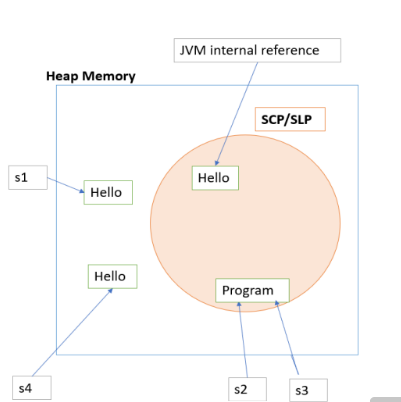
String s1=new String("Hello")

//creates only one object, in SCP

String s2="Program"; // Literal (constant/doesn't change)

String s3="Program"; // Literal – Does not create any Object as literal is already present in SCP.

String s4=new String("Hello"); // will create string Object in heap.



Thus, its preferable to create a string without new as you would have less string objects in memory.

String s4="Java"

String objects those are available in SCP are not applicable for Garbage collection as reference ismaintain by JVM.

**Interview Question.**

**String s = new string ("ABC"). How many objects are created after the above statement?**

* + So, when the class containing this code is loaded, JVM finds the string literal "ABC", creates a stringobject and puts its reference into string constant pool.Now when the statement is executed, JVM is forced to create a new string object with contents "ABC" becauseof the new operator. This time no reference will be put into string constant pool because already there isreference for the same literal.
  + So, two objects are created by the statement.

**Key points:**

1. Strings are objects in Java.

2. When we create a String object, it cannot be changed further. In other words, once a Stringobject has been created, we cannot change any characters in string. Therefore, string object isimmutable in Java.

4. String class in Java has numerous methods for string manipulation like length() will return thetotal number of characters in the string.

5. String is not a primitive data type. It is a reference data type.

**Immutable String**

* Java Strings are Immutable
* In Java, strings are immutable. This means, once we create a string, we cannot change that string.

// create a string

String str1 = "Hello! ";

str1 = str1 + "Hi ";

* Let's see what has happened here,

• JVM takes the first string "Hello! "

• creates a new string by adding "Hi" to the first string

• assign the new string "Hello! Hi" to the str1 variable

• the first string "Hello! " remains unchanged in SCP

**Why Strings are immutable ?**

* Let’s suppose we have huge number of person objects, and each person has city associated.

String city1="Pune"

String city2="Pune"

String city3="Pune"

Diagram

Description automatically generated

String cityn="Pune"

String city20="Mumbai"

* Thus, why to create multiple string with value "Pune". Create single string and make it immutable.

**Why Strings are immutable or final in Java ?**

There are several benefits of String because it’s immutable and final.

• String Pool is possible because String is immutable in java.

• It increases security because any hacker can’t change its value and it’s used for storing sensitive information such as database username, password etc.

• Since String is immutable, it’s safe to use in multi-threading and we don’t need any synchronization.

• Strings are used in java classloader and immutability provides security that correct class is getting loaded by Classloader.

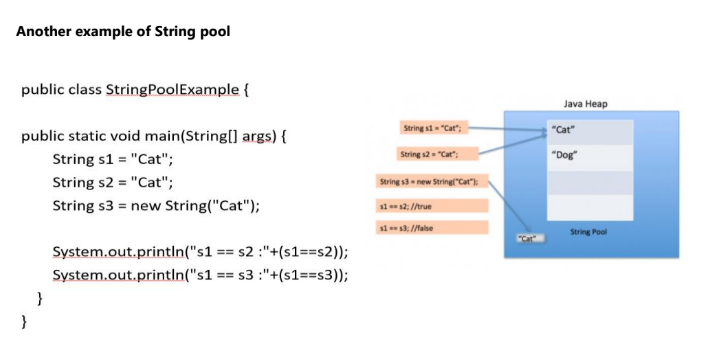
**String comparison**

* There are three ways to compare String in Java:
  1. By Using equals() Method :- It compares values of string for equality.(string class has overridden equals() method.
  2. By Using == Operator:- The == operator compares references not values.
  3. By 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. "Hello" > "Aello"

s1 < s2 : The method returns a negative value.



**String buffer**

* StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.
* StringBuffer may have characters and substrings inserted in the middle or appended to the end. It willautomatically grow to make room for such additions and often has more characters pre-allocated than are actually needed, to allow room for growth.

Graphical user interface, text, application, email

Description automatically generated

**StringBufferExample**

**public** **class** StringBufferExample {

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

// Its a thread-safe

StringBuffer sb=**new** StringBuffer("Hello SB"); // capacity 16 + length of parameter

System.***out***.println("Capacity :"+sb.capacity());

sb.append("I am learning Java"); // increase the capacity by (current\*2)+2.

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

sb.insert(8, "!!"); // You can insert substring into the string using StringBuffer

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

System.***out***.println("Capacity :"+sb.capacity());

}

}

**What is the difference between the String and StringBuffer?**

* The String is a final class in Java. The String is immutable. That means we can not change the value of the Stringobject afterword.
* Since the string is widely used in applications, we have to perform several operations on the String object,which generates a new String object each time, and all previous objects will be garbage object putting the pressure on the Garbage collector.
* Hence, the Java team introduced the StringBuffer class. It is a mutable String object, which means you can change its value.
* The string is immutable, but the StringBuffer is mutable.

Table

Description automatically generated

Table

Description automatically generated

