UNIT-IV

Java String Handling: String Constructors, Special string operations, Character Extraction, String Comparisons, Modifying a string, String Buffer.

Collections Framework: Overview, Collection Interfaces, Collection Classes, Accessing a collection via Iterator, Working with Maps, Generics

Unit-IV-PART-I

Java String handling

A string is a sequence of characters surrounded by double quotations.

In a java programming language, a string is the object of a built-in class **String**.

In the background, the string values are organized as an array of a character data type.

The string created using a character array can not be extended.

It does not allow to append more characters after its definition, but it can be modified.

Let's look at the following example java code.

Example

```
char[] name = {'J', 'a', 'v', 'a', ' ', 'T', 'u', 't', 'o', 'r', 'i', 'a', 'l', 's'};//name[14] = '@';      //ArrayIndexOutOfBoundsException
name[5] = '-';System.out.println(name);
```

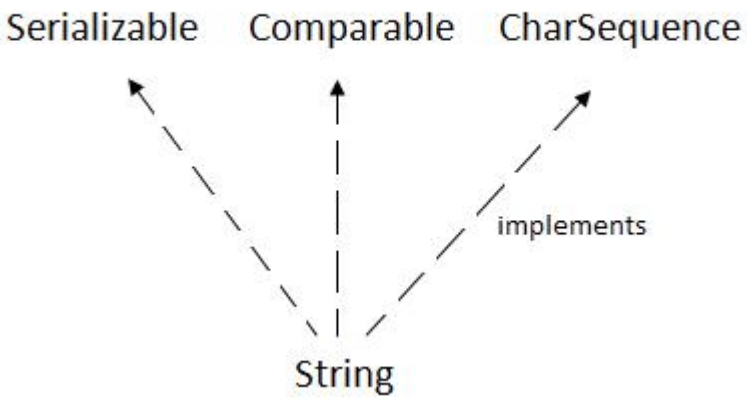
The **String** class defined in the package **java.lang** package. The String class implements **Serializable**, **Comparable**, and **CharSequence** interfaces.

The string created using the **String** class can be extended. It allows us to add more characters after its definition, and also it can be modified.

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.

java.lang.String

class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

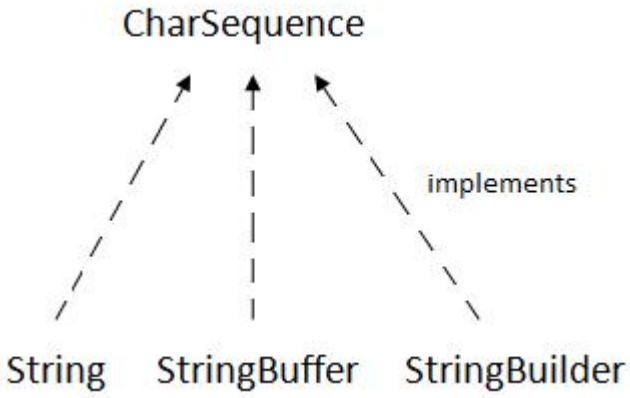


CharSequence Interface

The CharSequence interface is used to represent the sequence of characters.

String, StringBuffer and StringBuilder classes implement it.

It means, we can create strings in Java by using these three classes.



The Java String is immutable which means it cannot be changed.

Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

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

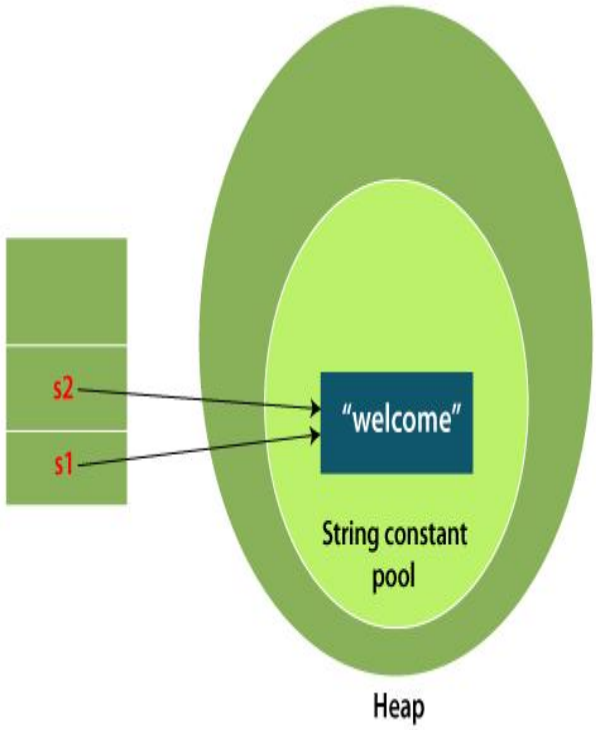
Java String literal is created by using double quotes. For Example:

```
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.

By new keyword

String s=new String("Welcome")

Java String Example

StringExample.java

```
1.	public class StringExample
2.	{
3.	public static void main(String args[])
4.	{
5.	String s1="java";//creating string by Java string literal
6.	char ch[]={'s','t','r','i','n','g','s'};
7.	String s2=new String(ch);//converting char array to string
8.	String s3=new String("example");//creating Java string by new key word
9.	System.out.println(s1);
10.	System.out.println(s2);
11.	System.out.println(s3);
12.	}
13.	}
```

Output:

```
java
strings
example
```

The above code, converts a **char** array into a **String** object. And displays the String objects **s1, s2**, and **s3** on console using **println()** 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.

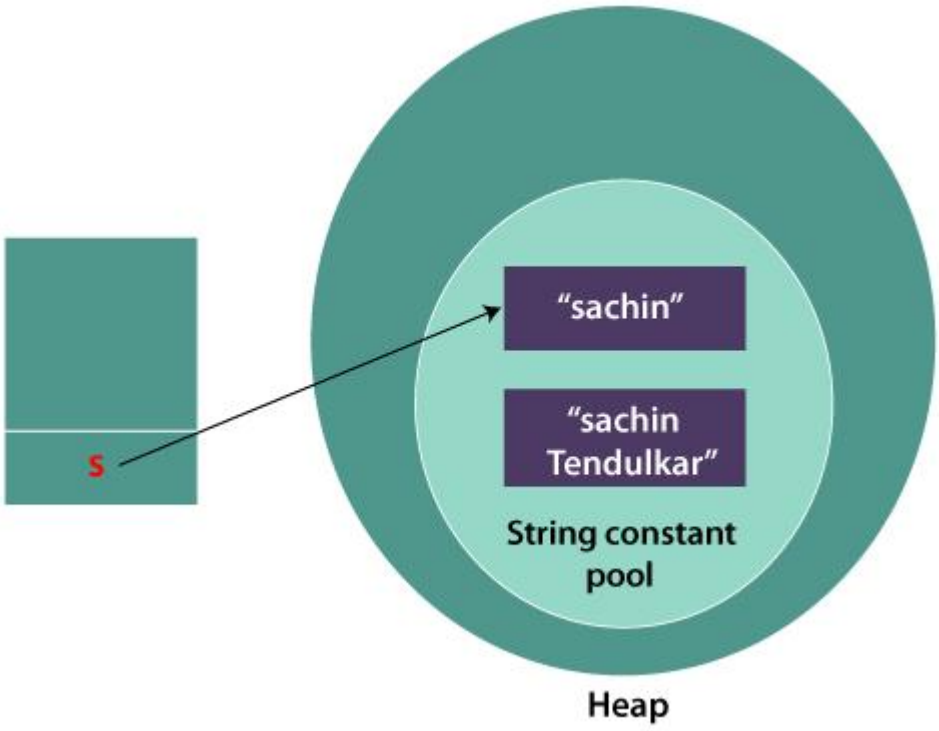
Ex:-

```
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
 }
}
```

O/P:-

```
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".

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 String Operations

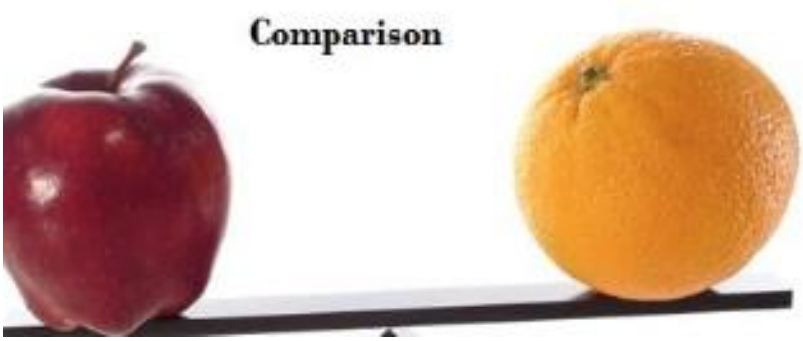
Java String compare

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



1) By Using equals() Method

The String class equals() method compares the original content of the string. It compares values of string for equality.

```
1.  class Teststringcomparison2
2.  {
3.   public static void main(String args[])
4.  {
5.    String s1="Sachin";
6.    String s2="SACHIN";
```

```
7.              System.out.println(s1.equals(s2));//false
8.            System.out.println(s1.equalsIgnoreCase(s2));//true
9.          }
10.        }
```

OUTPUT:-

false

true

In the above program, the methods of String class are used.

The equals() method returns true if String objects are matching and both strings are of same case. equalsIgnoreCase() returns true regardless of cases of strings.

2) By Using == operator

The == operator compares references not values.

Teststringcomparison3.java

```
1.          class Teststringcomparison3
2.          {
3.           public static void main(String args[])
4.          {
5.            String s1="Sachin";
6.            String s2="Sachin";
7.            String s3=new String("Sachin");
8.          System.out.println(s1==s2);//true (because both refer to same ins
9.          System.out.println(s1==s3);//false(because s3 refers to instance cre
ated in nonpool)
10.          }
11.         }
```

Output:

```
True     false
```

3) 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.

Teststringcomparison4.java

```
1.  class Teststringcomparison4{
2.   public static void main(String args[]){
3.     String s1="Sachin";
4.     String s2="Sachin";
5.     String s3="Ratan";
6.     System.out.println(s1.compareTo(s2));//0
7.     System.out.println(s1.compareTo(s3));//1(because s1>s3)
8.     System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
9.    }
10.  }
```

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

```
1.  class TestStringConcatenation1{
2.   public static void main(String args[]){
3.     String s="Sachin"+" Tendulkar";
4.     System.out.println(s);//Sachin Tendulkar
5.   }
6.  }
```

Output:

```
Sachin Tendulkar
```

2) String Concatenation by concat() method

The String concat() method concatenates the specified string to the end of current string. Syntax:

```
1.  public String concat(String another)
1.  class TestStringConcatenation3{
2.   public static void main(String args[]){
3.     String s1="Sachin ";
4.     String s2="Tendulkar";
5.     String s3=s1.concat(s2);
6.     System.out.println(s3);//Sachin Tendulkar
7.   }
8.  }
```

Output:

Sachin Tendulkar

The above Java program, concatenates two String objects **s1** and **s2** using **concat()** method and stores the result into **s3** object.

Java string methods

String charAt()

The **Java String class charAt()** method returns a char value at the given index number.

The index number starts from 0 and goes to n-1, where n is the length of the string.

It returns **StringIndexOutOfBoundsException,** if the given index number is greater than or equal to this string length or a negative number.

Syntax

```
1.  public char charAt(int index)
```

The method accepts **index** as a parameter. The starting index is 0. It returns a character at a specific index position in a string. It throws **StringIndexOutOfBoundsException** if the index is a negative value or greater than this string length.

Ex:-

```
1.  public class CharAtExample{
2.  public static void main(String args[]){
3.  String name="welcome to java";
4.  char ch=name.charAt(4);//returns the char value at the 4th index
5.  System.out.println(ch);
```

```
6.      }}
```

Output:-o

String contains()

The **Java String class contains()** method searches the sequence of characters in this string.

It returns true if the sequence of char values is found in this string otherwise returns false.

Syntax:-

```
1.      public boolean contains(CharSequence sequence)
```

```
1.      class ContainsExample{
2.      public static void main(String args[]){
3.      String name="what do you know about me";
4.      System.out.println(name.contains("do you know"));
5.      System.out.println(name.contains("about"));
6.      System.out.println(name.contains("hello"));
7.      }}
```

Output:

true

true

False

String indexOf()

The **Java String class indexOf()** method returns the position of the first occurrence of the specified character or string in a specified string.

```
1.  public class IndexOfExample2 {
2.     public static void main(String[] args) {
3.        String s1 = "This is indexOf method";
4.        // Passing Substring
5.         int index = s1.indexOf("method"); //Returns the index of this substring
6.        System.out.println("index of substring "+index);
7.     }
8.
9.  }
```

Output:

```
index of substring 16
```

String toLowerCase()

The **java string toLowerCase()** method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.

```
1.  public class StringLowerExample{
2.  public static void main(String args[]){
3.  String s1="WELCOME  HELLO stRIng";
4.  String s1lower=s1.toLowerCase();
5.  System.out.println(s1lower);
6.  }}
```

Output:

```
Welcome hello string
```

String toUpperCase()

The **java string toUpperCase()** method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.

```
1.  public class StringUpperExample{
2.  public static void main(String args[]){
3.  String s1="hello string";
4.  String s1upper=s1.toUpperCase();
5.  System.out.println(s1upper);
6.  }}
```

Output:

```
HELLO STRING
```

String length()

The **Java String class length()** method finds the length of a string. The length of the Java string is the same as the Unicode code units of the string.

Syntax:-

```
1.  public int length()
```

```
1.  public class LengthExample{
2.  public static void main(String args[]){
3.  String s1="javatpoint";
4.  String s2="python";
5.  System.out.println("string length is: "+s1.length());//10 is the lengt
    h of javatpoint string
6.  System.out.println("string length is: "+s2.length());//6 is the length
    of python string
7.  }
8.  }
```

Output:

```
string length is: 10
string length is: 6
```

String valueOf()

The **java string valueOf()** method converts different types of values into string.

By the help of string valueOf() method, you can convert int to string, long to string, boolean to string, character to string, float to string, double to string, object to string and char array to string.

```
1.  public static String valueOf(boolean b)
2.  public static String valueOf(char c)
3.  public static String valueOf(char[] c)
4.  public static String valueOf(int i)
5.  public static String valueOf(long l)
6.  public static String valueOf(float f)
7.  public static String valueOf(double d)
8.  public static String valueOf(Object o)
```

Example:-

```
1.  public class StringValueOfExample
2.  {
3.  public static void main(String args[])
4.  {
5.  int value=30;
6.  String s1=String.valueOf(value);
7.  System.out.println(s1+10);//concatenating string with 10
8.  }
9.  }
```

Output:-

3010

String trim()

The **Java String class trim()** method eliminates leading and trailing spaces. The Unicode value of space character is '\u0020'.

The trim() method in Java string checks this Unicode value before and after the string, if it exists then the method removes the spaces and returns the omitted string.

```
1.  public class StringTrimExample2 {
2.    public static void main(String[] args) {
3.      String s1 ="  hello java string   ";
4.      System.out.println(s1.length());
5.      System.out.println(s1); //Without trim()
6.      String tr = s1.trim();
7.      System.out.println(tr.length());
8.      System.out.println(tr); //With trim()
9.    }
10. }
```

Output

```
22
 hello java string 
17
hello java string
```

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.

1) StringBuffer Class append() Method

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

StringBufferExample.java

```
1.	class StringBufferExample
2.	{
3.	public static void main(String args[])
4.	{
5.	StringBuffer sb=new StringBuffer("Hello ");
6.	sb.append("Java");//now original string is changed
7.	System.out.println(sb);//prints Hello Java
8.	}
9.	}
```

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.  {
3.  public static void main(String args[])
4.  {
5.  StringBuffer sb=new StringBuffer("Hello ");
6.  sb.insert(1,"Java");//now original string is changed
7.  System.out.println(sb);//prints HJavaello
8.  }
9.  }
```

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
```

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.

StringBuilder Examples

Let's see the examples of different methods of StringBuilder class.

1) StringBuilder append() method

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

```
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.  } }
```

Output:

```
HJavaello
```

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
```

4) 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

Modifying string

1.replace() method

String replace() method replaces occurances of character with a specified new character.

```
public class Demo
{
   public static void main(String[] args)
   {
      String str = "Change me";
      System.out.println(str.replace('m','M'));
```

```
    }
}
```

O/P:-

Change Me

substring() method

String substring() method returns a part of the string. substring() method has two override methods.

1. public String substring(int begin);

2. public String substring(int begin, int end);

The first argument represents the starting point of the subtring. If the substring() method is called with only one argument, the subtring returns characters from specified starting point to the end of original string.

```
public class Demo
{
   public static void main(String[] args)
 {
      String str = "0123456789";
      System.out.println(str.substring(4));
      System.out.println(str.substring(4,7));
   }
}
```

O/P:-

456789

456

trim() method

This method returns a string from which any leading and trailing whitespaces has been removed.

```
public class Demo
{
   public static void main(String[] args)
      {
      String str = "   hello  ";
      System.out.println(str.trim());
   }
}
```

O/p:-

```
hello
```