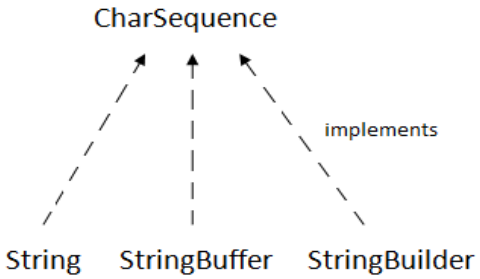
Strings in Java:

Definition:- Strings are sequences of Unicode characters. In many programming languages strings are stored in *arrays* of characters. However, in Java strings are a separate object type, String. The "+" operator is used for concatenation, but all other operations on strings are done with methods in the String class.

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.



The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

Java provides 3 types of string handling classes

String class -defined in "java.lang" package

- Once an object is created, no modifications can be done on that.

StringBuffer class and StringBuilder class– defined in "java.lang" package

- Modifications can be allowed on created object.

StringTokenizer class - defined in "java.util" package

- Used to divide the string into substrings based on tokens.

Declaration & Creation of String:

Syntax: String string_name; //Decl
String_name=new String("string"); //creation

Ex: String sname;
sname=new String(" Java");
(or)
String sname=new String("java"); //Both

Java Strings can be concatenated using the '+' operator

Ex: String Fullname=Name1+Name2;
If Name1="Rgukt",Name2="Basar", we get Fullname="RguktBasar"

String Arrays:

Java also supports the cration & usage of arrays that contain strings

Syntax: String str_array_name[ ]=new String[size];

Ex: String ItemArray[ ]=new String[6];

String Constructors:-

1. String s= new String(); //To create an empty String call the default constructor.
2. char chars[]={'a','b','c','d'};
   String s=new String(chars);
3. String(String str); //Construct a string object by passing another string object.
   String str = "abcd";
   String str2 = new String(str);
4. String(char chars[], int startIndex, int numChars)
   //To create a string by specifying positions from an array of characters
   char chars[]={'a','b','c','d','e','f'};
   String s=new String(chars,2,3); //This initializes **s** with characters "cde".
5. String(byte asciiChars[ ]) // Construct a string from subset of byte array.
6. String(byte asciiChars[ ], int startIndex, int numChars)

Examples:-

```
// Construct one String from another.
        class MakeString {
            public static void main(String args[]) {
                char c[] = {'J', 'a', 'v', 'a'};
                String s1 = new String(c);
                String s2 = new String(s1);
                System.out.println(s1);
                System.out.println(s2);
            }
        }
```

Output:

Java

Java

```
// Construct string from subset of char array.
        class SubStringCons {
            public static void main(String args[]) {
               byte ascii[] = {65, 66, 67, 68, 69, 70 };
                String s1 = new String(ascii);
                System.out.println(s1);
                String s2 = new String(ascii, 2, 3);
                System.out.println(s2);
            }
        }
```

Output:

ABCDEF

CDE

String Methods:-

Method Call	Return Type	Task Performed
s2=s1.toLowerCase()	String	Converts the string s1 to all lowercase
s2=s1.toUpperCase()	String	Converts the String s1 to all Uppercase
s2=s1.replace('x','y')	String	Replaces all appearances of 'x' with 'y'
s2=s1.trim()	String	Remove white spaces at the beginning & end of the string s1
s1.equals(s2)	Boolean	Returns 'true' if s1 is same as s2, same characters in same order (case-sensitive) 'false' otherwise
s1.equalsIgnoreCase(s2)	Boolean	'' '' , ignoring the case of characters
s1.length()	Int	Returns the no. of characters in s1
s1.charAt(n)	char	Returns the nth characters in s1
s1.getChars(m,n,c,0)	Void	Extracts more than one character at a time & copies into a character array 'c' m-source start n-source end 0-Target start
s1.compareTo(s2)	Int	Returns zero, if s1=s2 (case sensitive) -ve , if s1<s2 +ve , if s1>s2
s1.compareToIgnoreCase(s2)	Int	ignoring case sensitive
s1.regionMatches(m,s2,n,c)	Boolean	Compares a specific region inside a string with another specific region in another string True if same Else False m-start index of s1 n-start index of s2 c-no. of characters to compare

s1.regionMatches(B,m,s2,n,c)	Boolean	if B is true ,ignores case
s1.indexOf('ch') s1.indexOf("str")	Int	Searches for the 1st occurrence of a character or string in s1 & returns the index value else returns '-1'
s1.indexOf('ch',n) s1.indexOf("str",n)	Int	'' '' from the nth position in s1
s1.lastIndexOf('ch') s1.lastIndexOf("str")	Int	Searches for last occurrence of a character or string in s1. The search is performed from the end of the string towards beginning & returns the index value else '-1'
S1.lastIndexOf('ch',n) S1.lastIndexOf("str",n)	Int	'' '' from the nth position from end to begin
S1.substring(n)	String	Gives substring starting from nth character
S1.substring(n,m)	String	'' '' '' upto mth character (not including mth character)
S1.startsWith("str")	Boolean	Determines whether a given string 's1' begins with a specific string or not
S1.endsWith("str")	Boolean	'' '' '' '' ends '' ''
S1.concat(s2)	String	Concatenates s1&s2 stored
S1.toCharArray()	Char [ ]	Returns a character array containing a copy of the characters in a string
o.toString()	String	Converts all objects into string objects
String.valueOf(o)	String	Creates a string object of the parameter 'o' (Simple type or Object type)
String.valueOf(var)	String	Converts the parameter value to String representation

Character Extratction:-

```
class getCharsDemo {
public static void main(String args[]) {
        String s = "This is a demo of the      getChars method.";
        int start = 10;
        int end = 14;
        char buf[] = new char[end - start];
        s.getChars(start, end, buf, 0);
        System.out.println (buf);
        }
}
```

Output:

demo

```
class GetBytesDemo{
        public static void main(String[] args){
           String str = "abc“ + ”ABC”;
                byte[] b = str.getBytes();
                //char[] c=str.toCharArray();
                System.out.println(str);
                for(int i=0;i<b.length;i++){
                   System.out.print(b[i]+"  ");
                   //System.out.print(c[i]+"  ");
                }
        }
}
```

Output:

97	**98**	**99**	**65**	**66**	**67**
//a	b	c	A	B	C

String Comparison:

1. boolean **equals**(Object str)
   To compare two strings for equality. It returns true if the strings contain the same characters in the same order, and false otherwise.
2. boolean **equalsIgnoreCase**(String str)
   To perform a comparison that ignores case differences.

```
// Demonstrate equals() and equalsIgnoreCase().
        class equalsDemo {
                public static void main(String args[]) {
```

```
                String s1 = "Hello";
                String s2 = "Hello";
                String s3 = "Good-bye";
                String s4 = "HELLO";
                System.out.println(s1.equals(s2));
                System.out.println(s1.equals(s3));
                System.out.println(s1.equals(s4));
                System.out.println(s1.equalsIgnoreCase(s4));
        }
    }
```

Output:

true
false
false
true

String Comparison using regionMatches Method:-

boolean **regionMatches**(int startIndex, String str2, int str2StartIndex, int numChars)
boolean **regionMatches**(boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars)

- The regionMatches( ) method compares a specific region inside a string with another specific region in another string.
- startIndex specifies the index at which the region begins within the invoking String object.
- The String being compared is specified by str2. The index at which the comparison will start within str2 is specified by str2StartIndex.
- The length of the substring being compared is passed in numChars.

Example:-

```
    class RegionTest{
        public static void main(String args[]){
            String str1 = "This is Test";
            String str2 = "THIS IS TEST";
            if(str1.regionMatches(5,str2,5,3)) {
                // Case, pos1,secdString,pos1,len
                System.out.println("Strings are Equal");
             }
            else{
                System.out.println("Strings are NOT Equal");
            }
        }
```

```
        }
```

Output:

Strings are NOT Equal

```
//Sorting of String using compareTo method.
        class SortString {
            static String arr[] = { "is", "the", "time", "for", "all", "good", "men", "to",
        "come", "to", "the", "aid", "of", "their", "country“};
            public static void main(String args[]) {
                for(int j = 0; j < arr.length; j++) {
                    for(int i = j + 1; i < arr.length; i++) {
                        if(arr[i].compareTo(arr[j]) < 0) {
                            String t = arr[j];
                            arr[j] = arr[i];
                            arr[i] = t;
                        }
                    }
                    System.out.println(arr[j]);
                }
            }
        }
```

Output:-

aid all come country for good is men of
the the their time to to

Changing the Case of Characters Within a String :-

String toLowerCase( ):- converts all the characters in a string from uppercase to lowercase.

String toUpperCase( ):- converts all the characters in a string from lowercase to uppercase

// Demonstrate toUpperCase() and toLowerCase().

```
        class ChangeCase {
            public static void main(String args[]){
                String s = "This is a test.";
                System.out.println("Original: " + s);
                String upper = s.toUpperCase();
                String lower = s.toLowerCase();
                System.out.println("Uppercase: " + upper);
                System.out.println("Lowercase: " + lower);
            }
        }
```

Output:

Original: This is a test.
Uppercase: THIS IS A TEST.
Lowercase: this is a test.

StringBuffer Class:

- StringBuffer is mutable means one can change the value of the object .
- The object created through StringBuffer is stored in the heap. StringBuffer has the same methods as the StringBuilder , but each method in StringBuffer is synchronized that is StringBuffer is thread safe . Due to this it does not allow two threads to simultaneously access the same method . Each method can be accessed by one thread at a time .
- 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 Constructors

1. StringBuffer( ): It reserves room for 16 characters without reallocation.
   StringBuffer s=new StringBuffer();
2. StringBuffer(int size)It accepts an integer argument that explicitly sets the size of the buffer.
   StringBuffer s=new StringBuffer(20);
3. StringBuffer(String str): It accepts a String argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.
   StringBuffer s=new StringBuffer("RGUKTIIIT");

Methods

1. StringBuffer append() method

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

```
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
}
}
```

2. StringBuffer insert() method

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

```
class StringBufferExample2{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
```

```
}
```

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

```
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

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

```
class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

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

```
class StringBufferExample5{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

6. StringBuffer capacity() method
   The capacity() method of 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.

```
class StringBufferExample6{
public static void main(String args[]){
```

```
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}
```

7. StringBuffer ensureCapacity() method
   The ensureCapacity() method of 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.

```
class StringBufferExample7{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
}
```

StringBuilder

- StringBuilder is same as the StringBuffer , that is it stores the object in heap and it can also be modified .
- The main difference between the StringBuffer and StringBuilder is that StringBuilder is also not thread safe. StringBuilder is fast as it is not thread safe .

Constructors:

- StringBuilder demo2= new StringBuilder("Hello");
  The above object too is stored in the heap and its value can be modified
- demo2=new StringBuilder("Bye");
  Above statement is right as it modifies the value which is allowed in the StringBuilder

String Tokenizer class:-

- The string tokenizer class allows an application to break a string into tokens.

- The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments.
- StringTokennizer class is used to split a String into different tokens as by defined delimiter.(space is the default delimiter).

Constructors:-

1. StringTokenizer(String str) :
   str is string to be tokenized. Considers default delimiters like new line, space, tab, carriage return and form feed.
2. StringTokenizer(String str, String delim) :
   delim is set of delimiters that are used to tokenize the given string.
3. StringTokenizer(String str, String delim, boolean flag):
   The first two parameters have same meaning.

   The flag serves following purpose.

   If the flag is false, delimiter characters serve to separate tokens. For example, if string is "hello RGUKT" and delimiter is " ", then tokens are "hello" and "RGUKT".

   If the flag is true, delimiter characters are considered to be tokens. For example, if string is "hello RGUKT" and delimiter is " ", then tokens are "hello", " " and "RGUKT".

Methods

- int **countTokens( )** //returns number of tokens in the string.
- boolean **hasMoreTokens( )** //checks whether tokens are there or not
- String **nextToken( )** //returns the token in the string

Example:-

```
import java.util.StringTokenizer;
public class StringTokenizer_Test {
static String str = "Hello,Welcome,to,Java,Programming";
   public static void main(String args[]) {
        StringTokenizer st = new StringTokenizer(str);
        StringTokenizer st1 = new StringTokenizer(str,",");
        StringTokenizer st2 = new StringTokenizer(str,",",true);
        while(st.hasMoreTokens()) {
             String tokens = st.nextToken();
             System.out.print(tokens + "\n");
        }
         while(st1.hasMoreTokens()) {
             String tokens = st1.nextToken();
             System.out.print(tokens + "\n");
        }
        while(st2.hasMoreTokens()) {
                  String tokens = st2.nextToken();
                  System.out.print(tokens + "\n");
        }
   }
```

```
}
```

Output:

```
Hello,Welcome,to,Java,Programming
Hello
Welcome
to
Java
Programming
Hello
,
Welcome
,
to
,
Java
,
Programming
```

Exercise1:

Write a java program that reads a line of integers and then displays each integer and find the sum of the integers (using StringTokenizer)?

```
import java.util.Scanner;
import java.util.*;
class Token{
    static int sum=0;
    public static void main(String sree[]){
            Scanner s=new Scanner(System.in);
            System.out.print("Enter sum of integers: ");
            String str=s.next();
            StringTokenizer st=new StringTokenizer(str,"+");
            while(st.hasMoreTokens()){
                    sum=sum+Integer.parseInt(st.nextToken());
            }
            System.out.println("Sum of "+str+"is: "+sum);
    }
}
```

Input:

Enter sum of integers: 10+20+30+40

Output:

Sum of 10+20+30+40 is: 100