String class: It is a class which is present under java.lang package. String class contains many methods.

There are 2 ways to define a String:

1. String s = new String (“value”);

2. String ss = “value”;

1. Define String with new Keyword: Whenever we create String with the help of new keyword then it will create one object at heap area and other at SCP area (provided if any other object by same value is not available).

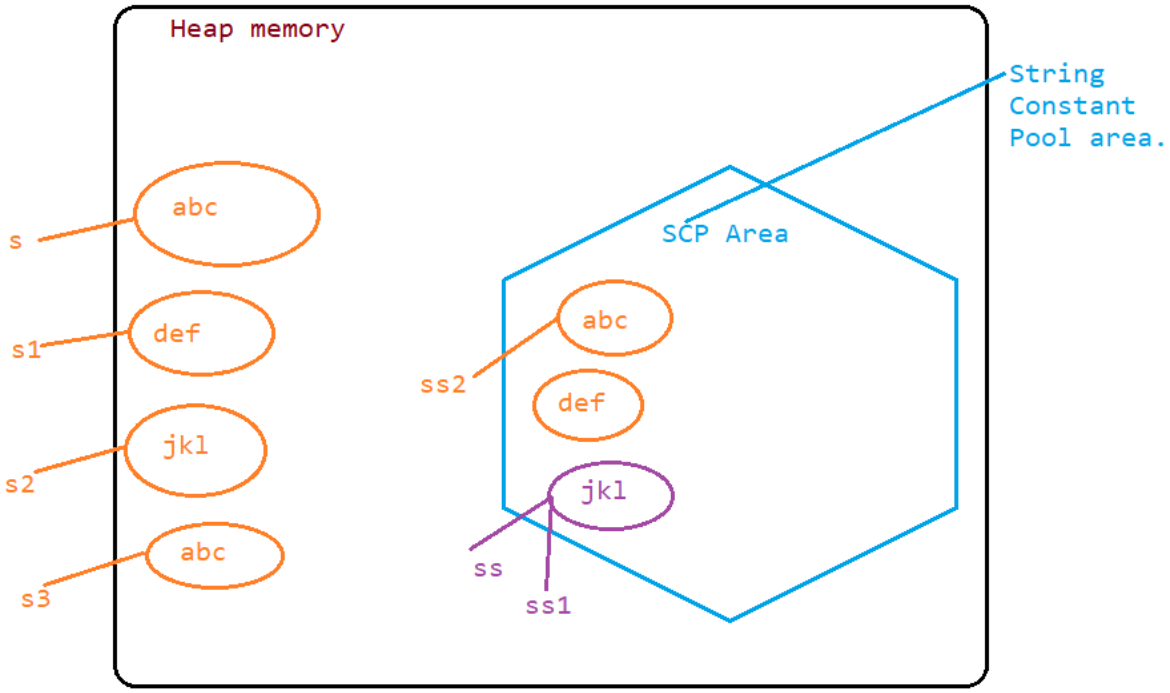
Object will always be referred by heap area.

2. Defining String without new keyword: Whenever we create a String without new keyword then it will create an object only in the SCP area(Provided any other object with same value is not available) and referred to the same object.

If same value of another String got created without new keyword then existing object inside the SCP area will get referred but it will not create a new one.

Example:

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

        String s = new String("abc");

        String s1 =  new String("def");

        String ss = "jkl";

        String ss1 = "jkl";

        String s2 =  new String("jkl");

        String s3 = new String("abc");

        String ss2 = "abc";

}
```



Note: It is always better to create the String without new keyword which doesn't cause multiple object creation.

To check whether variables are pointing to the same object or not we can use == operator, it will return true if they are pointing to the same object but false if they are not.

```
boolean ispointingsame = ss==ss1;
System.out.println(ispointingsame);//true


boolean ispointsameobj = ss==s2;
```

```
System.out.println(ispointsameobj);// false
```

Difference between == and equals method:

== is used to check whether two strings are referring to the same object or not whereas equals method is used for content comparison.

Example of equals method:

```
boolean isequal = s3.equals(s);

System.out.println(isequal);//true


boolean isequal2 = ss.equals(s2);
System.out.println(isequal2);// true
```

Note:

String class is immutable in nature which means after performing any operation on the same string the original string will not get change.

To change the original String we have to assign the new value to the original String then only we will be able to change the original String value.

But other class like StringBuffer is mutable as if we

perform any operation to the String then original string get change which makes it mutable in nature.

```
String var1 = "Velocity";


var1.concat(" Corporate");

System.out.println("Output of String is "+var1);// Velocity--> Immutable


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

StringBuffer sb = new StringBuffer("Velocity");

sb.append(" Corporate");

System.out.println("Output of String Buffer is "+sb);// Velocity Corporate ---> Mutable
```

Methods in String class:

1. equals(String s): This method returns true if both the string are exactly identical to eachother otherwise false.

Example:

1. equals(String s)

```
String s1 = "Velocity";
```

```
        String s2 = "velocity";

    boolean s3 = s1.equals(s2);

    System.out.println(s3);//false
```

2. equalsIgnoreCase(String s): This method returns true if the content of both the strings are equal irrespective of their case (can be either lower / upper case) otherwise false.

Example:

```
    //    2. equalsIgnoreCase(String s)

    String s4 = "Velocity";
    String s5 = "velOcity";

    boolean s6 = s4.equalsIgnoreCase(s5);

    System.out.println(s6);//true
```

3. length(): This method returns the total number of characters available inside a String .

Example:

```
//    3. length();

    String s7 = "abcdefghi";

    int s8 = s7.length();

    System.out.println(s8);//9
```

4. charAt(int index): This method returns character value based on the index position which we are providing in it of a particular character.

Example:

```
//      4. charAt(int index);

        String s9 = "abcdef";

        char s10 = s9.charAt(1);

        System.out.println(s10);//b
```

Note: If we provide the index position which is beyond the index limit then we will get StringIndexOutOfBoundException.

Example:

```
//      4. charAt(int index);

        String s9 = "abcdef";

        char s10 = s9.charAt(10);

        System.out.println(s10);
```

Output:

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 10

Assignment:

```
//	WAP to print reverse of a String value

//	WAP to print the horizontal String to the vertical individual character

	//	WAP to print reverse of a String value

	String s11 = "This is String";

	String rev = "";

	int size = s11.length();

	int lastindex = size-1;

	for (int i=lastindex; i>=0; i--)
	{
		rev = rev + s11.charAt(i);
	}

	System.out.println(rev);



//	WAP to print the horizontal String to the vertical individual character

	String s12 = "Velocityhkjhfdkjshfkjsdfhkjsdfhksdjfhsdkfhsdkjfhsdkjfk";
```

```
for(int i=0; i<s12.length(); i++  )
{
	char letter = s12.charAt(i);

	System.out.println(letter);
}
```

5. replace(char old, char new): This method replace the old char with new character that we will provide as an input.
Example:

```
//	5. replace(char old, char new)

	String s13 = "abcdef";

	String s14 = s13.replace('b', 'f');

	System.out.println(s14);//afcdef
```

6. replace(String old, String new): This method returns a replaced String value which we are providing as an new argument.

```
//	6. replace(String old, String new)

	String s15 = "Constructor";

	String s16 = s15.replace("truc", "tor");

	System.out.println(s16);//Constortor
```

7. substring(int index): This method returns a String value which is starting from the provided index value.

```
//	7. substring(int index):
```

```
        String s17 = "Velocity";

        String s18 = s17.substring(3);

        System.out.println(s18);//ocity
```

8. substring(int beginindex, int endindex): This returns a string value which starts from the beginindex position to the endindex position-1.
Example:

```
//      8. substring(int beginindex, int endindex)

        String s19 = "Corporate";

        String s20 = s19.substring(2, 7);

        System.out.println(s20);//rpora
```

9. contains(String value): This method returns the true if the sequence we have provided is matched inside the String otherwise it will return false.
Example:

```
//      9. contains(String value)

        String s21 = "abcdefghijklk";

        boolean s22 = s21.contains("lk");

        System.out.println(s22);//true
```

10. toLowerCase(): This method returns a String which is actually a String with all the characters in lowercase.

```
//      10. toLowerCase()
```

```
String s23 = "VELOciTy";

String s24 = s23.toLowerCase();

System.out.println(s24);//velocity
```

11. toUpperCase(): This method returns a String which is actually a String with all the characters in Uppercase.

```
//      11. toUpperCase()

String s25 = s24.toUpperCase();

System.out.println(s25); //VELOCITY
```

12. indexOf(char ch): This method returns an index position of the character that we have provided from the string value.
Example:

```
//      12. indexOf(char ch)

String s26= "abcdefabc";

int s27 = s26.indexOf('b');

System.out.println(s27);//1
```

WAP to count a character inside a String:

```
//      WAP to count a character in a String


String s28 = "aabbaaccd";
int count =0;

for(int i=0; i<s28.length(); i++)
```

```
        {
                char c = s28.charAt(i);

                if(c=='a')
                {
                        count++;
                }

        }

        System.out.println("Character a is present for "+count+" times");
```

13. trim(): This method is used to remove all the spaces associated with the String at starting and trailing point.

```
//      13. trim()

        String s29 = "     Happy new year     ";

        String s30 = s29.trim();
        System.out.println(s30);//Happy new year
```

14. split(String s): This method returns an String array which has the elements based on the value we have splitted the string into and store as an elements of an array.

Example:

```
// 14. split(String s)

        String s32 = "This is String";
```

```
        String[] s33 = s32.split("i");

        for(String ss:s33)
        {
                System.out.println(ss);
        }
```

Output:

```
Th
s
s Str
ng
```

```
//      WAP to print  the reverse string without reversing the letters but the words
//      ex. This is String ----> String is This
```

15. toCharArray(): This method returns a char array of the given String.

```
//      15. toCharArray():

        String s34 = "String Value #123";

        char[] chararray = s34.toCharArray();

        int sizee = chararray.length;
        for(int ii= 0; ii<sizee; ii++)
        {
                System.out.println(chararray[ii]);
        }
```

Output:

S
t
r
i
n
g

V
a
l
u
e

#
1
2
3

Conversion of Primitive type to String data type:

16. valueOf(primitivedatatypevalue): This method returns a String value based on the value that we are providing inside argument(primitive data type ex. int, char, boolean).

Example:

```
//      16. ValueOf(primitive data type)

        int val = 100;
        String stringvalue = String.valueOf(val);

        System.out.println(stringvalue+55);//10055
```

Output:

10055

Example 2:

```
boolean bol = false;

String s35 = String.valueOf(bol);

System.out.println(s35);//false
```

Conversion of String type to Primitive data type:

To convert from String to primitive data type we have to use individual Wrapper class of primitive data type.

Wrapper class: It is a class which represents a primitive value as an object.

To convert String to primitive data type we have to use parseXxx();

Example 1:

```
String s36 = "10";

int s37 =Integer.parseInt(s36);

int addedvalue = s37+2;
System.out.println(addedvalue);//12
```

Example 2:

```
String s39 = "56.23";

        double s40 = Double.parseDouble(s39);

        System.out.println(s40);//56.23
```

Example 3:

```
String s41 = "true";

        boolean s42 = Boolean.parseBoolean(s41);

        System.out.println(s42);//true
```

18. isDigit(char ch): This method returns true if the provided character is a digit otherwise it returns false.

```
//      18. isDigit(char c)

        char s43 = '8';

        boolean s44 = Character.isDigit(s43);

        System.out.println(s44);//true
```

19. isAlphabetic(char ch): This method returns true if the provided character is an alphabet otherwise it returns false.

```
//      19. isAlphabetic(char ch);

        char s45 = 'g';
```

```
boolean s46 = Character.isAlphabetic(s45);

System.out.println(s46);//true
```

20. replaceAll(String s): In this method we take String as an argument which follows the set of rules.

Example:

```
//      20 replaceAll(String str):

        String s48 = "A@@c3c4e32%#nt324ure";

        String s49 = s48.replaceAll("[a-z]", ".");

        System.out.println(s49);//A@@.34.32%#..324...

        String s51 = s48.replaceAll("[a-c]", ".");

        System.out.println(s51);//A@@.34e32%#nt324ure

        String s52 = s48.replaceAll("[A-Za-z]", "0");

        System.out.println(s52);//0@@034032%#00324000

        String s53 = s48.replaceAll("[^a-z]", "1");
        System.out.println(s53);//111c11e1111nt111ure

        String s54 = s48.replaceAll("[^a-zA-Z]", "");
        System.out.println(s54);//Accenture


        String s55 = s48.replaceAll("[0-9]", "k");
```

```
System.out.println(s55);//A@@ckckekk%#ntkkkure
```