String
Vs
StringBuffer

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

---

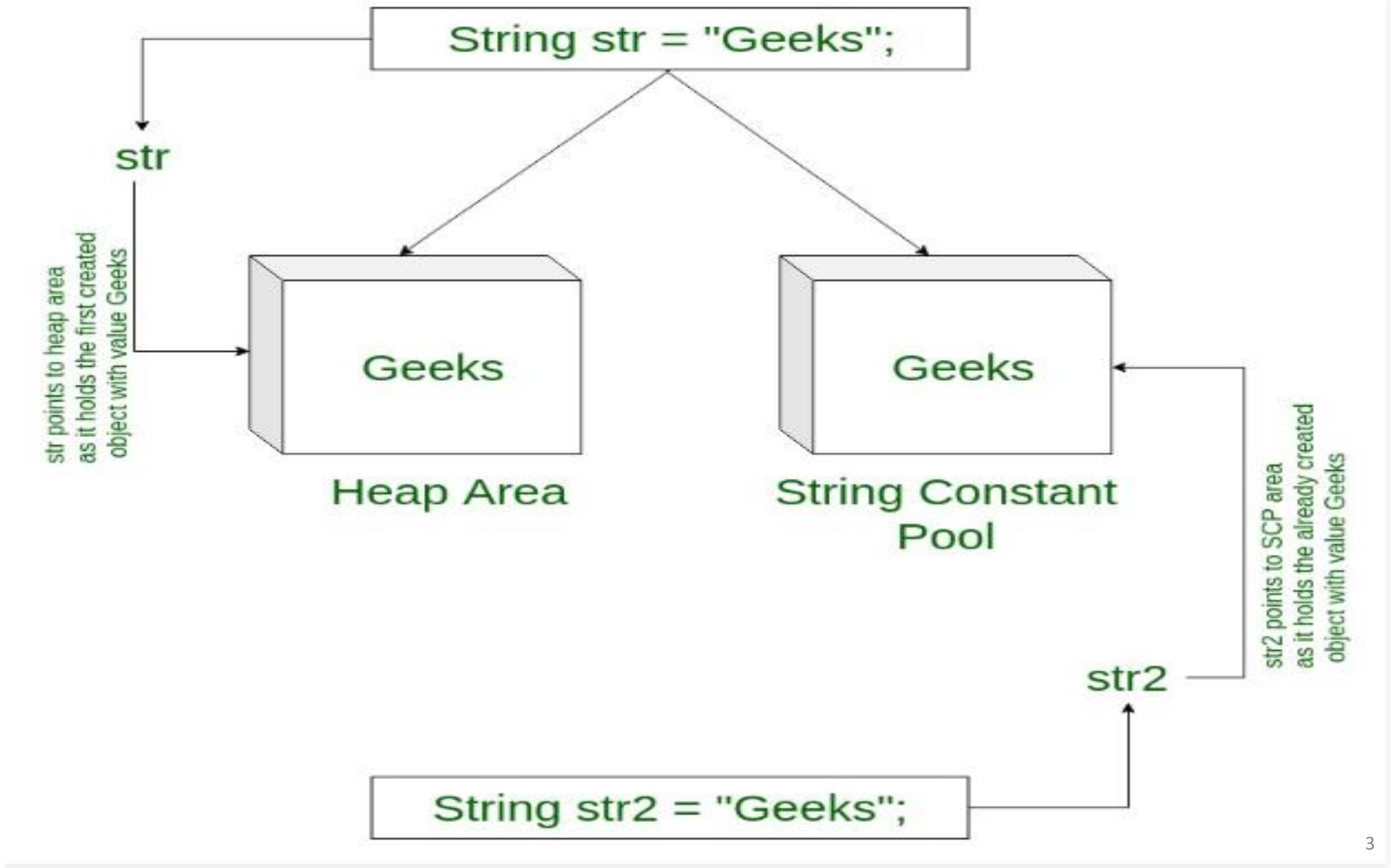
1) String Literal

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

```
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
```


String str = "Geeks";
str
str points to heap area as it holds the first created object with value Geeks
Geeks
Heap Area
Geeks
String Constant Pool
str2 points to SCP area as it holds the already created object with value Geeks
str2
String str2 = "Geeks";

2) By new keyword

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

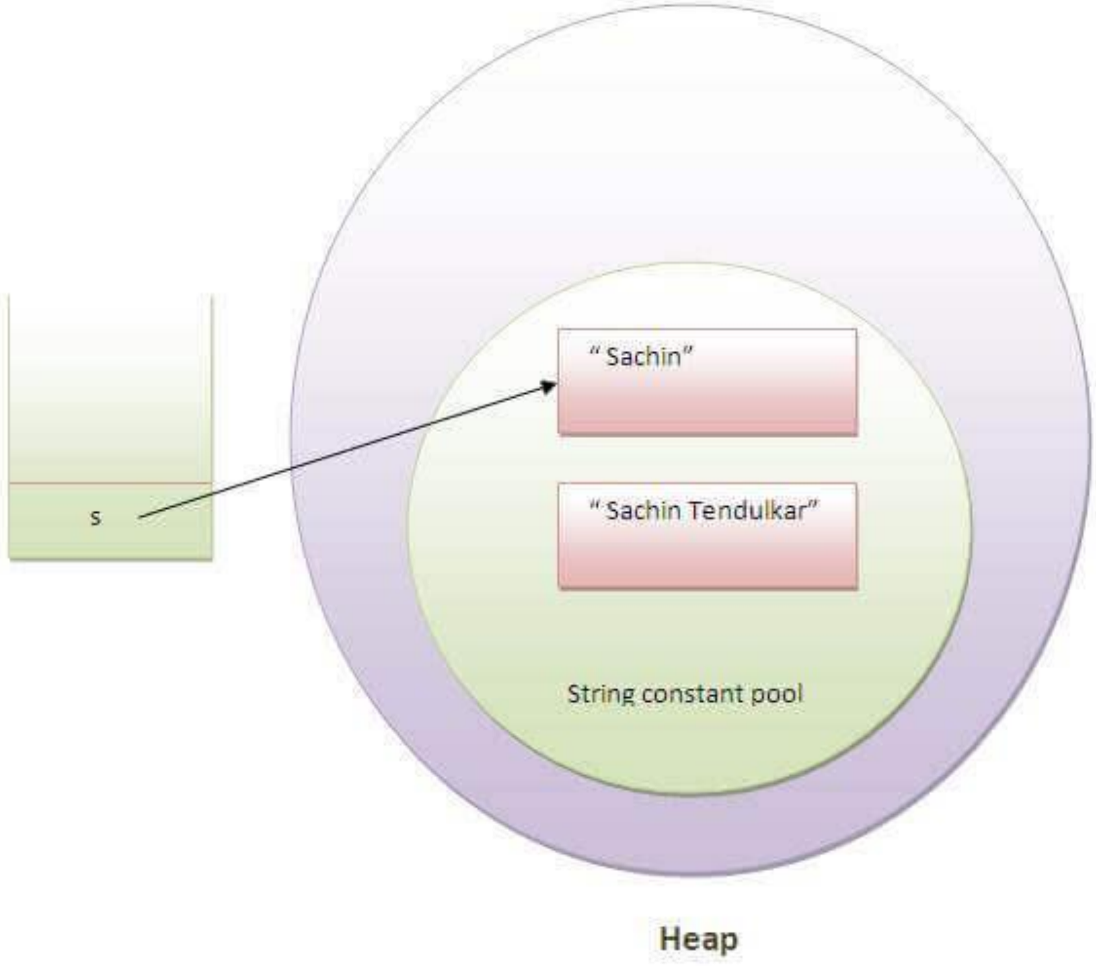
In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Java String Example

```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);
}}
```

Test it Now

```
java
strings
example
```


s
" Sachin"
" Sachin Tendulkar"
String constant pool
Heap

Introduction

- A chain of characters is called a string.
- In Java, each character in a string is a 16-bit Unicode character.
- Because Unicode characters are 16 bits (not the 7 or 8 bits that ASCII provides), a rich, international set of characters is easily represented in Unicode.
- In Java, strings are objects.
- Java offers a final class ***java.lang.String***, which handles strings.

The String Constructors

- The String class supports several constructors. To create an empty string, we can call the default constructor. For example:

```
String s=new String();
```

- To create a string initialized by an array of characters, we can use the following constructor.

```
String s=new String(char characters[]);
```

For Example:

```
char chars[]={'H', 'e' , 'l' , 'l' , 'o'};
String s=new String(chars);
System.out.println("String is : "+s);
```

The String Constructors

- We can specify a sub range of a character array as an initializer using the following:

```
String s=new String(char chars[], int startIndex, int numChars);
```

For Example:

```
char chars[]={'H', 'e' , 'l' , 'l' , 'o',' ', 'J' , 'a' , 'v' , 'a','!'};
String s2=new String(chars,4,4);
System.out.println("String is : "+s2);
```

- We can construct a String object that contains the same character sequence as another String object using the following constructor

```
String s=new String(String strObj);
```

For Example:

```
char chars[]={'H', 'e' , 'l' , 'l' , 'o',' ', 'J' , 'a' , 'v' , 'a','!'};
String s1=new String(chars);
System.out.println("String is : "+s1);
String s2=new String(s1);
System.out.println("String is : "+s2);
```

The String Constructors

- String class also provides constructors that initialize a string when given a byte array. Their forms are as follows:

```
String s=new String(byte asciiChars[]);
String s=new String(byte asciiChars[], int startIndex, int numChars);
```

For Example:

```
byte ascChars[]={65, 66 , 67 , 68 , 69, 70};

String s1=new String(ascChars);
System.out.println("String is : "+s1);

String s2=new String(ascChars,2,3);
System.out.println("String is : "+s2);
```

Methods of the String Class

- The String class offers a wide spectrum of methods to handle strings in an application. We can use these methods to perform operations on a string, including the following:
  - Concatenate a string to another string
  - Examine individual characters of the string
  - Compare strings
  - Search for a character or substrings in a string
  - Extract a substring from a string
  - Create a copy of a string with all characters uppercased
  - Create a copy of a string with all characters lowercased

public char charAt(int index)

- This method returns the character located at the String's specified index. String indexes are zero-based.
- For example:

```
String x = "airplane";
System.out.println( x.charAt(2) );      //  output is 'r'
```

public String concat(String s)

- This method returns a String with the value of the String passed in to the method appended to the end of the String used to invoke the method.

For example:

```
    String x = "Hello";
    System.out.println( x.concat(" Java!") );
```

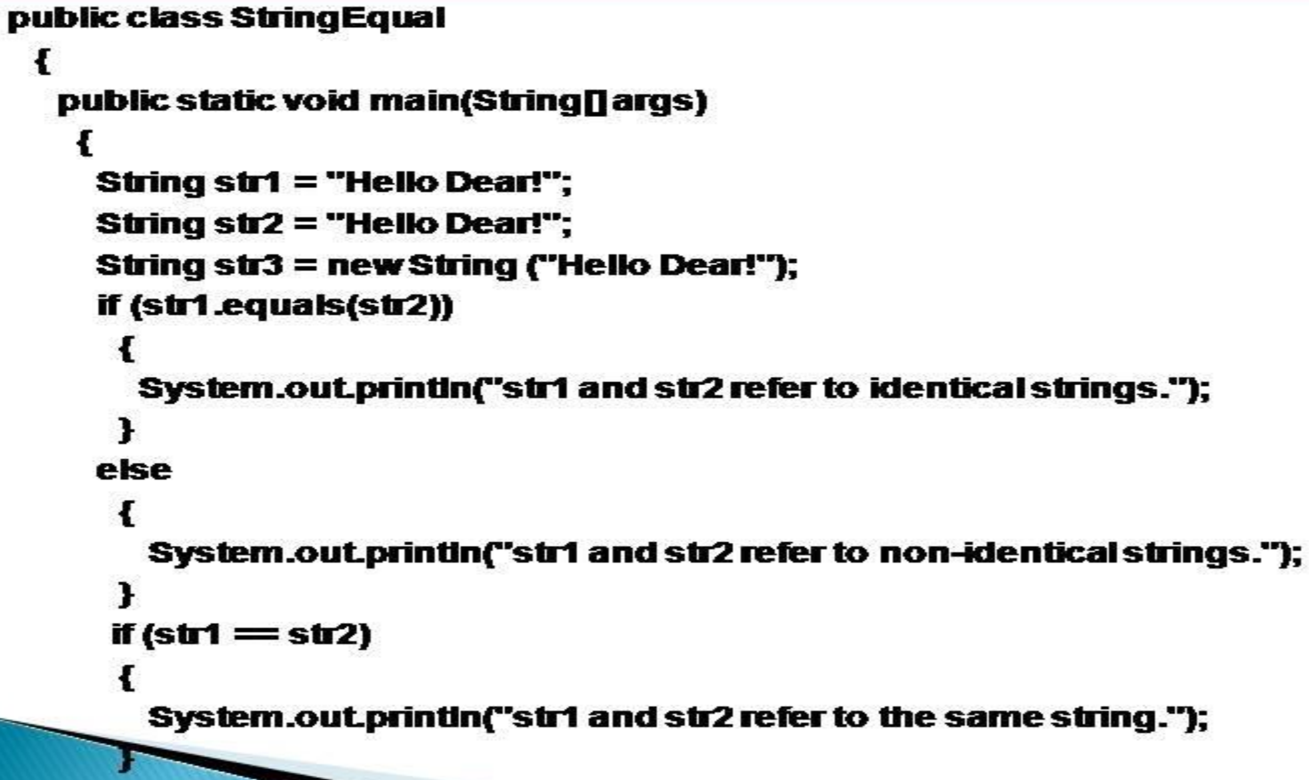
// output is “Hello Java”

public boolean equals(String s)

- This method returns a boolean value (true or false) depending on whether the value of the String in the argument is the same as the value of the String used to invoke the method.
- This method will compare the characters in the Strings casewise.
- The equals(...) method is inherited from the Object class and is overridden to perform the deep comparison—that is, it compares the characters of the two strings.
- It returns true if the two strings contain an identical chain of characters.
- The shallow comparison, which is done in the Object class version of the equals(...) method, simply uses the == operator. It returns true if the two object references are equal—that is, they refer to the same String object.

Continued

public boolean equals(String s)



```
public class StringEqual
  {
    public static void main(String[] args)
      {
        String str1 = "Hello Dear!";
        String str2 = "Hello Dear!";
        String str3 = new String ("Hello Dear!");
        if (str1.equals(str2))
          {
            System.out.println("str1 and str2 refer to identical strings.");
          }
        else
          {
             System.out.println("str1 and str2 refer to non-identical strings.");
          }
         if (str1 == str2)
          {
             System.out.println("str1 and str2 refer to the same string.");
          }
```

public boolean equals(String s)

```
 else
  {
        System.out.println("str1 and str2 refer to different strings.");
  }
 if (str1.equals(str3))
  {
        System.out.println("str1 and str3 refer to identical strings.");
  }
 else
  {
    System.out.println("str1 and str3 refer to non-identical strings.");
  }
if (str1 == str3)
  {
    System.out.println("str1 and str3 refer to the same string.");
  }
else
 {
        System.out.println("str1 and str3 refer to different strings.");
 }}}
```

public boolean equalsIgnoreCase(String s)

- This method performs exactly the same function as the equals() method except that it performs a comparison of the characters in the strings ignoring the case.
- For example,

```
String x = "Exit";
System.out.println( x.equalsIgnoreCase("EXIT")); //TRUE
System.out.println(x.equalsIgnoreCase("tixe");    //FALSE
```

public int length()

- This method returns the length of the String used to invoke the method.
- For example:

```
String x = "01234567";
System.out.println( x.length() );     // returns "8"
```

public String replace(char old, char new)

- This method returns a String whose value is that of the String used to invoke the method, updated so that any occurrence of the char in the first argument is replaced by the char in the second argument.
- For example:

```
String x = "oxoxoxox";
System.out.println( x.replace('x', 'X') );

// output is "oXoXoXoX"
```

public String toLowerCase()

- This method returns a String whose value is the String used to invoke the method, but with any uppercase characters converted to lowercase.
- For example:

```
String x = "A New Moon";
System.out.println( x.toLowerCase());


 // output is "a new moon"
```

public String toUpperCase()

- This method returns a String whose value is the String used to invoke the method, but with any lowercase characters converted to uppercase.
- For example:

```
String x = "A New Moon";
System.out.println( x.toUpperCase());


 // output is "A NEW MOON"
```

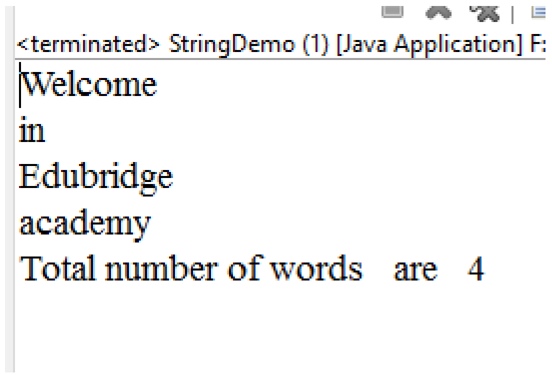
public String trim()

- This method returns a String whose value is the String used to invoke the method, but with any leading or trailing blank spaces removed.
- For example:

```
String x = " hi ";

System.out.println( x + "x" );
     // result is " hi x"

System.out.println( x.trim() + "x");
     // result is "hix"
```

Split():

```
import java.util.Arrays;

public class StringDemo
{

        public static void main(String[] args)
        {
        // TODO Auto-generated method stub



        String   s ="Welcome in Edubridge academy";
        String str[]= s.split(" ");

        for(int i=0;i<str.length;i++)
        {
        System.out.println(str[i]);
        }
}
```



```
<terminated> StringDemo (1) [Java Application] F:
Welcome
in
Edubridge
academy
Total number of words   are   4
```

boolean endsWith(String suffix)

- This method returns true if the current string ends with suffix, else returns false.
- For example:

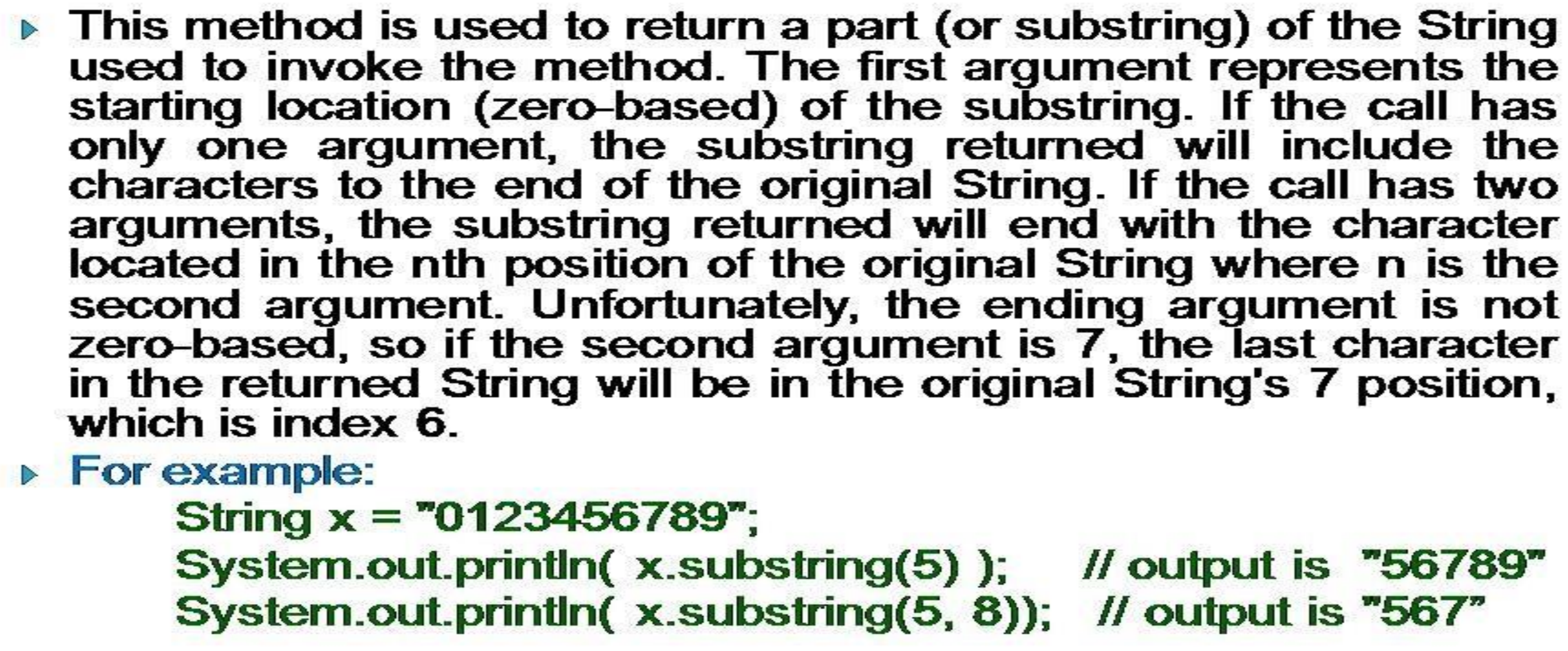
```
String x = "Foobar";
System.out.println( x.endsWith("bar"));    // output is true
```

boolean startsWith(String prefix)

- This method returns true if this string starts with the specified prefix.
- For example:

```
String x = "Foobar";
System.out.println( x.startsWith("Foo"));    // output is true
```

public String substring(int begin), public String substring(int begin, int end)



- This method is used to return a part (or substring) of the String used to invoke the method. The first argument represents the starting location (zero-based) of the substring. If the call has only one argument, the substring returned will include the characters to the end of the original String. If the call has two arguments, the substring returned will end with the character located in the nth position of the original String where n is the second argument. Unfortunately, the ending argument is not zero-based, so if the second argument is 7, the last character in the returned String will be in the original String's 7 position, which is index 6.
- For example:

```
String x = "0123456789";
System.out.println( x.substring(5) );    // output is  "56789"
System.out.println( x.substring(5, 8));   // output is "567"
```

int compareTo(String str)

- This method makes a lexical comparison. Returns a negative integer if the current string is less than str, 0 if the two strings are identical, and an integer greater than zero if the current string is greater than str. For example:

```
class SortString
{
     static String arr[] = {"Now", "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]);
               }
     }
}
```

int indexOf(int ch)

- This method returns the index within the string of the first appearance of ch.
- For example:

```
String s = "Sun is shining brightly";
System.out.println("indexOf(s) = " + s.indexOf('s'));


        //output is 5
```

int lastIndexOf(int ch)

- This method returns the index within the string of the last appearance of ch.
- For example:

```
String s = "Sun is shining brightly";
System.out.println("lastIndexOf(s) = " +  s.lastIndexOf('s')
```

//output is 7

Java String compare

1. **By equals() method**

2. **By = = operator**

3. **By compareTo() method**

equals() method

```
class Teststringcomparison1
{
   public static void main(String args[])
   {
        String s1="Sachin";
        String s2="Sachin";
        String s3=new String("Sachin");
        String s4="Saurav";
        System.out.println(s1.equals(s2));
        System.out.println(s1.equals(s3));
        System.out.println(s1.equals(s4));
    }
}
```

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

String compare by == operator

```
class Teststringcomparison3
{
   public static void main(String args[])
   {
      String s1="Sachin";
      String s2="Sachin";
      String s3=new String("Sachin");
      System.out.println(s1==s2);//true (because both refer to same instance)
      System.out.println(s1==s3);//false(because s3 refers to instance created in non pool)
    }
}
```

String compare by compareTo()

The String **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 variables. If:

s1 == s2 :0
s1 > s2   :positive value
s1 < s2   :negative value

```
class Teststringcomparison4
{
   public static void main(String args[])
   {
         String s1="Sachin";
         String s2="Sachin";
         String s3="Ratan";
         System.out.println(s1.compareTo(s2));    //   0
         System.out.println(s1.compareTo(s3));   //    1(because s1>s3)
         System.out.println(s3.compareTo(s1));   //   -1(because s3 < s1 )
   }
}
```

```
public class StringDemo2
{
public static void main(String[] args)
{
	String s="Aman has scored 80 marks";
	int digitCount=0;
	int alphaCount=0;
	for(int i=0;i<s.length();i++)
	{
		char ch= s.charAt(i);
		if(Character.isDigit(ch))
		{
				digitCount++;
		}
		if(Character.isAlphabetic(ch))
		{
				alphaCount++;
		}
	}
	System.out.println("Total number of numeric values  "+digitCount);
	System.out.println("Total number of numeric values  "+alphaCount);
}

}
```

StringBuffer

1. **StringBuffer** is mutuable. (Mutable means one can change the value of the object.

2. The object created through StringBuffer is stored in the heap.

3. StringBuffer has the same method as the **StringBuilder**, but each method in StringBuffer is **synchronised** that is **StringBuffer is thread safe**.

4. But being thread safe has disadvantages too as the performance of the **StringBuffer** hits due to thread safe property. Thus **StringBuilder** is faster than **StringBuffer** when calling the same method of each class.

5. StringBuffer can be converted to String by using **toString()** method.

StringBuffer

```
StringBuffer s = new StringBuffer(“Hello World”);

// The above object stored in heap and its value can be changed.

s.append(“Java Programming”);

//It modifies the value which allowed in StringBuffer
```

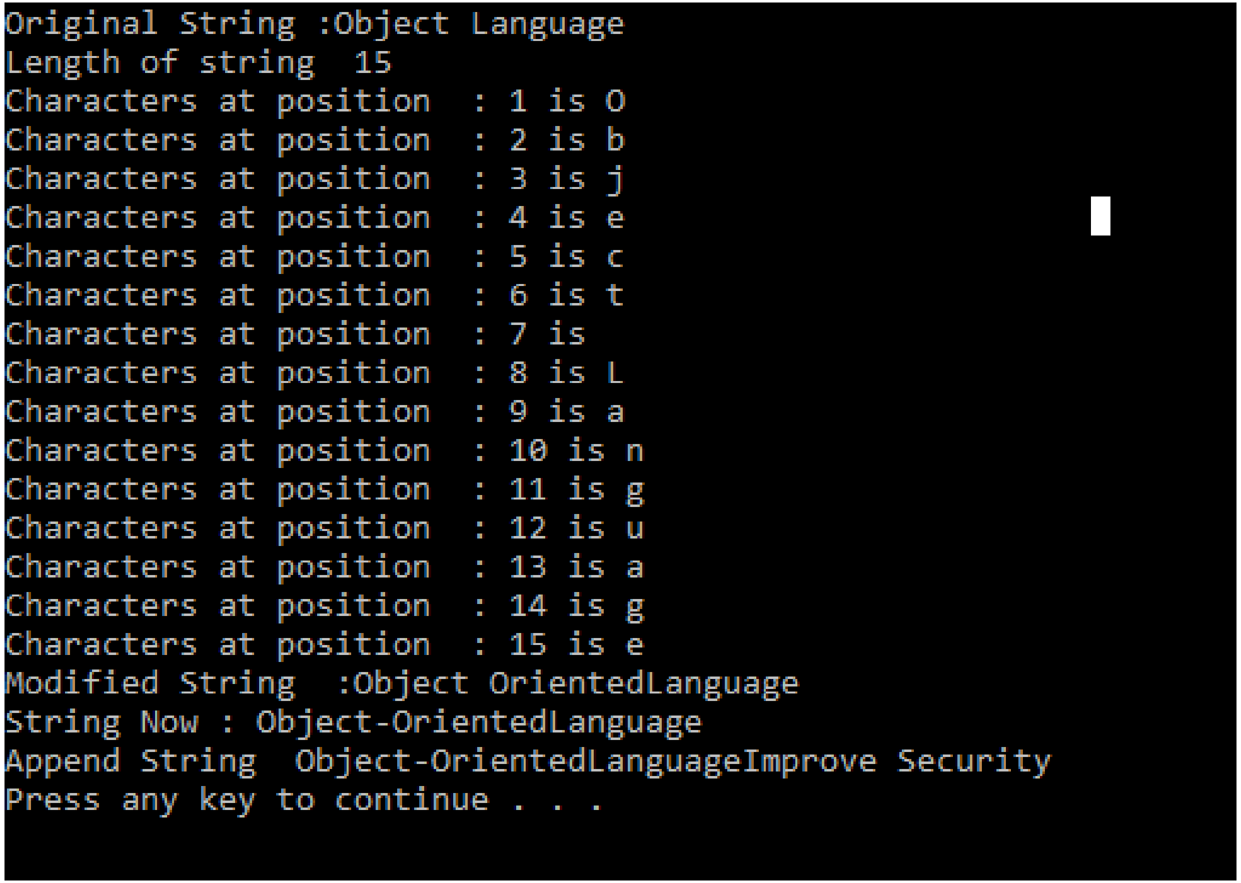
StringBuffer

```
Class  StringDemo
{
        public static void main(String args[])
        {
            StringBuffer s = new StringBuffer(“Hello World”);
            s.append(“Java Programmig”);
            System.out.println(s);
        }
}
```

HEAP
Hello World

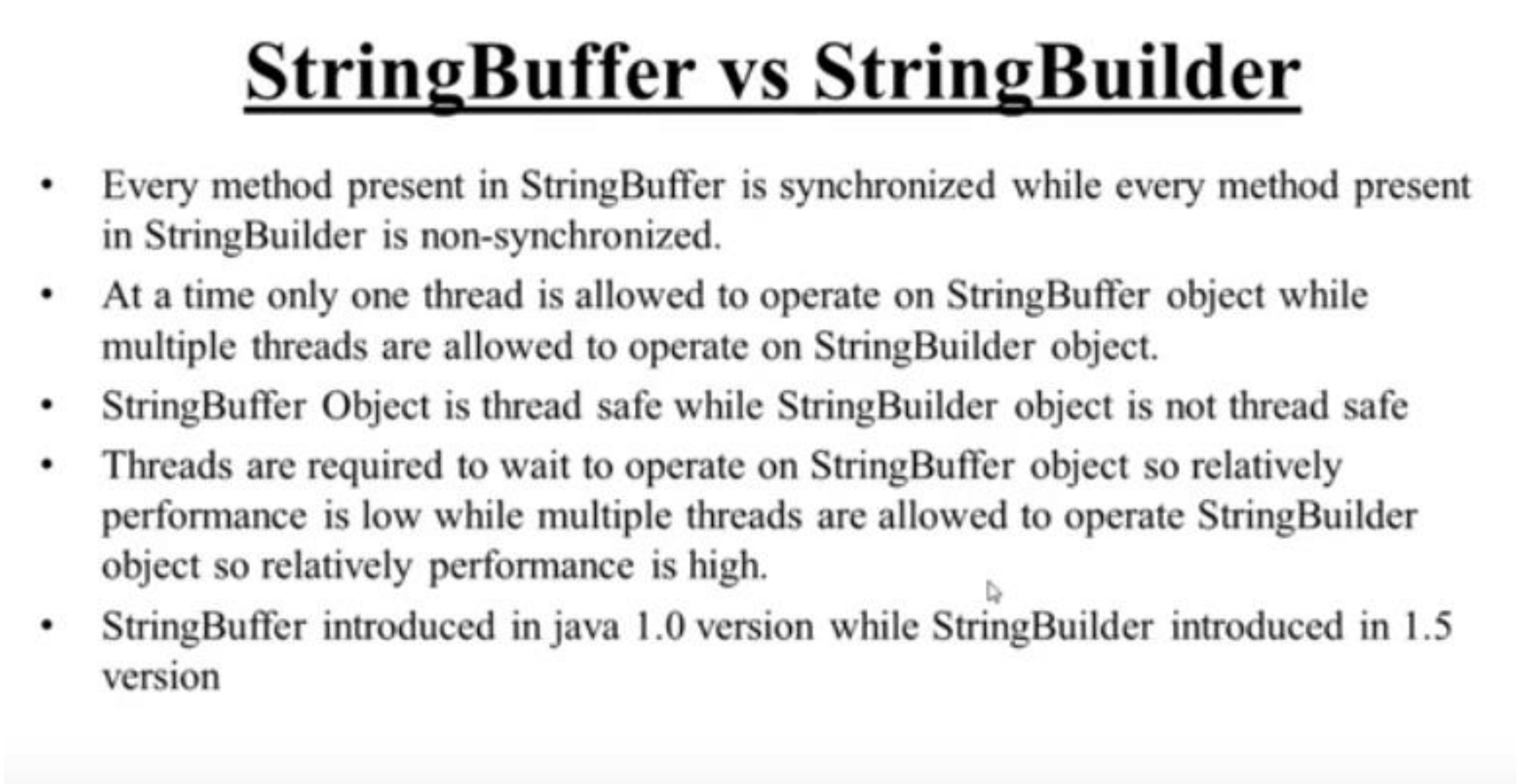
HEAP
Hello World Java Programming

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

                StringBuffer str = new StringBuffer("Object Language");

                System.out.println("Original String :"+str);

                // Obtaining string Length

                System.out.println("Length of string  " +str.length());

                //Accessing character in string

                for(int i=0;i<str.length();i++)
                {

                        int p = i+1;

                    System.out.println("Characters at position  : "+p+" is "+str.charAt(i));
                 }
                 // inserting a string in the middle

                 String str2 = new String(str.toString());
                 int pos = str2.indexOf("Language");

                 str.insert(pos,"Oriented");

                 System.out.println("Modified String  :"+str);

                 // Modifying Character

                 str.setCharAt(6,'-');

                 System.out.println("String Now : "+str);

                 // Appending a string at the end

                 str.append("Improve Security ");
                 System.out.println("Append String  "+str);


        }
```



```
Original String :Object Language
Length of string  15
Characters at position  : 1 is O
Characters at position  : 2 is b
Characters at position  : 3 is j
Characters at position  : 4 is e
Characters at position  : 5 is c
Characters at position  : 6 is t
Characters at position  : 7 is
Characters at position  : 8 is L
Characters at position  : 9 is a
Characters at position  : 10 is n
Characters at position  : 11 is g
Characters at position  : 12 is u
Characters at position  : 13 is a
Characters at position  : 14 is g
Characters at position  : 15 is e
Modified String  :Object OrientedLanguage
String Now : Object-OrientedLanguage
Append String  Object-OrientedLanguageImprove Security
Press any key to continue . . .
```

String	StringBuffer
1. String is Immutable	1. Stringbuffer is mutable
2. String is slow in speed.	2. StringBuffer is fast.
3. Consumes more memory when you concat more strings.	3. Consumes less memory
4. Override equals() method of object class.	4. Does not override equals method of object class.



StringBuffer vs StringBuilder

- Every method present in StringBuffer is synchronized while every method present in StringBuilder is non-synchronized.
- At a time only one thread is allowed to operate on StringBuffer object while multiple threads are allowed to operate on StringBuilder object.
- StringBuffer Object is thread safe while StringBuilder object is not thread safe
- Threads are required to wait to operate on StringBuffer object so relatively performance is low while multiple threads are allowed to operate StringBuilder object so relatively performance is high.
- StringBuffer introduced in java 1.0 version while StringBuilder introduced in 1.5 version

S.No	StringBuffer	StringBuilder
1.	Every method present in StringBuffer is always synchronized.	No method present in StringBuilder is synchronized.
2.	At a time Only one thread is allowed to operate on StringBuffer object ,Hence StringBuffer object is Thread safe.	At a time Multiple thread are allowed to operate on StringBuilder object ,and Hence StringBuilder object is not Thread safe.
3.	It increases waiting time of the threads and hence Relatively performances is low.	Threads are not required to wait to operate on StringBuilder object and hence Relatively performances is High.
4.	It is introduced in **1.0v**	It is introduced in **1.5v**

5.	`class StringBufferExample`
	`{`
	`public static void main(String args[])`
	`{`
	`StringBuffer sb=new StringBuffer("Govind");`
	`        sb.append(" Ballabh Khan");`
	`        System.out.println(sb);`
	`        }`
	`        }`
	
	o/p= **Govind Ballabh Khan**
		`class StringBuilderExample`
		`{`
		`public static void main(String args[])`
		`{`
		`StringBuilder sb=new  StringBuilder("Govind");`
		`        sb.append(" Ballabh Khan");`
		`        System.out.println(sb);`
		`        }`
		`        }`
		
		o/p= **Govind Ballabh Khan**