String

String is a predefined class, which can be used for storing a group of characters.

To store a group of characters we need to create an object of String class. The String class object can be created in two ways.

1. The String class object can be created by using new operator
   String str = new String("hello");
2. The String class object can be created by specifying a group of characters enclosed directly in a pair of double quotes(" ").
   String str = "hello";

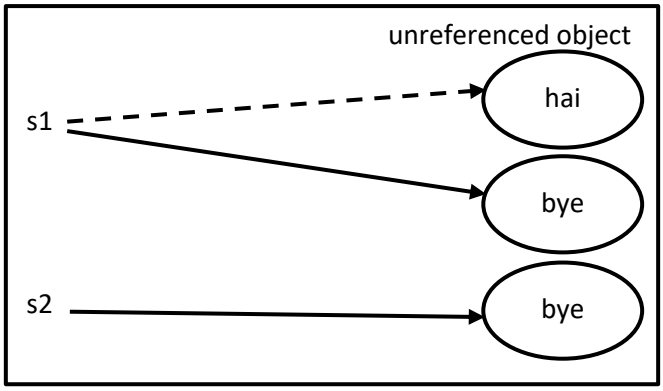
The String objects created by both the mechanisms are called as immutable object, which means once the String object is created we cannot modify the content of the object.

Difference between the two mechanisms of creating String objects:

String Object Created by using new Operator:

1) **Location:** The String objects created by using new operator are stored in heap memory.
2) **Allocation:** If the String objects are stored in heap memory then, it will never verify, whether the heap memory contains objects with same content or not, it will always create new object and store the content.

```
String s1 = new String("hai");
s1 = new String("bye");
String s2 = new String("bye");
```



heap

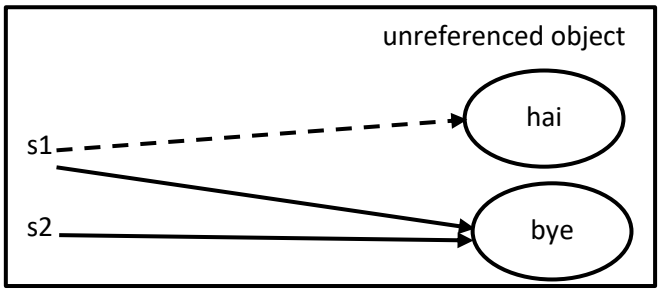
Unreferenced Object: If an object does not have any live reference i.e. if no reference variable is referring to the object then, the object should be called as unreferenced object.

3) **Deallocation:** If the heap memory contains unreferenced objects then, they will be deallocated by garbage collector.

String object created by specifying a group of characters enclosed directly in a pair of double quotes:

1) **Location:** If a String object is created by enclosing a group of characters in a pair of double quotes (" ") then, they are stored in string constant pool.
2) **Allocation:** If the String objects are stored in string constant pool then, it will always verify whether the string constant pool contain objects with same content or not if available then, refer to existing object. If not available then create a new object.

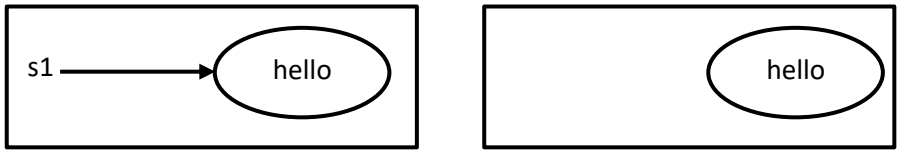
```
String s1 = "hai";
s1 = "bye";
String s2 = "bye";
```



string constant pool

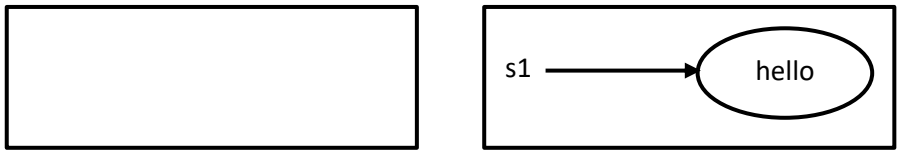
3) **Deallocation:** If the string constant pool contains unreferenced object then, they will be deallocated by the string constant pool itself, when the string constant pool is completely filled.

If a String object is created by using new operator then the content will be stored in both heap and string constant pool but the reference will refer to the object available in heap memory.
String s1 = new String("hello");



heap string constant pool

If a String object is created by specifying a group of characters enclosed directly in a pair of double quotes then the content will be stored in only in string constant pool.
String s1 = "hello";



heap string constant pool

Method Of String Class:

1) **int length():** This method will return the count of the number of characters available in a String.

   Program:

```
class StringDemo  {
    public static void main(String[] args)  {
        String str = new String("java program");
        System.out.println (str.length());
    }
}
```

2) **char charAt(int index):** This method will return a character that is available at the specified index position. The index position starts from 0. If the specified index is not within the range then, it will generate a runtime error called StringIndexOutOfBoundsException.
   Example:

```
String str = new String("java program");
System.out.println(str.charAt(5));
System.out.println(str.charAt(15));
```

3) **String concat(String):** This method can be used to append the contents of one string to another string.
   Example:

```
String s1 = new String("java");
String s2 = new String("program");
s1= s1.concat (s2);
System.out.println(s1);
System.out.println(s2 );
```

4) **int compareTo(String):** This method can be used to compare the unicode values of the characters available in a String, by considering their case. This method is designed for sorting the strings.
   S1 > S2 +ve
   S1 < S2 -ve
   S1 == S2 0

5) **int compareToIgnoreCase(String):** This method can be used to compare the unicode values of the characters available in a String by ignoring their case. This method is designed for sorting the strings.
   Example:

```
String s1 = new String("abc");
```

```
String s2 = new String("bcd");
System.out.println(s1.compareTo(s2));
System.out.println(s1.compareToIgnoreCase(s2));
```

6) **boolean equals(Object):** This method can be used to compare the contents of the string objects by considering their case.
7) **boolean equalsIgnoreCase(String):** This method can be used to compare the content of the string objects by ignoring their case.

Example:

```
String s1 = new String("abc");
String s2 = new String("ABC");
System.out.println(s1.equals(s2));
System.out.println(s1.equalsIgnoreCase(s2));
```

8) **boolean startsWith(String):** This method can be used to check whether a string begins with a specified group of characters or not.
9) **boolean endsWith(String):** This method can be used to check whether a string ends with a specified group of characters or not.

Note: The startsWith() and endsWith() will consider the case.
Example:

```
String str = new String("java program");
System.out.println(str.startsWith("ja"));
System.out.println(str.startsWith("jab"));
System.out.println(str.startsWith("JAVA"));
System.out.println(str.endsWith("ram"));
System.out.println(str.endsWith("Gram"));
```

10) **int indexOf(char):** This method will return the index of the first occurrence of the specified character.
11) **int lastIndexOf:** This method will return the index of the last occurrence of the specified character.
Note: The indexOf() and lastIndexOf() will return -1, if the specified character is not available.
Example:

```
String str = new String("java program");
System.out.println(str.indexOf('a'));
System.out.println(str.indexOf('p'));
System.out.println(str.indexOf('q'));
System.out.println(str.lastIndexOf('a'));
System.out.println(str.lastIndexOf('p'));
System.out.println(str.lastIndexOf('z'));
```

12) **String replace(char old, char new):** This method can be used to replace all the occurrences of the specified character with a new character.
Example:

```
String str = new String("java jug jar jungle");
System.out.println(str.replace('j', 'b'));
```

13) **String substring(int index):** This method will return a group of characters beginning from the specified index position up to the end of the string.
14) **String substring(int index, int offset):** This method will return a group of characters beginning from the specified index position up to the specified offset.

Note: The offset represents the position of a character in a string and it begins with 1.
Example:

```
String str = new String("java program");
System.out.println(str.substring(3));
System.out.println(str.substring (2,9));
```

15) **String toLowerCase():** This method will convert the contents of a string to completely lower case.
16) **String toUpperCase():** This method will convert the contents of a string to completely upper case.
Example:

```
String str = new String("java program");
```

```
System.out.println(str);
System.out.println(str.toLowerCase());
System.out.println(str.toUpperCase());
```

17) **String trim():** This method can be used to remove the leading and trialing spaces available in a string
Example:
```
String str = new String("java program");
System.out.println(str+"bye");
System.out.println(str.trim()+"bye");
```

18) **String intern():** This method will refer to an object available in string constant pool which was created at the time of object creation in heap memory.
Example:
```
String s1 = new String("hello");
String s2 = s1.intern();
System.out.println(s1);
System.out.println(s2);
```

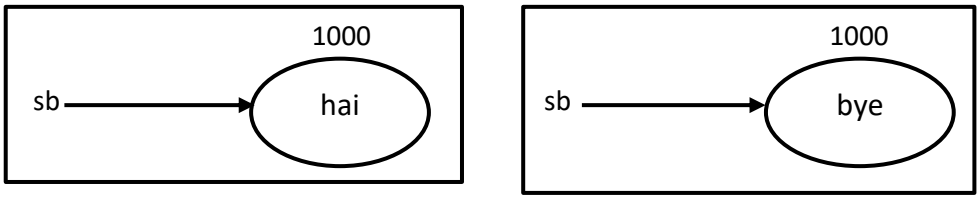
StringBuffer

StringBuffer is a predefined class, used for storing group of characters. StringBuffer object can be created in only one way and that is by using new operator.

Syntax:
```
StringBuffer sb = new StringBuffer("welcome");
```

The StringBuffer objects are mutable which means we can modify the content of the object.

Example:
```
StringBuffer sb = new StringBuffer("hai");
sb = new StringBuffer("bye");
```



Methods Of StringBuffer:

1) **int length():** This method will return the count of the number of characters available in StringBuffer.

   Example:
   ```
   StringBuffer sb = new StringBuffer("welcome");
   System.out.println(sb);
   System.out.println( sb.length());
   ```

2) **StringBuffer append(xxxx):** This method can be used to append the specified content to the existing StringBuffer object.

   Example:
   ```
   StringBuffer sb = new StringBuffer("java");
   System.out.println(sb.append(1.7));
   System.out.println(sb.append("program"));
   ```

3) **StringBuffer deleteCharAt(int index):** This method can be used to delete a character that is available specified index position.
4) **StringBuffer delete(int index, int offset):** This method can be used to delete a group of characters beginning with the specified index position up to the specified offset.

   Example:
   ```
   StringBuffer sb = new StringBuffer("java1.7program");
   System.out.println(sb.deleteCharAt(5));
   ```

System.out.println(sb.delete(3,10));

5) **String substring(int index):** This method can be used to retrieve a part of a StringBuffer beginning from the specified index position up to the end of the StringBuffer .
6) **String substring(int index, int offset):** This method can be used to retrieve a part of the StringBuffer beginning from the specified index position up to the specified offset.

   Note: The substring() will not modify the content of the StringBuffer.

   Example:
   StringBuffer sb = new StringBuffer("java 1.7program");
   System.out.println(sb.substring(6));
   System.out.println(sb.substring(1,4));

7) **StringBuffer insert(int index, xxx):** This method can be used to insert the specified content at the specified index position.

   Example:
   StringBuffer sb = new StringBuffer("program");
   System.out.println(sb.insert(0,"java"));
   System.out.println(sb.insert(4,1.7));

8) **StringBuffer replace(int index, int offset, String):** This method can be used to replace a group of characters beginning with the specified index position up to the specified offset, with the specified String.

   Example:
   StringBuffer sb = new StringBuffer("java1.8program");
   System.out.println(sb.replace (0,7,"jse"));

9) **StringBuffer reverse():** This method can be used to reverse the contents of StringBuffer.

   Example:
   StringBuffer sb = new StringBuffer("java1.7program");
   System.out.println(sb.reverse());

Program :

```
class Sample  {
    public static void main(String[] args)  {
        StringBuffer sb = new StringBuffer("java");
        System.out.println(sb.append("program").insert(0,"core")
        .delete(4,8) .append("test").replace(4,8,"frog").substring (7)
        .concat("example") .substring (4).replace ('e', 'n').toUpperCase()
        .chatAt (8));
    }
}
```

StringBuilder

The StringBuilder class is introduced in java 1.5 version and it can be used for storing a group of characters.

Syntax:
StringBuilder sb = new StringBuilder("java program");

StringBuilder objects are mutable that is we can modify or update the content of the StringBuilder object.

Difference between StringBuffer & StringBuilder:
The StringBuffer object is synchronized (thread-safe) i.e. it can be accessed by only one thread at a time, whereas StringBuilder objects is not synchronized i.e. it can be accessed by multiple threads at a time.