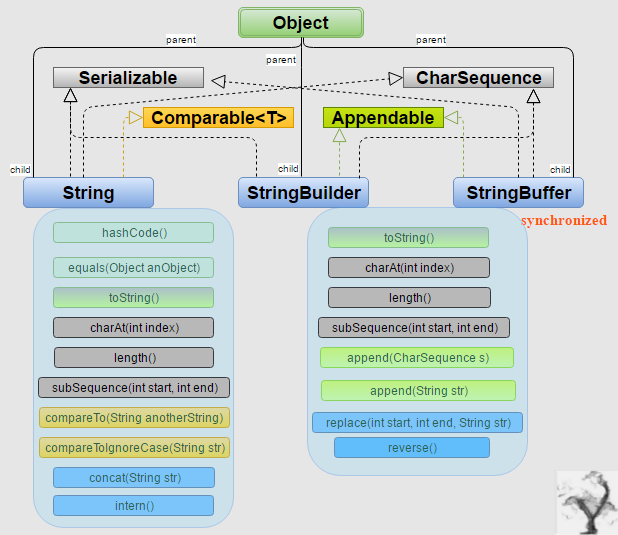
****

**StringBuffer :**

**StringBuffer class objects are mutable in Java**. It creates strings of flexible length that can be modified in terms of both length and content.

The methods of string buffer class can directly manipulate data inside the object. We can easily insert characters and substrings in the middle of the string or add another string to the end. StringBuffer can automatically expand to create room for such additions.

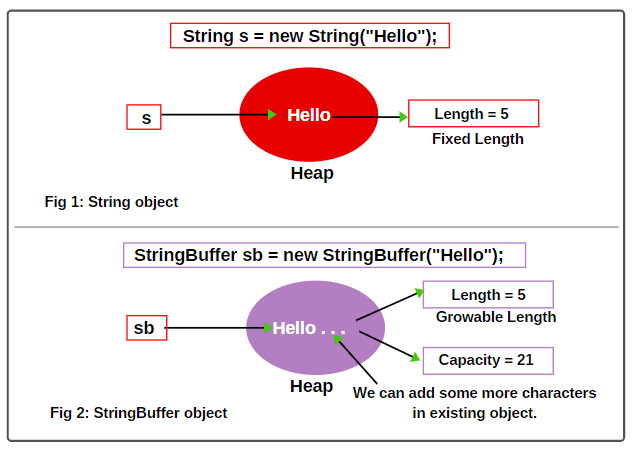
**Java StringBuffer class declaration**

public final class StringBuffer

       extends Object

               implements Serializable,CharSequence

StringBuffer class is present in java.lang package similar to the string class.



**Capacity:** The total number of characters hold in the StringBuffer object is called capacity.

**Length:** The number of characters already present in the StringBuffer object is called length.

## How to create StringBuffer Objects in Java

## 1.By new keyword

## StringBuffer sb= new StringBuffer();

## sb= “ kk”;

## Length = 2

## Initial Capacity = 16

## Now suppose 16 characters are completed. Can we add 17th character in the existing StringBuffer object?

## Once the capacity of StringBuffer object is filled, a new StringBuffer object will be created with bigger capacity.

## New capacity = (current capacity +1) \* 2

## New capacity = (16 + 1) \*2 = 34

## 2.StringBuffer sb = new StringBuffer (“kamal”);

## 3.StringBuffer sb = new StringBuffer( 30);

## 

## 

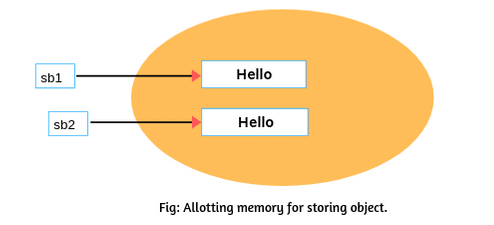
## 

## 

## 

StringBuffer sb1 = new  StringBuffer(“Hello”);

StringBuffer sb2 = new StringBuffer(“Hello”);



# StringBuffer Methods in Java

**1. append Method:**

The append( ) method appends (or concatenates) the string representation of any other data type (e.g., boolean, int, float, long, String, Object, etc) to the end of the sequence of characters in the StringBuffer object.

This method has several overloaded versions that are as follows:

* public StringBuffer append(boolean b)
* public StringBuffer append(char c)
* public StringBuffer append(char[ ] c )
* public StringBuffer append(CharSequence cs)
* public StringBuffer append(int i)
* public StringBuffer append(float f)
* public StringBuffer append(double d)
* public StringBuffer append(long l)
* public StringBuffer append(String s)
* public StringBuffer append(StringBuffer sb1)
* public StringBuffer append(Object o)
* public StringBuffer append(char[ ]  c, int begin, int num)
* public StringBuffer append(CharSequence cs, int begin, int num)

String.valueOf( ) is called for each parameter to get its string representation. The result is added to the end of the sequence of characters in the current StringBuffer object.

**1. length():**

The length() method returns the length of the current StringBuffer object. Length is the number of characters stored in buffer. The general syntax of length() method is as:

                  public int length()

**2. capacity():**

The capacity() method returns the current capacity of StringBuffer. The capacity is the total size of storage available for newly inserted characters, beyond which storage allocation will occur.

The general syntax of capacity() method is as:

                  public int capacity()

**3.charAt():**

This method obtains the value of a single character at the specified index. Its general form is as follows:

       public int charAt(int i)

**4. setCharAt():**

This method sets the value of a character at a specified index within StringBuffer object. The general form of this method is:

       public void setCharAt(int i,char ch)

**5. insert():**

Java StringBuffer class provides insert() method that inserts a string representation of all simple data types plus String, Object, and CharSequence into invoking StringBuffer object.

The general syntax of the insert() method is as follows:

public StringBuffer insert(int index,data type)

**6. delete():**

The delete() method deletes a sequence of characters from the calling StringBuffer object. The general form of the delete() method is as below:

        StringBuffer delete (int startIndex,int endIndex)

**7. deleteCharAt():**

The deleteCharAt( ) method deletes the character at the specified index and then returns the resulting StringBuffer object. The general form is as follows:

        StringBuffer deleteCharAt(int index)

**8. replace():**

StringBuffer class in Java provides a replace() method that replace one set of characters with another set inside a StringBuffer object. Its signature is given here:

StringBuffer replace(int startIndex,int endIndex,String s)

**9. indexOf():**

This method searches the invoking StringBuffer for the first occurrence of String s. It returns the index of the match, or –1 if no match is found. This method comes two flavors:

      public indexOf(String s)

      public indexOf(String s)

**10. lastIndexOf():**

This method searches the calling StringBuffer object for the last occurrence of String s. It returns the index of the match, or –1 if no match is found. It comes into flavors:

      public int lastIndexOf(String s)

      public int lastIndexOf(String s, int startIndex)

**11. substring( ):**

This method is used to obtain a portion of a StringBuffer by calling substring( ). It comes in the following two forms:

           String substring (int startIndex)

           String substring(int startIndex,int endIndex)

**12.reverse()** StirngBuffer class in Java provides a method reverse() that reverse the characters within a StringBuffer object. The reverse() method returns the reversed object on which it was called. Its general syntax is given as:

           StringBuffer reverse()

**13. getChars():**  
This method copies a substring of a StringBuffer object into a character array specified by c. The general syntax is as follows:

       public void getChars(int sourseStart,int sourceEnd,char[]c,int k)

# Java 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.

 StringBuffer and StringBuilder both represent a mutable sequence of characters. We can change the content of StringBuffer and StringBuilder without constructing a new object.

## StringBuilder Methods in Java

### 1) StringBuilder append() method

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

**StringBuilderExample.java**

**class** StringBuilderExample{

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

**Output:**

Hello Java

### 2) StringBuilder insert() method

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

**StringBuilderExample2.java**

**class** StringBuilderExample2{

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

**Output:**

HJavaello

### 3) StringBuilder replace() method

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

**StringBuilderExample3.java**

**class** StringBuilderExample3{

1. **public** **static** **void** main(String args[]){
2. StringBuilder sb=**new** StringBuilder("Hello");
3. sb.replace(1,3,"Java");
4. System.out.println(sb);//prints HJavalo
5. }
6. }

**Output:**

HJavalo

### 4) StringBuilder delete() method

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

**StringBuilderExample4.java**

**class** StringBuilderExample4{

1. **public** **static** **void** main(String args[]){
2. StringBuilder sb=**new** StringBuilder("Hello");
3. sb.delete(1,3);
4. System.out.println(sb);//prints Hlo
5. }
6. }

**Output:**

Hlo

### 5) StringBuilder reverse() method

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

**StringBuilderExample5.java**

**class** StringBuilderExample5{

1. **public** **static** **void** main(String args[]){
2. StringBuilder sb=**new** StringBuilder("Hello");
3. sb.reverse();
4. System.out.println(sb);//prints olleH
5. }
6. }

**Output:**

olleH

### 6) StringBuilder capacity() method

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder 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.

**StringBuilderExample6.java**

**class** StringBuilderExample6{

1. **public** **static** **void** main(String args[]){
2. StringBuilder sb=**new** StringBuilder();
3. System.out.println(sb.capacity());//default 16
4. sb.append("Hello");
5. System.out.println(sb.capacity());//now 16
6. sb.append("Java is my favourite language");
7. System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2
8. }
9. }

**Output:**

16

16

34

### 7) StringBuilder ensureCapacity() method

The ensureCapacity() method of StringBuilder 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.

**StringBuilderExample7.java**

**class** StringBuilderExample7{

1. **public** **static** **void** main(String args[]){
2. StringBuilder sb=**new** StringBuilder();
3. System.out.println(sb.capacity());//default 16
4. sb.append("Hello");
5. System.out.println(sb.capacity());//now 16
6. sb.append("Java is my favourite language");
7. System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2
8. sb.ensureCapacity(10);//now no change
9. System.out.println(sb.capacity());//now 34
10. sb.ensureCapacity(50);//now (34\*2)+2
11. System.out.println(sb.capacity());//now 70
12. }
13. }

**Output:**

16

16

34

34

70

## 

**Difference between String vs StringBuffer vs StringBuilder**

**1. Immutable and Mutable:**

a) [*String in Java*](https://www.scientecheasy.com/2020/05/string-in-java.html/) is immutable and final class. That means objects of string class cannot be changed once it created. So, whenever we change any string or perform string manipulation, it will create a new object.

Let’s understand it with the help of an example program.

**Program code:**

public class Test {

 public static void main(String [] args)

 {

  String str = new String("java");

    str.concat(" program"); // concat() method adds string at the end.

    System.out.println(str); // It will print "java program" because string is an immutable object.

 }

}

Output:

        java program

In the above program, the new object is not assigning with any reference variable. Therefore, it is called unreferenced object and the garbage collector will automatically remove it from the memory.

b) *[StringBuffer in Java](https://www.scientecheasy.com/2020/05/stringbuffer-class-in-java.html/)* is a mutable class. That means, objects of StringBuffer class can be changed. Let’s understand it with the help of an example program.

c) *[StringBuilder](https://www.scientecheasy.com/2022/01/java-stringbuilder-class.html/)* is a mutable class in Java. That means, objects of StringBuilder class can be modified or changed. Look at the program source code to understand it.

**2. Object creation:**

a) An instance of String can be created into two ways:

* Using string literal
* Using new keyword

Let’s take an example to understand both ways.

// By String literal.

String str = "Java";

 // By new operator.

String str2 = new String("Programming");

b) An instance of StringBuffer and StringBuilder can be created in only one way:

* Using new operator

For example:

StringBuffer sb = new StringBuffer("Java Programming");

StringBuilder sbl = new StringBuilder("Java World");

**3. String Constant Pool:**

a) Internally, String class uses the string constant pool (SCP) for storing string objects to save the memory. Whenever we create a string literal, JVM checks string constant pool first.

If the string already exists in string constant pool, no new string object will be created in the string pool by JVM.

b) There is no concept of String constant pool in the case of StringBuffer and StringBuilder.

**4. Uses:**

a) String class should be used when the content is fixed and not change frequently.

b) StringBuffer class should be used when content is not fixed and often changing but also need thread safety.

c) StringBuilder class should be used when content is not fixed and often changing but also do not need thread safety.

**5. Memory:**

a) String consume more memory when we concat too many strings because every time it creates a new object.

b) StringBuffer and StringBuilder consume less memory when we append strings.

**6. Speed:**

a) String is slower than StringBuffer and StringBuilder.

b) StringBuffer is slower than StringBuilder, but it is faster than String.

c) StringBuilder is faster than string and StringBuffer.

Let’s create a program in which we will test performance of String, StringBuffer, and StringBuilder classes.

**7. Override equals() and hashCode() methods of Object class:**

a) String class overrides the equals() of the Object class to compare contents of two strings as well as hashCode() method.

b) SringBuffer and StringBuilder classes do not override the equals() and hashCode() methods of Object class.

Let’s create a Java program in which we will compare two strings using equals() method. The equals() method of Object class compares the references of two objects. If two objects have the same references, then it returns true.

The equals() method of the String class is not same as the equals() method of the Object class. It is overridden, this method accepts a String value and compares it with the current object and returns true .

Whereas the equals() method of String class is not similar to the equals() method of Object class. It is overridden method of equals() method of Object class that accepts a string value and compares it with the current string object.

It returns true only if the sequence of characters in the both String objects are exactly the same. If the equals() methods returns true on comparing two string objects, their hash code values must be same.

As you observe in the program, sting class overrides equals() method of Object class, compares contents of strings, and returns true as both are having the same contents.

StringBuffer class does not override the equals() method of Object class. Its functionality is similar to the equals() method of the Object class. Let’s understand and see if we can compare two stringBuffer objects using equals() method.

As you can observe in this program, when we are comparing two stringBuffer objects using equal() method, it returns false even content is same. Because StringBuffer class not overriding equals() and hashcode() methods of Object class. ​

**8. Thread-safe:**

a) String class is not recommended to use in threaded environment.

b) StringBuffer class is recommended to use in multithreaded environment.

