# String Handling

Agend

1. Introduction

1. Java.lang.Object class

2. java.lang.String class

2.  Importance of String constant pool (SCP)

3.  Interning of String objects

4.  String class constructors

5.  Important methods of String class

3. StringBuffer

6.  Constructors

7.  Important methods of StringBuffer

4. StringBuilder (1.5v)

8.  StringBuffer Vs StringBuilder

9.  String vs StringBuffer vs StringBuilder

5. Wrapper classes

10.  Constructors

11.  Wrapper class Constructor summery

6. Utility methods

12.  valueOf() method

13.  xxxValue() method

14.  parseXxx() method

15.  toString() method

7. Dancing between String, wrapper object and primitive

8. Partial Hierarchy of java.lang package

9. Autoboxing and Autounboxing

16.  Autoboxing

17.  Autounboxing

18.  Conclusions

The following are some of important classes present in java.lang package.

1. Object class

2. String class

3. StringBuffer class

4. StringBuilder class (1.5 v)

5. Wrapper Classes

6. Autoboxing and Autounboxing(1.5 v)

 For writing any java program the most commonly required classes and

interfaces are encapsulated in the separate package which is nothing but

java.lang package.

 It is not required to import java.lang package in our program because it is

available by default to every java program.

What is your favorite package? Why java.lang is your favorite package?

It is not required to import lang package explicitly but the remaining packages we have

to import.

## Java.lang.Object class:

1. For any java object whether it is predefine or customized the most commonly

required methods are encapsulated into a separate class which is nothing but

object class.

2. As object class acts as a root (or) parent (or) super for all java classes, by default

its methods are available to every java class.

3. Note : If our class doesn't extends any other class then it is the direct child class

of object

If our class extends any other class then it is the indirect child class of Object.

The following is the list of all methods present in java.lang Object class :

1. public String toString();

2. public native int hashCode();

3. public boolean equals(Object o);

4. protected native Object clone()throws CloneNotSupportedException;

5. public final Class getClass();

6. protected void finalize()throws Throwable;

7. public final void wait() throws InterruptedException;

8. public final native void wait()throws InterruptedException;

9. public final void wait(long ms,int ns)throws InterruptedException;

10. public final native void notify();

11. public final native void notifyAll();

toString( ) method :

1. We can use this method to get string representation of an object.

2. Whenever we are try to print any object reference internally toString() method

will be executed.

3. If our class doesn't contain toString() method then Object class toString()

method will be executed.

4. Example:

5. System.out.println(s1); => super(s1.toString());

6. Example 1:

7. class Student

8. {

9. String name;

10. int rollno;

11. Student(String name, int rollno)

12. {

13. this.name=name;

14. this.rollno=rollno;

15. }

16. public static void main(String args[]){

17. Student s1=new Student("saicharan",101);

18. Student s2=new Student("ashok",102);

19. System.out.println(s1);

20. System.out.println(s1.toString());

21. System.out.println(s2);

22. }

23. }

24. Output:

25. Student@3e25a5

26. Student@3e25a5

27. Student@19821f

28.

35. To provide our own String representation we have to override toString() method

in our class.

Ex : For example whenever we are try to print student reference to print his a

name and roll no we have to override toString() method as follows.

36. public String toString(){

37. return name+"........"+rollno;

38. }

39. In String class, StringBuffer, StringBuilder, wrapper classes and in all collection

classes toString() method is overridden for meaningful string representation.

Hence in our classes also highly recommended to override toString() method.

40.

41. Example 2:

42.

43. class Test{

44. public String toString(){

45. return "Test";

46. }

47. public static void main(String[] args){

48. Integer i=new Integer(10);

49. String s=new String("ashok");

```
Test t=new Test();
System.out.println(i);
System.out.println(s);
System.out.println(t);
}
}
Output:
10
ashok
Test
```

## hashCode() method :

1. For every object jvm will generate a unique number which is nothing but hashCode.

2. Jvm will using hashCode while saving objects into hashing related data structures like HashSet, HashMap, and Hashtable etc.

3. If the objects are stored according to hashCode searching will become very efficient (The most powerful search algorithm is hashing which will work based on hashCode).

4. If we didn't override hashCode() method then Object class hashCode() method

will be executed which generates hashCode based on address of the object but it

doesn't mean hashCode represents address of the object.

## equals() method:

1. We can use this method to check equivalence of two objects.

2. If our class doesn't contain .equals() method then object class .equals() method

will be executed which is always meant for reference comparison[address

comparison]. i.e., if two references pointing to the same object then only .equals(

) method returns true .

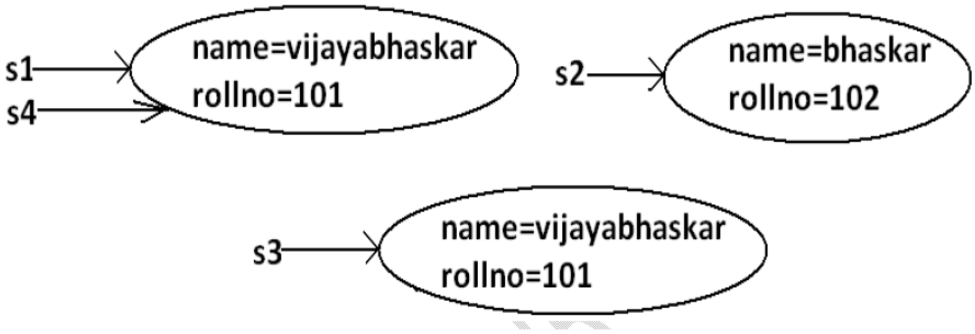
Example 5:

```
class Student

{

String name;

int rollno;

Student(String name,int rollno)

{

this.name=name;
this.rollno=rollno;

}

public static void main(String[] args){

Student s1=new Student("vijayabhaskar",101);

Student s2=new Student("bhaskar",102);

Student s3=new Student("vijayabhaskar",101);

Student s4=s1;

System.out.println(s1.equals(s2));

System.out.println(s1.equals(s3));

System.out.println(s1.equals(s4));

}}
```

Output:

```
False

False

True
```

Diagram:



⍰

In

the

above program Object class .equals() method got executed which is always

meant for reference comparison that is if two references pointing to the same

object then only .equals(() method returns true.

In object class .equals() method is implemented as follows which is meant for

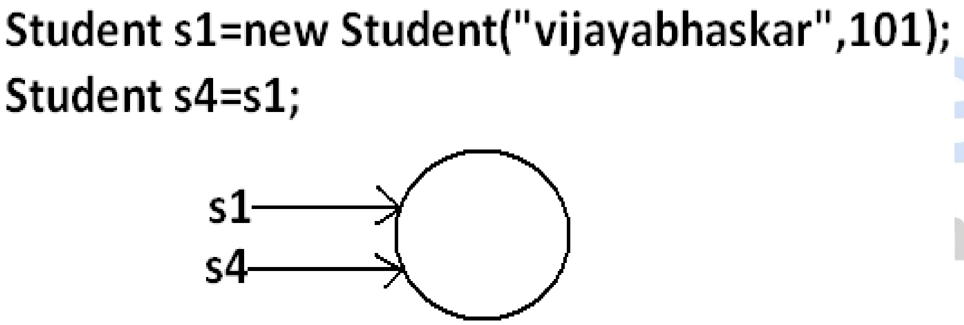
reference comparison.

 public boolean equals(Object obj) {

 return (this == obj);

 }

Diagram:

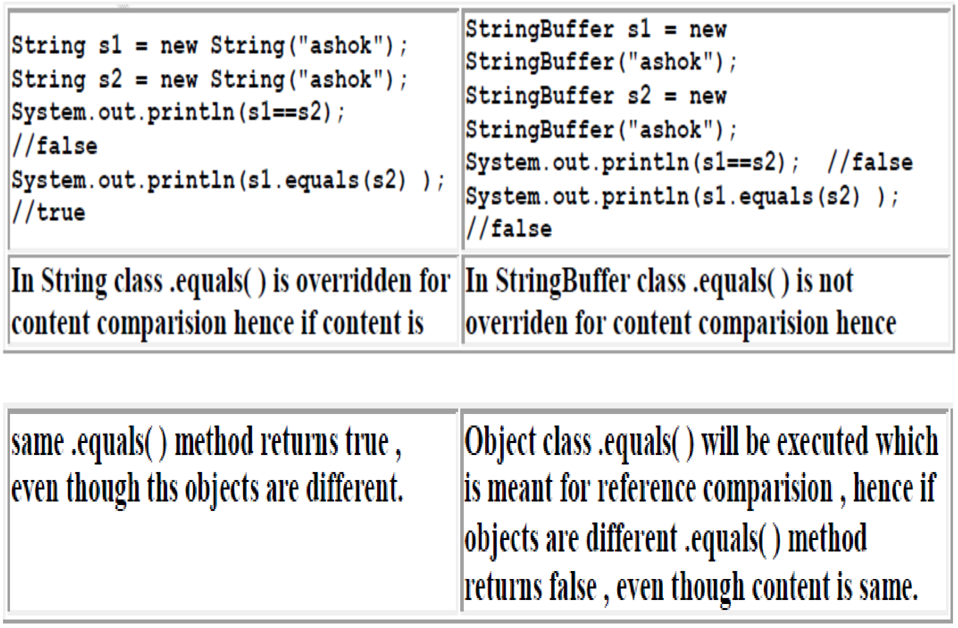


```
Student s1=new Student("vijayabhaskar",101);
Student s4=s1;
```

If 2 references pointing to the same object then .equals() method return true directly

without performing any content comparison this approach improves performance of the

system



| | |
|---|---|
| String s1 = new String("ashok");<br>String s2 = new String("ashok");<br>System.out.println(s1==s2);<br>//false<br>System.out.println(s1.equals(s2) );<br>//true | StringBuffer s1 = new<br>StringBuffer("ashok");<br>StringBuffer s2 = new<br>StringBuffer("ashok");<br>System.out.println(s1==s2);  //false<br>System.out.println(s1.equals(s2) );<br>//false |
| In String class .equals( ) is overridden for content comparision hence if content is same .equals( ) method returns true , even though ths objects are different. | In StringBuffer class .equals( ) is not overriden for content comparision hence Object class .equals( ) will be executed which is meant for reference comparision , hence if objects are different .equals( ) method returns false , even though content is same. |

Note : In String class , Wrapper classes and all collection classes .equals( ) method is

overriden for content comparision

## Relationship between .equals() method and ==(double equal operator) :

1. If r1==r2 is true then r1.equals(r2) is always true i.e., if two objects are equal by

== operator then these objects are always equal by .equals( ) method also.

2. If r1==r2 is false then we can't conclude anything about r1.equals(r2) it may

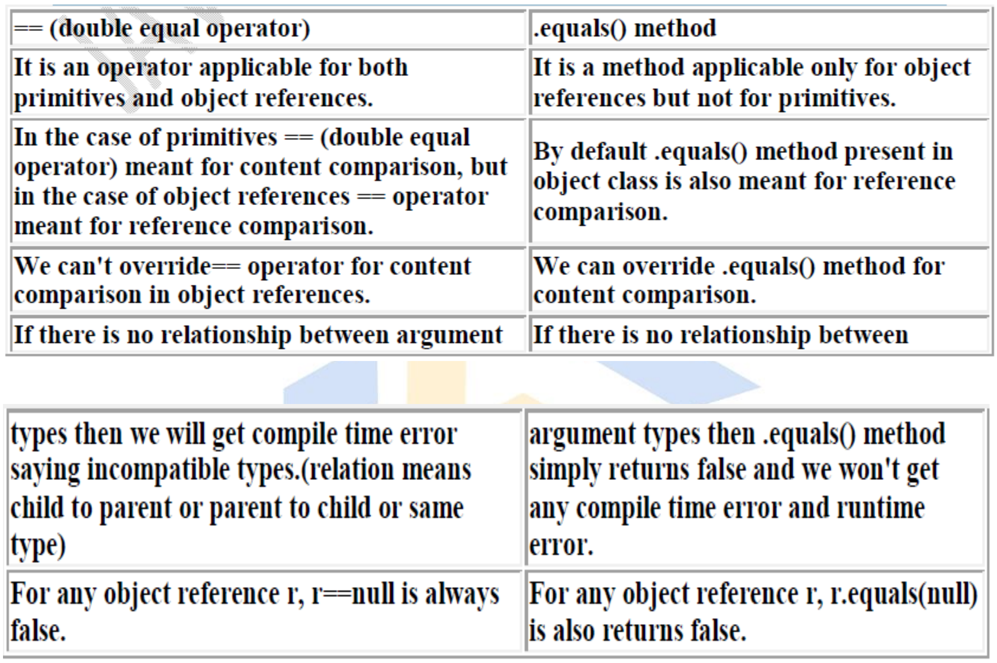
return true (or) false.

3. If r1.equals(r2) is true then we can't conclude anything about r1==r2 it may

returns true (or) false.

4. If r1.equals(r2) is false then r1==r2 is always false.

## Differences between == (double equal operator) and .equals() method?



| == (double equal operator) | .equals() method |
|---|---|
| It is an operator applicable for both primitives and object references. | It is a method applicable only for object references but not for primitives. |
| In the case of primitives == (double equal operator) meant for content comparison, but in the case of object references == operator meant for reference comparison. | By default .equals() method present in object class is also meant for reference comparison. |
| We can't override== operator for content comparison in object references. | We can override .equals() method for content comparison. |
| If there is no relationship between argument types then we will get compile time error saying incompatible types.(relation means child to parent or parent to child or same type) | If there is no relationship between argument types then .equals() method simply returns false and we won't get any compile time error and runtime error. |
| For any object reference r, r==null is always false. | For any object reference r, r.equals(null) is also returns false. |

String s = new String("ashok");

StringBuffer sb = new StringBuffer("ashok");

System.out.println(s == sb); // CE : incomparable types : String and StringBuffer

System.out.println(s.equals(sb)); //false

Note:

in general we can use == (double equal operator) for reference comparison whereas

.equals() method for content comparison.

return false;

```
}
return false;
}
public static void main(String[] args){
Person p1=new Person("vijayabhaskar",101);
Person p2=new Person("vijayabhaskar",101);
Integer i=new Integer(102);
System.out.println(p1.equals(p2));
System.out.println(p1.equals(i));
}
}
```

Output:

```
True
False
```

### Clone () method:

1. The process of creating exactly duplicate object is called cloning.

2. The main objective of cloning is to maintain backup purposes.(i.e., if something goes wrong we can recover the situation by using backup copy.)

3. We can perform cloning by using clone() method of Object class.

```
protected native object clone() throws CloneNotSupportedException;
```

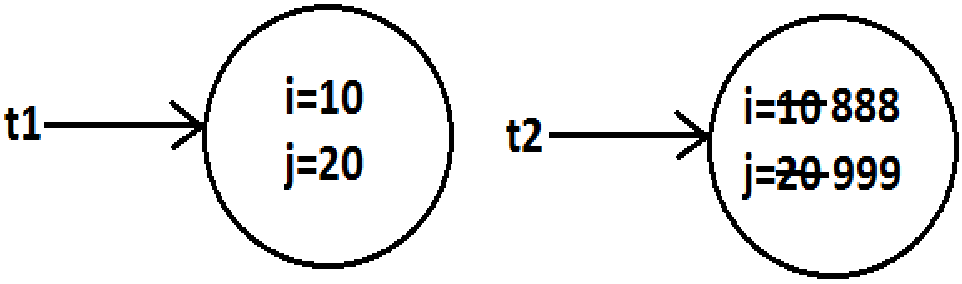
Example:

```
class Test implements Cloneable
{
int i=10;
int j=20;
public static void main(String[] args)throws
CloneNotSupportedException
{
Test t1=new Test();
Test t2=(Test)t1.clone();
t2.i=888;
t2.j=999;
System.out.println(t1.i+"---------------"+t1.j);
System.out.println(t2.i+"---------------"+t2.j);
}
}
```

Output:

```
10---------------20
888---------------999
```

Diagram:



 We can perform cloning only for Cloneable objects.

 An object is said to be Cloneable if and only if the corresponding class

implements Cloneable interface.

 Cloneable interface present in java.lang package and does not contain any

methods. It is a marker interface where the required ability will be provided

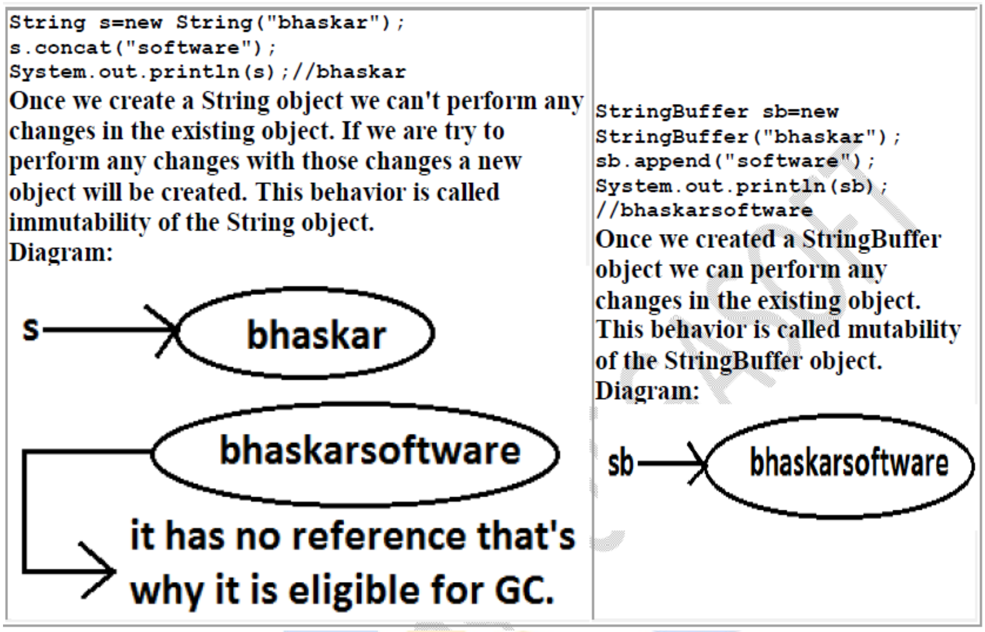
automatically by the JVM.

 If we are trying to perform cloning or non-clonable objects then we will get

RuntimeException saying CloneNotSupportedException.

java.lang.String class :

Case 1:



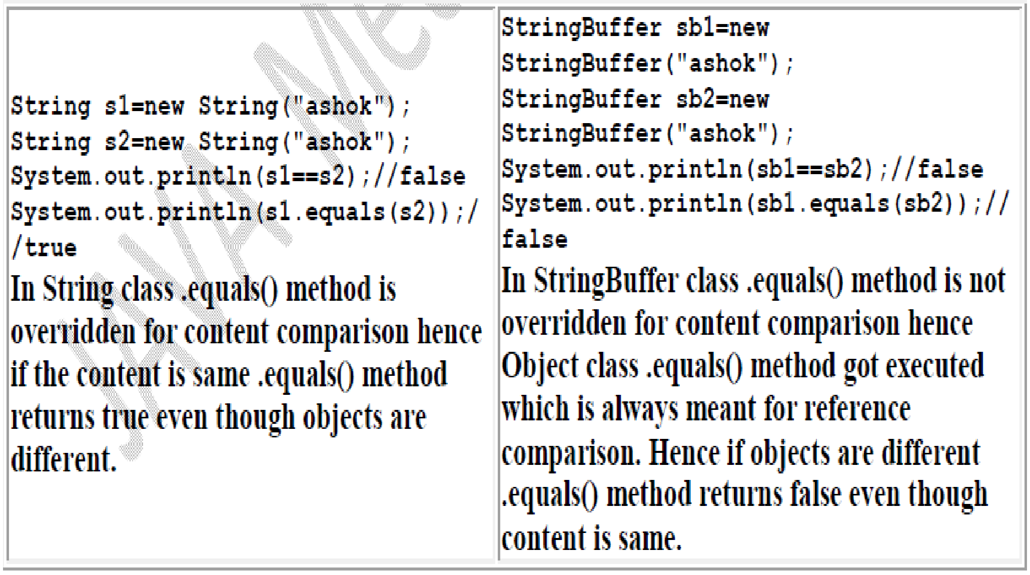
```
String s=new String("bhaskar");
s.concat("software");
System.out.println(s);//bhaskar
```

Once we create a String object we can't perform any changes in the existing object. If we are try to perform any changes with those changes a new object will be created. This behavior is called immutability of the String object.
Diagram:

```
StringBuffer sb=new
StringBuffer("bhaskar");
sb.append("software");
System.out.println(sb);
//bhaskarsoftware
```

Once we created a StringBuffer object we can perform any changes in the existing object. This behavior is called mutability of the StringBuffer object.
Diagram:

Case 2 :

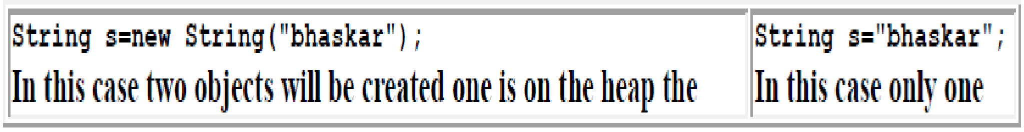


```
String s1=new String("ashok");
String s2=new String("ashok");
System.out.println(s1==s2);//false
System.out.println(s1.equals(s2));/
/true
```

In String class .equals() method is overridden for content comparison hence if the content is same .equals() method returns true even though objects are different.

```
StringBuffer sb1=new
StringBuffer("ashok");
StringBuffer sb2=new
StringBuffer("ashok");
System.out.println(sb1==sb2);//false
System.out.println(sb1.equals(sb2));//
false
```

In StringBuffer class .equals() method is not overridden for content comparison hence Object class .equals() method got executed which is always meant for reference comparison. Hence if objects are different .equals() method returns false even though content is same.

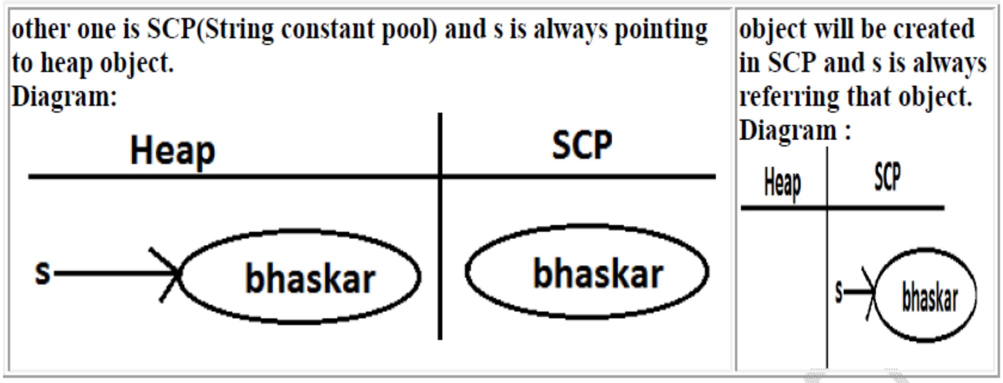


```
String s=new String("bhaskar");
```

In this case two objects will be created one is on the heap the

```
String s="bhaskar";
```

In this case only one



| | |
|---|---|
| other one is SCP(String constant pool) and s is always pointing to heap object.<br>Diagram: | object will be created in SCP and s is always referring that object.<br>Diagram : |

Note :

1. Object creation in SCP is always optional 1st JVM will check is any object

already created with required content or not. If it is already available then it will

reuse existing object instead of creating new object. If it is not already there then

only a new object will be created. Hence there is no chance of existing 2 objects

with same content on SCP that is duplicate objects are not allowed in SCP.

2. Garbage collector can't access SCP area hence even though object doesn't have

any reference still that object is not eligible for GC if it is present in SCP.

3. All SCP objects will be destroyed at the time of JVM shutdown automatically.

Example 1:

String s1=new String("bhaskar");

String s2=new String("bhaskar");

String s3="bhaskar";

String s4="bhaskar";

Note :

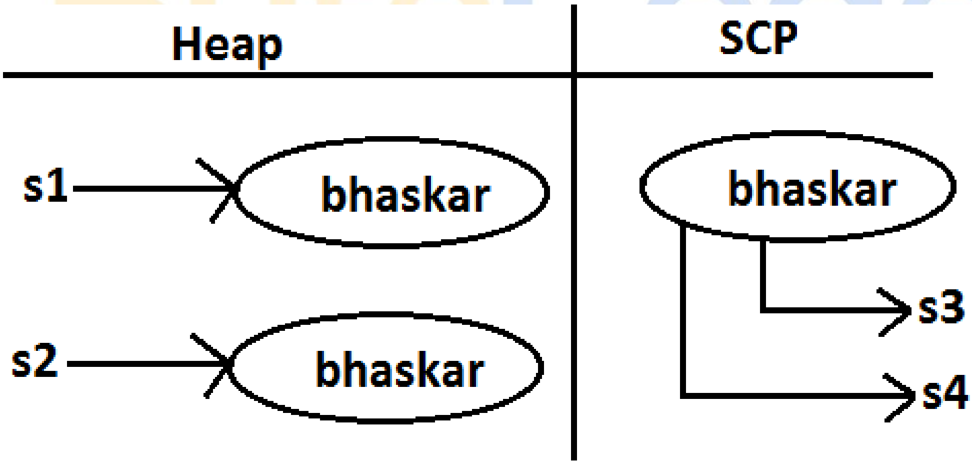
When ever we are using new operator compulsory a new object will be created on the

Heap . There may be a chance of existing two objects with same content on the heap but

there is no chance of existing two objects with same content on SCP . i.e., duplicate

objects possible in the heap but not in SCP .

Diagram :



Example 2:

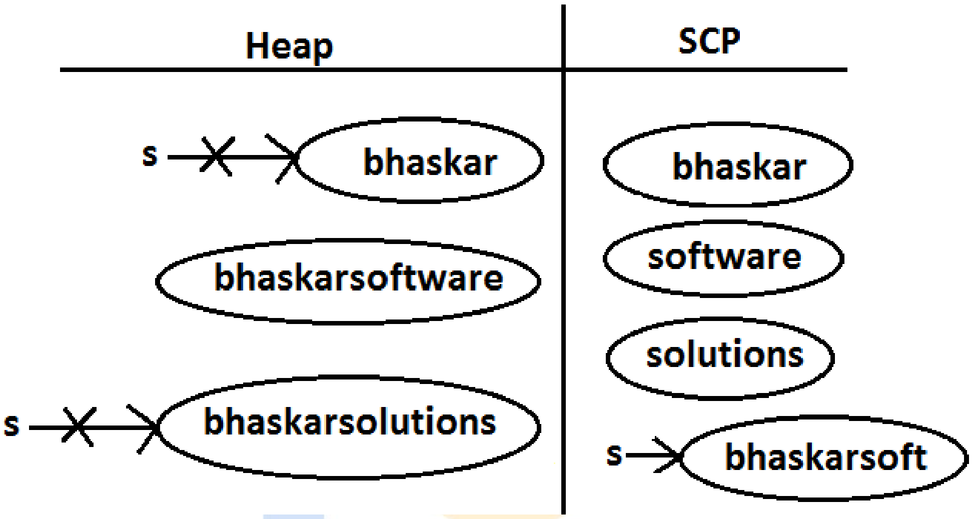
String s=new String("bhaskar");

s.concat("software");

s=s.concat("solutions");

s="bhaskarsoft";

Diagram :



For every String Constant one object will be created in SCP. Because of runtime

operation if an object is required to create compulsory that object should be placed on

the heap but not SCP.

Example 3:

String s1=new String("spring");

s1.concat("fall");

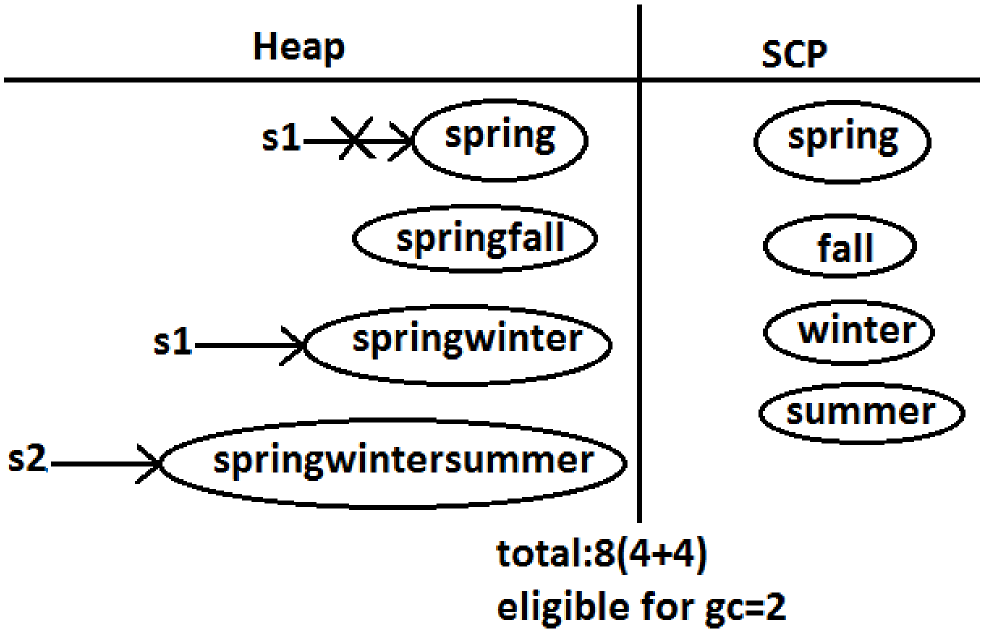
s1=s1+"winter";

String s2=s1.concat("summer");

System.out.println(s1);

```
System.out.println(s2);
```

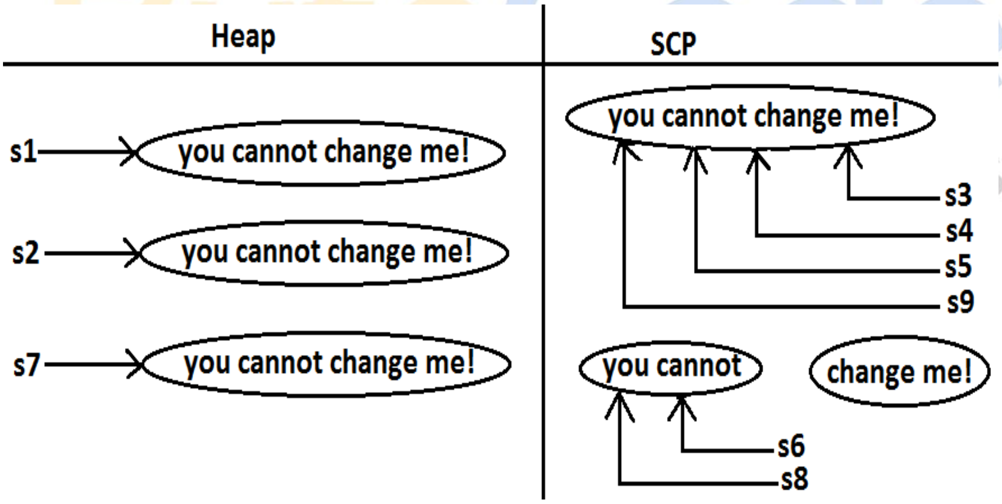
Diagram :



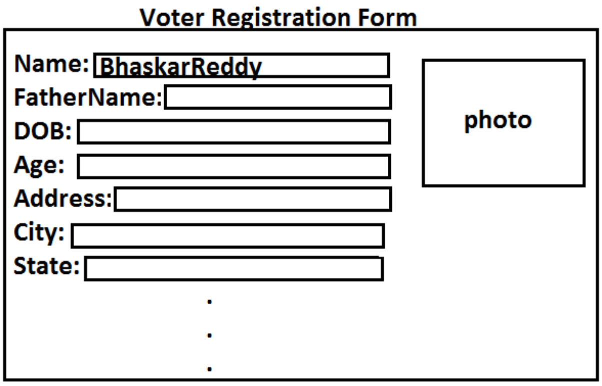
Example:

```
class StringDemo

{

public static void main(String[] args)

{

String s1=new String("you cannot change me!");

String s2=new String("you cannot change me!");

System.out.println(s1==s2);//false

String s3="you cannot change me!";

System.out.println(s1==s3);//false

String s4="you cannot change me!";
System.out.println(s3==s4);//true

String s5="you cannot "+"change me!";

System.out.println(s3==s5);//true

String s6="you cannot ";

String s7=s6+"change me!";

System.out.println(s3==s7);//false

final String s8="you cannot ";

String s9=s8+"change me!";

System.out.println(s3==s9);//true

System.out.println(s6==s8);//true

}

}
```

Diagram:



## Importance of String constant pool (SCP) :



Voter Registration Form

Name: BhaskarReddy

FatherName:

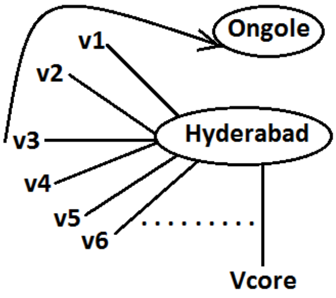
DOB:

Age:

Address:

City:

State:



1. In our program if any String object is required to use repeatedly then it is not

recommended to create multiple object with same content it reduces

performance of the system and effects memory utilization.

2. We can create only one copy and we can reuse the same object for every

requirement. This approach improves performance and memory utilization we

can achieve this by using "scp".

3. In SCP several references pointing to same object the main disadvantage in this

approach is by using one reference if we are performing any change the

remaining references will be impacted. To overcome this problem sun people

implemented immutability concept for String objects.

4. According to this once we creates a String object we can't perform any changes

in the existing object if we are trying to perform any changes with those changes

a new String object will be created hence immutability is the main disadvantage

of scp.

FAQS :

1. What is the main difference between String and StringBuilder?

2. What is the main difference between String and StringBuffer ?

3. Other than immutability and mutability is there any other difference between

String and StringBuffer ?

In String .equals( ) method meant for content comparison where as in

StringBuffer meant for reference comparision .

4. What is the meaning of immutability and mutability?

5. Explain immutability and mutability with an example?

6. What is SCP?

A specially designed memory area for the String literals/objects .

7. What is the advantage of SCP?

Instead of creating a separate object for every requirement we can create only one object and we can reuse same object for every requirement. This approach

improves performance and memory utilization.

8. What is the disadvantage of SCP?

In SCP as several references pointing to the same object by using one reference if

we are performing any changes the remaining references will be inflected. To

prevent this compulsory String objects should be immutable. That is

immutability is the disadvantage of SCP.

9. Why SCP like concept available only for the String but not for the StringBuffer?

As String object is the most commonly used object sun people provided a

specially designed memory area like SCP to improve memory utilization and

performance.

But StringBuffer object is not commonly used object hence specially designed

memory area is not at all required.

10. Why String objects are immutable where as StringBuffer objects are mutable?

In the case of String as several references pointing to the same object, by using

one reference if we are allowed perform the change the remaining references will

be impacted. To prevent this once we created a String object we can't perform

any change in the existing object that is immutability is only due to SCP.

But in the case of StringBuffer for every requirement we are creating a separate

object will be created by using one reference if we are performing any change in

the object the remaining references won't be impacted hence immutability

concept is not require for the StringBuffer.

11. Similar to String objects any other objects are immutable in java?

In addition to String objects , all wrapper objects are immutable in java.

12. Is it possible to create our own mutable class?

Yes.

13. Explain the process of creating our own immutable class with an example?

14. What is the difference between final and immutability?

15. What is interning of String objects?

Interning of String objects :

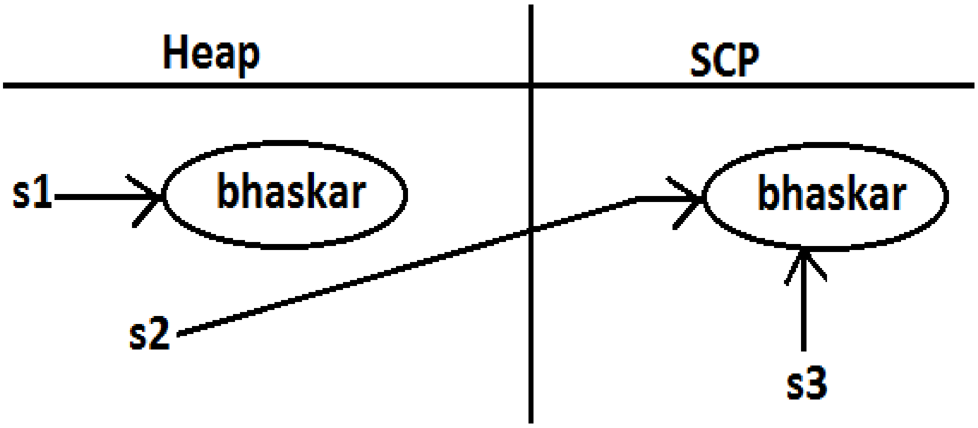
By using heap object reference, if we want to get corresponding SCP object , then we

should go for intern() method.

Example 1:

```
class StringDemo {
public static void main(String[] args) {
String s1=new String("bhaskar");
String s2=s1.intern();
System.out.println(s1==s2); //false
String s3="bhaskar";
System.out.println(s2==s3);//true
}
}
```

Diagram:



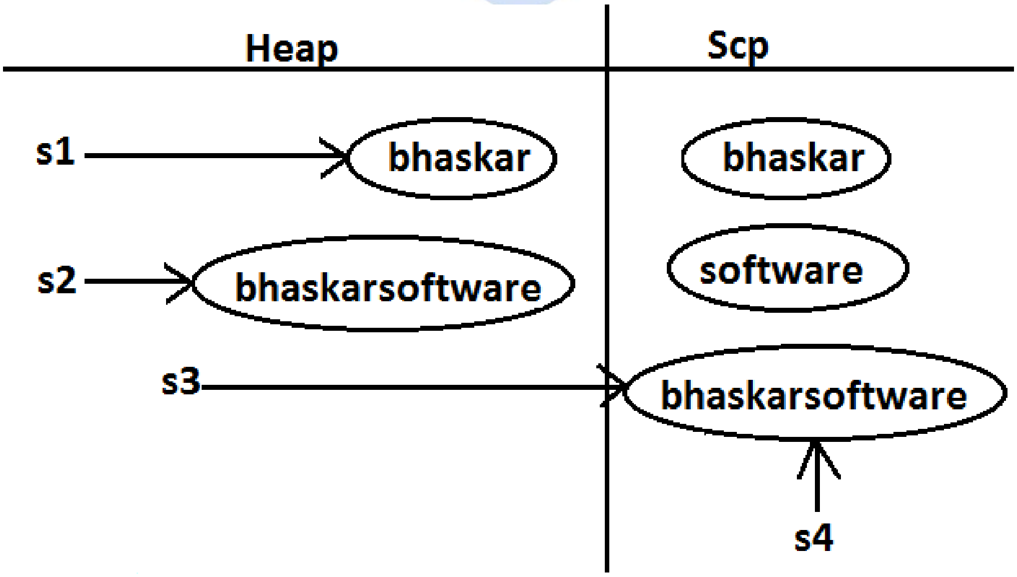
If the corresponding object is not there in SCP then intern() method itself will create

that object and returns it.

Example 2:

```
class StringDemo {
public static void main(String[] args) {
String s1=new String("bhaskar");
String s2=s1.concat("software");
String s3=s2.intern();
String s4="bhaskarsoftware";
System.out.println(s3==s4);//true
```

Diagram 2:



## String class constructors :

1. String s=new String();

Creates an empty String Object.

2. String s=new String(String literals);

To create an equivalent String object for the given String literal on the heap.

3. String s=new String(StringBuffer sb);

Creates an equivalent String object for the given StringBuffer.

4. String s=new String(char[] ch);

creates an equivalent String object for the given char[ ] array.

Example:

```
class StringDemo {
public static void main(String[] args) {
char[] ch={'a','b','c'} ;
String s=new String(ch);
System.out.println(ch);//abc
}
}
```

5. String s=new String(byte[] b);

Create an equivalent String object for the given byte[] array.

Example:

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

```
byte[] b={100,101,102};

String s=new String(b);

System.out.println(s);//def

}

}
```

Important methods of String class:

1. public char charAt(int index);

Returns the character locating at specified index.

Example:

```
class StringDemo {

public static void main(String[] args) {

String s="ashok";

System.out.println(s.charAt(3));//o

System.out.println(s.charAt(100));// RE :

StringIndexOutOfBoundsException

}

}

// index is zero based
```

2. public String concat(String str);

3. Example:

4. class StringDemo {

```
5. public static void main(String[] args) {
6. String s="ashok";
7. s=s.concat("software");
8. //s=s+"software";
9. //s+="software";
10. System.out.println(s);//ashoksoftware
11. }
12. }
```

The overloaded "+" and "+=" operators also meant for concatenation purpose

only.

13. public boolean equals(Object o);

For content comparison where case is important.

It is the overriding version of Object class .equals() method.

14. public boolean equalsIgnoreCase(String s);

For content comparison where case is not important.

Example:

```
class StringDemo {
public static void main(String[] args) {
String s="java";
System.out.println(s.equals("JAVA"));//false
System.out.println(s.equalsIgnoreCase("JAVA"));//true
}
}
```

Note: We can validate username by using .equalsIgnoreCase() method where

case is not important and we can validate password by using .equals() method

where case is important.

15. public String substring(int begin);

Return the substring from begin index to end of the string.

Example:

```
class StringDemo {
public static void main(String[] args) {
String s="ashoksoft";
System.out.println(s.substring(5));//soft
}
}
```

16. public String substring(int begin, int end);

Returns the substring from begin index to end-1 index.

Example:

```
class StringDemo {
```

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

String s="ashoksoft";

System.out.println(s.substring(5));//soft

System.out.println(s.substring(3,7));//okso

}

}
```

17. public int length();

Returns the number of characters present in the string.

Example:

```
class StringDemo {

public static void main(String[] args) {

String s="jobs4times";

System.out.println(s.length());//10

//System.out.println(s.length);//compile time error

}

}

/*

CE :

StringDemo.java:7: cannot find symbol

symbol : variable length

location: class java.lang.String
```

```
*/
```

Note: length is the variable applicable for arrays where as length() method is

applicable for String object.

18. public String replace(char old, char new);

To replace every old character with a new character.

Example:

```
class StringDemo {
public static void main(String[] args) {
String s="ababab";
System.out.println(s.replace('a','b'));//bbbbbb
}
}
```

19. public String toLowerCase();

Converts the all characters of the string to lowercase.

Example:

```
class StringDemo {
public static void main(String[] args) {
String s="ASHOK";
System.out.println(s.toLowerCase());//ashok
}
}
```

20. public String toUpperCase();

Converts the all characters of the string to uppercase.

Example :

```
class StringDemo {

public static void main(String[] args) {

String s="ashok";

System.out.println(s.toUpperCase());//ASHOK

}

}
```

21. public String trim();

We can use this method to remove blank spaces present at beginning and end of

the string but not blank spaces present at middle of the String.

Example:

```
class StringDemo {

public static void main(String[] args) {

String s=" sai charan ";

System.out.println(s.trim());//sai charan

}

}
```

22. public int indexOf(char ch);

It returns index of 1st occurrence of the specified character if the specified

character is not available then return -1.

Example:

```
class StringDemo {

public static void main(String[] args) {

String s="saicharan";

System.out.println(s.indexOf('c')); // 3

System.out.println(s.indexOf('z')); // -1

}

}
```

23. public int lastIndexOf(Char ch);

It returns index of last occurrence of the specified character if the specified

character is not available then return -1.

Example:

```
class StringDemo {

public static void main(String[] args) {

String s="arunkumar";

System.out.println(s.lastIndexOf('a'));//7
System.out.println(s.indexOf('z'));//-1
}
}
```

Note :

Because runtime operation if there is a change in content with those changes a new

object will be created only on the heap but not in SCP.

If there is no change in content no new object will be created the same object will be

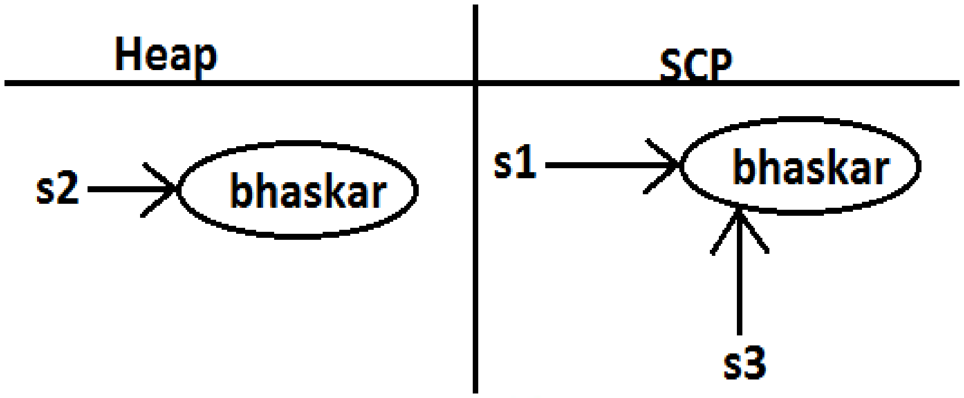
reused.

This rule is same whether object present on the Heap or SCP

Example 1 :

```
class StringDemo {
public static void main(String[] args) {
String s1="bhaskar";
String s2=s1.toUpperCase();
String s3=s1.toLowerCase();
System.out.println(s1==s2);//false
System.out.println(s1==s3);//true
}
}
```

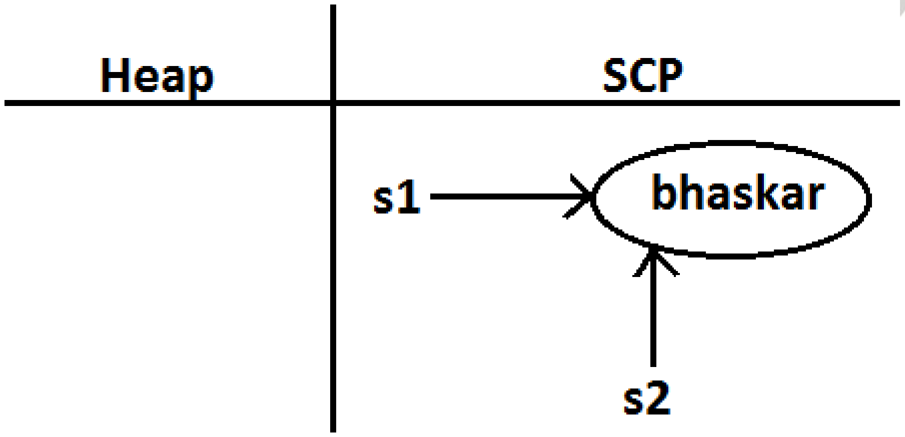
Diagram :



Example 2:

```
class StringDemo {

public static void main(String[] args) {

String s1="bhaskar";

String s2=s1.toString();

System.out.println(s1==s2);//true

}

}
```

Diagram :



```
class StringDemo {
```

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

String s1=new String("ashok");

String s2=s1.toString();

String s3=s1.toUpperCase();

String s4=s1.toLowerCase();

String s5=s1.toUpperCase();

String s6=s3.toLowerCase();

System.out.println(s1==s6); //false

System.out.println(s3==s5); //false

}

}
```

Final vs immutability :

1. final modifier applicable for variables where as immutability concept applicable for objects

2. If reference variable declared as final then we can't perform reassignment for the reference variable it doesn't mean we can't perform any change in that object.

3. That is by declaring a reference variable as final we won't get any immutability

nature .

4. final and immutability both are different concepts .

Example:

```
class Test

{

public static void main(String[] args)

{

final StringBuffer sb=new StringBuffer("ashok");

sb.append("software");

System.out.println(sb);//ashoksoftware

sb=new StringBuffer("solutions");//C.E: cannot assign a

value to final variable sb

}

}
```

In the above example even though "sb" is final we can perform any type of change in

the corresponding object. That is through final keyword we are not getting any

immutability nature.

Which of the following are meaning ful ?

1. final variable (valid)

2. final object (invalid)

3. immutable variable (invalid)

4. immutable object (valid)

## StringBuffer :

1. If the content will change frequently then never recommended to go for String object because for every change a new object will be created internally.

2. To handle this type of requirement we should go for StringBuffer concept.

3. The main advantage of StringBuffer over String is, all required changes will be performed in the existing object only instead of creating new object.(won't create

new object)

## Constructors :

1. StringBuffer sb=new StringBuffer();

Creates an empty StringBuffer object with default initialcapacity "16".

Once StringBuffer object reaches its maximum capacity a new StringBuffer

object will be created with

Newcapacity=(currentcapacity+1)*2.

Example:

```
class StringBufferDemo {
public static void main(String[] args) {
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//16
sb.append("abcdefghijklmnop");
System.out.println(sb.capacity());//16
sb.append("q");
System.out.println(sb.capacity());//34
}
}
```

2. StringBuffer sb=new StringBuffer(int initialcapacity);

Creates an empty StringBuffer object with the specified initial capacity.

Example:

```
class StringBufferDemo {
public static void main(String[] args) {
StringBuffer sb=new StringBuffer(19);
System.out.println(sb.capacity());//19
}
}
```

3. StringBuffer sb=new StringBuffer(String s);

Creates an equivalent StringBuffer object for the given String with

capacity=s.length()+16;

Example:

```
class StringBufferDemo {
public static void main(String[] args) {
StringBuffer sb=new StringBuffer("ashok");
System.out.println(sb.capacity());//21
}
}
```

## Important methods of StringBuffer :

1. public int length();

Return the no of characters present in the StringBuffer.

2. public int capacity();

Returns the total no of characters StringBuffer can accommodate(hold).

3. public char charAt(int index);

It returns the character located at specified index.

Example:

```
class StringBufferDemo {
public static void main(String[] args) {
StringBuffer sb=new StringBuffer("saiashokkumarreddy");
System.out.println(sb.length());//18

System.out.println(sb.capacity());//34

System.out.println(sb.charAt(14));//e

System.out.println(sb.charAt(30));//RE :

StringIndexOutofBoundsException

}

}
```

4. public void setCharAt(int index, char ch);

To replace the character locating at specified index with the provided character.

Example:

```
class StringBufferDemo {

public static void main(String[] args) {

StringBuffer sb=new StringBuffer("ashokkumar");

sb.setCharAt(8,'A');

System.out.println(sb);

}

}
```

5. public StringBuffer append(String s);

6. public StringBuffer append(int i);

7. public StringBuffer append(long l);

8. public StringBuffer append(boolean b); All these are overloaded methods.

9. public StringBuffer append(double d);

10. public StringBuffer append(float f);

11. public StringBuffer append(int index, Object o);

12. Example:

13. class StringBufferDemo {

14. public static void main(String[] args) {

15. StringBuffer sb=new StringBuffer();

16. sb.append("PI value is :");

17. sb.append(3.14);

18. sb.append(" this is exactly ");

19. sb.append(true);

20. System.out.println(sb);//PI value is :3.14 this is exactly true

21. }

22. }

23.

24. public StringBuffer insert(int index,String s);

25. public StringBuffer insert(int index,int i);

26. public StringBuffer insert(int index,long l);

27. public StringBuffer insert(int index,double d); All are overloaded methods

28. public StringBuffer insert(int index,boolean b);

29. public StringBuffer insert(int index,float f);

30. public StringBuffer insert(int index, Object o);

31. To insert at the specified location.

32. Example :

33. class StringBufferDemo {

34. public static void main(String[] args) {

35. StringBuffer sb=new StringBuffer("abcdefgh");

36. sb.insert(2, "xyz");

37. sb.insert(11,"9");

38. System.out.println(sb);//abxyzcdefgh9

39. }

40. }

41. public StringBuffer delete(int begin,int end);

To delete characters from begin index to end n-1 index.

42. public StringBuffer deleteCharAt(int index);

To delete the character locating at specified index.

Example:

class StringBufferDemo {

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

StringBuffer sb=new StringBuffer("saicharankumar");

System.out.println(sb);//saicharankumar

sb.delete(6,13);

System.out.println(sb);//saichar

sb.deleteCharAt(5);

System.out.println(sb);//saichr

}

}
```

43. public StringBuffer reverse();

44. Example :

```
45. class StringBufferDemo {

46. public static void main(String[] args) {

47. StringBuffer sb=new StringBuffer("ashokkumar");

48. System.out.println(sb);//ashokkumar

49. System.out.println(sb.reverse());//ramukkohsa

50. }

51. }
```

52. public void setLength(int length);

Consider only specified no of characters and remove all the remaining characters.

Example:

```
class StringBufferDemo {
public static void main(String[] args) {
StringBuffer sb=new StringBuffer("ashokkumar");
sb.setLength(6);
System.out.println(sb);//ashokk
}
}
```

53. public void trimToSize();

To deallocate the extra allocated free memory such that capacity and size are equal.

Example:

```
class StringBufferDemo {
public static void main(String[] args) {
StringBuffer sb=new StringBuffer(1000);
System.out.println(sb.capacity());//1000
sb.append("ashok");
System.out.println(sb.capacity());//1000
sb.trimToSize();
System.out.println(sb.capacity());//5

}

}
```

54. public void ensureCapacity(int initialcapacity);

To increase the capacity dynamically(fly) based on our requirement.

Example:

```
class StringBufferDemo {

public static void main(String[] args) {

StringBuffer sb=new StringBuffer();

System.out.println(sb.capacity());//16

sb.ensureCapacity(1000);

System.out.println(sb.capacity());//1000

}

}
```

Note :

Every method present in StringBuffer is syncronized hence at a time only one thread is

allowed to operate on StringBuffer object , it increases waiting time of the threads and

creates performence problems , to overcome this problem we should go for

StringBuilder.

## StringBuilder (1.5v)

1. Every method present in StringBuffer is declared as synchronized hence at a

time only one thread is allowed to operate on the StringBuffer object due to this,

waiting time of the threads will be increased and effects performance of the

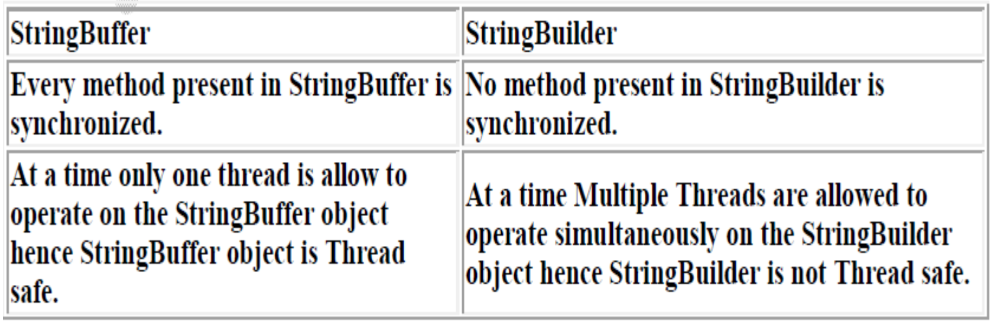
system.

2. To overcome this problem sun people introduced StringBuilder concept in 1.5v.

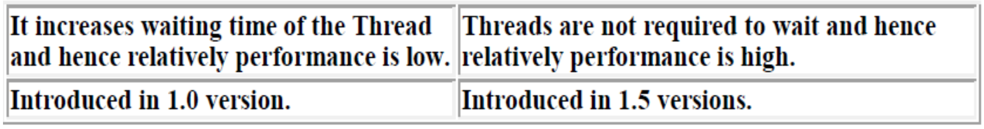
StringBuffer Vs StringBuilder

StringBuilder is exactly same as StringBuffer(includung constructors and methods )

except the following differences :



| StringBuffer | StringBuilder |
|---|---|
| Every method present in StringBuffer is synchronized. | No method present in StringBuilder is synchronized. |
| At a time only one thread is allow to operate on the StringBuffer object hence StringBuffer object is Thread safe. | At a time Multiple Threads are allowed to operate simultaneously on the StringBuilder object hence StringBuilder is not Thread safe. |



| | |
|---|---|
| It increases waiting time of the Thread and hence relatively performance is low. | Threads are not required to wait and hence relatively performance is high. |
| Introduced in 1.0 version. | Introduced in 1.5 versions. |

## String vs StringBuffer vs StringBuilder :

1. If the content is fixed and won't change frequently then we should go for String.

2. If the content will change frequently but Thread safety is required then we

should go for StringBuffer.

3. If the content will change frequently and Thread safety is not required then we

should go for StringBuilder.