Difference between String and StringBuffer
==========

String :
1) String are immutable
ex.,
```
String s = "Kajol";
s.concat(" Kolagir");
System.out.println(s);  //Kajol
//if u want to do changes then store the result in another
object
String res = s.concat(" Kolagir");
System.out.println(res);    //Kajol Kolagir
```
2) In string equals() method is used for content comparison and ==
operator for reference comparison.
ex.,
```
String s1 = new String("Kaju");
String s2 = new String("Kaju");
System.out.println(s1==s2); //false
System.out.println(s1.equals(s2));  //true
```

StringBuffer :
1) StringBuffer are mutable.
```
StringBuffer str = new StringBuffer("Kajol");
str.append(" Kolagir");
System.out.println(str);    //Kajol Kolagir
```
2) In string equals() method is used for content comparison and ==
operator for reference comparison.
ex.,
```
StringBuffer sb1 = new StringBuffer("Kaju");
StringBuffer sb2 = new StringBuffer("Kaju");
System.out.println(sb1==sb2);   //false
System.out.println(sb1.equals(sb2));    //false
```

Note :
Whenever we going to create object using new keyword in String the object
will created in heap
area and also the content will store in SCP area.
for ex., we hava
```
String s1 = new String("Kaju");
String s2 = new String("Kaju");
String s3 = "Kaju";
String s4 = "Kaju";
```
in this example two object will create in heap area for s1 and s2 and 1
content will store in SCP because
content is same of all 4 variables.

Example for String constant Pool(SCP) and heap memory :
==========

```
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(s4==s5);     //true

        String s6 = "You cannot ";
        String s7 = s6 + "change me";
        System.out.println(s4==s7); //false

        final String s8 = "You cannot ";
        String s9 = s8 + "change me";
        System.out.println(s4==s9);     //true

        Heap                SCP
        s1                  ("You cannot change me") s3, s4, s5(there is
constant value that's why), s9(s8 is constant we used there final keyword
that's why content will assign)
        s2                  ("You cannot ") s6
        s7(there is variable and constant value that's why new object
will create in heap area)


String Constant Pool (SCP) :
============================

Defination : In java memory allocation is divided into three parts i.e.
Heap, Stack and String Pool. The SCP is a special area used to store
string literals.

String literal : It is a string created using double quotes while String
object is a string created using the new() operator.
ex., String str = "Kaju"    //string literal
String str = new String("Kaju");    //string object

Note : String literals are created in the SCP and String objects are
created in the heap memory area.

Suppose we want to create two string with same content "Kaju". If a
string with "Kaju" already exist, the new literals will be pointing to
the already existing literal. In the case of
String objects, a new String object will be created in the heap every
time.
ex.,
        String a = "Kaju";
        String b = "Kaju";
        String c = new String("Kaju");
        System.out.println(a==b);   //true
        System.out.println(a==c);   //false

String.intern() API :
*********************
```

We can yse the method String.intern() to create string literals for the String objects. When invoked on a string object, method intern() creates an exact copy of a string object in the heap memory and stores it in the SCP.
ex.,

```
String a = "Kaju";
String b = "Kaju";
String c = new String("Kaju");
String d = c.intern();
System.out.println(a==b);   //true
System.out.println(a==c);   //false
System.out.println(a==d);   //true
```

Advantage :
****
1) Enhanced Security : SCP allows being string immutable. Immutable objects help in making the application more secure because they may store sensitive information.
2) Thread Safety : Any immutable object is thread-safe by its nature This means multiple threads can share and manipulate string without risk of corruption and inconsistency.

Disadvantage :
****
1) String class cannot be extended : If we want to extend the string class to add more features, we cannot do it. An immutable class is declared final to avoid extensibility.
2) Sensitive data in memory for a long time : SCP takes advantage of special treatment from the Garbage Collector(GC). The GC does not visit SCP with the same frequency as other memory zones.
Due to this , sensitive data is kept in the SCP for a long time and can br prone to unwanted usage. To avoid this potential drawback, it is advisable to store sensitive data (ex., passwors) in char[] instead String.
3) Possible OutOfMemoryError : SCP is a small memory zone in comparison with others and can be filled pretty quickly. Stroing too many string literals in the SCP will lead to OutOfMemoryError.

Important FAQ :
1) Why SCP concept is available only for String object but not for StringBuffer?
=> String is like regular customer and StringBuffer not that much so. Most commonly used object in java is String object and rarely used object is StringBuffer that's why Java people provided concept SCP for string.

2) Why String objects are immutable where as StringBuffer objects are mutable?
=> In the case of String just because of SCP same object can be reused multiple time tha's why String is immutable and StringBuffer not support SCP concept that's why it's mutable.

3) In addition to string objects any other objects are immutable in java?
=> Yes. all wrapper class objects are also immutable

```
IMP Constructor of String Class :
*********************************
1) String s = new String();     //empty string object
2) String s = new String(string literal);
3) String s = new String(StringBuffer sb);  //for given stringbuffer
string object
4) String s = new String (StringBuilder sd);    //for given stringbuilder
string object
5) String s = new String(char[] ch);    //for given char array string
object
6) String s = new String(byte[] b);     //for byte array
ex.,
        //String str = new String()
        String str1 = new String();
        System.out.println(str1);

        //String str = new String(string literal)
        String str2 = new String("kaju");
        System.out.println(str2);   //kaju

        //String str = new String(StringBuilder sb)
        StringBuilder sd = new StringBuilder("Kaju");
        String str3 = new String(sd);
        System.out.println(str3);   //Kaju

        //String str = new String(StringBuffer sb);
        StringBuffer sb = new StringBuffer("Kaju");
        String str4 = new String(sb);
        System.out.println(str4);   //Kaju

        //String str = new String(char[] ch);
        char[] ch = {'k', 'a', 'j', 'u'};
        String str5 = new String(ch);
        System.out.println(str5);    //kaju

        //String str = new String(byte[] b)
        byte[] b = {97, 98, 99, 100};     //byte range is -128 to 127
        String str6 = new String(b);
        System.out.println(str6);   //abcd (it will convert byte into
char)


IMP Methods of String :
***********************

        //1) public char charAt(int index)      //return character of
specific position
        String str = "Kaju";
        System.out.println(str.charAt(3));      //u

        //2) public String concat(String s)     //to do concadenation of
two string
        String conc = str.concat(" kolgair");
```

```
        System.out.println(conc);   //kaju kolgir

        //3) public boolean equals(String s)    //it is used for content
comparison and case is important
        System.out.println(str.equals("Kaju"));     //true

        //4) public boolean equalsIgnoreCase(String s)  //used for
content comparison but ignoring the cases
        System.out.println(str.equalsIgnoreCase("kaju"));   //true

        //5) public boolean isEmpty();      //it will return string is
empty or not
        System.out.println(str.isEmpty());  //false

        //6) public int length()    //return the size or length of the
string
        System.out.println(str.length());   //4

        //7) public String replace(char oldChar, char newChar)
//replace
        System.out.println(str.replace('u', 'i'));      //Kaji

        //8) public String substring(int begin);     //return substring
        System.out.println(str.substring(2));       //ju

        //9) public String substring(int begin, int end)    //return
substring from begin index to end-1 index
        System.out.println(str.substring(1, 3));    //aj

        //10) public int indexOf(char c)    //return index of specific
character. if char is not present then it will return -1.
        //and if character is present multiple times it will return 1st
occurence index
        System.out.println(str.indexOf('K'));   //0

        String s = "KKK";
        //11) public int lastIndexOf(char c)    //return last index
occurence of specific character
        System.out.println(s.lastIndexOf('K')); //2

        //12) public String toLowerCase()   //it will return string in
lowercase
        System.out.println(s.toLowerCase());    //kkk

        //13) public String toUpperCase()   //it will return string in
uppercase
        System.out.println(s.toUpperCase());    //KKK

        //14) public String trim();     //it will remove the spaces of
starting and ending of the string
        String tr = "  kk  ";
        System.out.println(tr.trim());      //kk

IMP conclusion about the String immutability :
```

```
*******************************************

                String s1 = new String("kaju");
                String s2 = s1.toUpperCase();
                String s3 = s1.toLowerCase();

                //in this example s1 and s2 object reference will store
in heap area and in SCP area 'kaju' will store
                //for s3 and s1 is same content that's why for s3 new
object reference will not create. it will refere same s1 object reference
                System.out.println(s1==s2);     //false
                System.out.println(s1==s3);     //true



How to create our own Immutable(don't allow to change content) class :
**********************************************************************
*

Example.,
final class Test {
    private int i;

    Test(int i) {
        this.i=i;
    }

    public Test modify(int i) {
        if(this.i==i) {
            return this;
        }
        else {
            return new Test(i);
        }
    }

    public static void main(String[] args) {
        Test t1 = new Test(10);
        Test t2 = t1.modify(10);    //here if we are modifying object
with same content then we will get previous object reference
        Test t3 = t1.modify(100);   //we will get new object reference

        System.out.println(t1==t2);     //true
        System.out.println(t2==t3);     //false
    }
}


Final VS Immutability :
***********************
Final :
1) final means not immutablility. You can get through below example
        final StringBuffer sb = new StringBuffer("kajol");
        sb.append(" kolgir");
```

```
        System.out.println(sb);     //kajol kolgir
```

Immutability :

Examples :

1)

```
public static void main(String[] args) {
        String ta = "A";
        ta = ta.concat("B");
        String tb = "C";
        ta = ta.concat(tb);
        ta.replace("C", "D");
        ta = ta.concat(tb);
        System.out.println(ta);     //ABCC

        //working : ta object will create A (GC)
                    ta object will refer AB (GC)
                    tb new object will create C (GC)
                    ta will refer ABC   (GC)
                    we not storing ta.replace() so it will create object
but not refer for any variable. so it will be eligible GC
                    ta object will refer ABCC

}
```

2)

```
public static void main(String[] args) {
        String str = " ";
        System.out.println(str.length());
        str.trim();
        System.out.println(str.equals(""));     //false
}
```

3)

```
public static void main(String[] args) {
        String s = "Kaju Kolgir";
        int len = s.trim().length();
        System.out.println(len);    //11
}
```

4)

```
public static void main(String[] args) {
        String s = "Kajol Kolgir";
        s.trim();
        int index = s.indexOf(" ");
        System.out.println(index);      //5     (indexing will start from
0)
}
```

5)

```
public static void main(String[] args) {
        String s1 = "Kajol";
        String s2 = new String("kajol");
        //which statememnt will be here
```

```
        if(s1.equalsIgnoreCase(s2))
        {
            System.out.println("Equal");
        }
        else {
            System.out.println("Not Equal");
        }
        //op : Equal
}
```

Need of StringBuffer :
****

1) if you want to change content in string then go for StringBuffer because it's a mutable
2) all changes will be perform only existing object. new object reference will not created
ex.,

```
public static void main(String[] args) {
        String s2 = new String("kajol");
        s2 = "kolgir";
        System.out.println(s2==(s2 = "kolgir"));    //true
}
```

Important constructor of StringBuffer :
****

1) StringBuffer sb = new StringBuffer();
default capacity is 16 and newCapacity = (CurrentCapacity + 1) * 2;
ex.,

```
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);
- here we can set the capacity

3) StringBuffer sb = new StringBuffer(String s);
In this capacity will be (s.length()+16)
ex.,

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

Methods of StringBuffer :
****

```
 public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("Kajol");

        //1) public int length();
        System.out.println(sb.length());        //5

        //2) public int capacity();
        System.out.println(sb.capacity());      //21

        //3) public char charAt(int index);
        System.out.println(sb.charAt(0));
        //System.out.println(sb.charAt(5));
//StringIndexOutOfBoundsException

        //4) public void setCharAt(int index, char newChar)
        sb.setCharAt(3, 'u');
        System.out.println(sb);     //kajul

        //5) public StringBuffer append(String s);      //in argument you
can take any type of argument (Overloaded method)
        //this method will add data at the end of the string
        System.out.println(sb.append(" Kolagir"));      //Kajul Kolagir

        //6) public StringBuffer insert(int index, String s)      //here
you can add any type of value (Overloaded method)
        System.out.println(sb.insert(0, "Kolagir "));   //Kolagir Kajul
Kolagir

        //7) public StringBuffer delete(int beginIndex, int endIndex)
//it will delete string form begin index to endIndex-1
        System.out.println(sb.delete(0, 8));        //Kajul Kolagir (0 to
7 string will delete)

        //8) public StringBuffer deleteCharAt(int index);       //delete
char at specific position
        System.out.println(sb.deleteCharAt(4));     //Kaju Kolagir

        //9) public StringBuffer reverse();
        System.out.println(sb.reverse());       //rigaloK ujaK

        //10) public StringBuffer setLength();      //if we write the
string big length and you want to specific size string then we can use
        sb.setLength(4);
        System.out.println(sb);     //riga

        //11) public void ensureCapacity(int capacity)  //increase
dynamically capacity
        //for ex., already you declared string and after you want to
increase capacity then we can use this method
        sb.ensureCapacity(1000);
        System.out.println(sb.capacity());      //1000

        //12) public int trimToSize();     //to delete the extra capacity
which is not need for us
```

```
        sb.trimToSize();        //delete the extra space capacity
        System.out.println(sb.capacity());      //4
 }
```

Need of StringBuilder :
****
- every method in StringBuffer is synchronized (only one thread allowed at a time)
- to overcome above problem in 1.5 version java people introduced StringBuilder concept.
(they did like, replaced Buffer with Builder and removed synchronized keyword from method) otherwise all things are same like StringBuffer

Difference between StringBuffer and StringBuilder :
****

StringBuffer :
1) Every methods present inside StringBuffer are Synchronized
2) At a time only one thread is allowed to operate on StringBuffer object and hence it is thread safe
3) Threads are required to wait to operate on StringBuffer object and hence relativerly performance is slow
4) introduced in 1.0 version

StringBuilder :
1) Every methods present inside StringBuilder are Non-Synchronized
2) At a time multiple threads is allowed to operate on StringBuilder object and hence it is not thread safe
3) Threads are not required to wait to operate on StringBuilder object and hence relativerly performance is high
4) introduced in 1.5 version

Some IMP Examples :
1)

```
public static void main(String[] args) {
        StringBuilder sb = new StringBuilder(5);
        String s = "";
        if(sb.equals(s)) {
            //false   //StringBuilder doesn/t override equlas method that's why here
            //object reference comparison will happen not a content comparison
            System.out.println("Match 1");
        }
        else if(sb.toString().equals(s.toString())) {  //true     //here both are string so content comparison will happen
            System.out.println("Match 2");
        }
        else {
            System.out.println("No match found");
        }
```

```
         //OP : Match 2

 }

 2)
public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Kaju");
        String str1 = sb.toString();
        String str2 = str1; //true
        //String str2 = new String("Kaju");   //false
        //String str2 = sb.toString();    //false
        //String str2 = "Kaju";   //false
        System.out.println(str1==str2);
}

3) Which statement will empty the contents of a StringBuilder variabled
named sb?
i) sb.deleteAll();      //deleteAll method is not there for StringBuilder
ii) sb.delete(0, sb.size());    //size method is not there for
StringBuilder
iii) sb.delete(0, sb.length());     //this is correct
iv) sb.removeAll(); //removeAll method is not there for StringBuilder

4)
class Test {

    String msg;

    public Test() {

    }
    public Test(String msg) {
        this.msg = msg;
    }

    public static void main(String[] args) {
        Test t1 = new Test();
        System.out.println(t1);     //String.Test@58372a00

        System.out.println("Hello"+new StringBuilder(" Kajol"));
//Hello Kajol
        System.out.println("Hello"+new Test(" Kajol"));
//HelloString.Test@378bf509

    }
}


5)  You are developing a banking module you have developed named Test
that has a mask method
You must have ensure that mask method returns String that hides all
digits of the credit card number
except last four digits and hypens that seperate each froup of 4 digits
```

```
class Test {

    public static String mask(String creditCard) {
        String x = "xxxx-xxxx-xxxx-".toUpperCase();
        StringBuilder sb = new StringBuilder(creditCard);
        //return x+sb.substring(15, 19);
        //return x+creditCard.substring(15, 19);

        StringBuilder sb2 = new StringBuilder(x);
        sb2.append(creditCard, 15, 19);
        return sb2.toString();
    }

    public static void main(String[] args) {
        System.out.println(mask("1234-3456-1234-6543"));
    }
}
```