**String /String buffer:**

String is Immutable (Not changeable) and StringBuffer is Mutable(Changeable)

Once the object is created for String, no changes can be made but whereas if the object is created in StringBuffer, changes can be made to the object.

String s=new String(“Divya”);

s.concat(“Study”);

sysout(s);

O/p: Divya

As there is no reference given to s.concat(“Study”), this change is not stored in any reference for String object. Hence the output is Divya.

StringBuffer sb=new StringBuffer(“Divya”);

sb.append(“Study”);

sysout(sb);

O/p: DivyaStudy

Though there is no reference given to s.append(“Study”), since StringBuffer is mutable, this change is stored in the same reference for StringBuffer object. Hence the output is DivyaStudy.

**Difference between “==” and .equals(String s):**

String s1=new String(“Divya”);

String s2=new String(“Divya”);

Sysout(s1==s2);//false

sYsout(s1.equals(s2));//true

== is the reference comparison, if the two references point to same object, then the == returns True otherwise false. Here two new objects are created with same content. As the references are pointing to two new objects, the output for == is false

.equals(s2) is also the reference comparison in Object class but equals() method is overridden in String class which contains content comparison. Hence the output for String object with equals() method does content comparison and the outpot for sYsout(s1.equals(s2)); is True.

StringBuffer sb1=new StringBuffer(“Divya”);

StringBuffer sb2=new StringBuffer(“Divya”);

Sysout(sb1==sb2);//false

sYsout(sb1.equals(sb2));//false

Here == is the reference comparison. Equals() method is not overridden in StringBuffer and the method is called from Object class which contains the reference comparison logic. As both objects are pointing to two different references, output for .equals is also false.

**String s=new String(“Divya”);**

In this case two objects will be created. One in Heap memory and one in SCP(String constant pool).The object created in scp is for future usage.

The object in heap memory point to the reference s and the object in SCP is referred to the implicit reference that is in JVM.

**String s=”Divya”;**

In this case, no new object will be created in heap memory but an object is created in SCP only if that particular object is not there before. If the object is already present, then the reference s will be pointed to the already created object in scp.

**Examples:**

1. String s1=new String(“Divya”);

String s2=new String(“Divya”);

In this case, two new objects will be created in Heap memory and one object if it is already present in scp, no object will be created. So total three objects will be there in this case

String s1=”Divya”;

Strin s2=”Divya”;

In this case, no object will be created in Heap memory but in scp, if the object is not present, then an object will be created. So only one object will be created here.

1. String s=new String(“Divya”);

s.concat(“Software”);

s=s.concat(“Solutions”);

|  |  |
| --- | --- |
| **Heap** | **SCP** |
| Divya=>s>>GarbageCollection as now s points to DivyaSolutions | Divya |
| DivyaSoftware>>GarbageCollection as there is no reference | Software |
| DivyaSolutions=>s | Solutions |

1. String s1=new String(“Spring”);

s1.concat(“Fall”);

String s2=s1.concat(“Winter”);

s2.concate(“Summer”);

sysout(s1);

sysout(s2);

|  |  |
| --- | --- |
| **Heap** | **SCP** |
| Spring=>s1 | Spring |
| SpringFall>>As there is no reference, this is for GC | Fall |
| s2=>SpringWinter | Winter |
| SpringWinterSummer>>As there is no reference hence to GC | Summer |

o/p: Spring

SpringWinter

1. String s1=new String(“You cannot change me”);

String s2=new String(“You cannot change me”);

sysout(s1==s2);//false(two references for two diff objects. Hence false)

String s3=”You cannot change me”;

Sysout(s1==s3);//false(Again two diff references and two objects. One in heap and another in scp)

String s4=”You cannot change me”;

Sysout(s3==s4);//true(s3 and s4 referes to single object )

String s5=”You cannot”+”change me”;//This operation is performed during compile time and hence its exactly same as “You cannot change me”

Sysout(s4==s5);//true

String s6=”You cannot”;//Since this is string constant, this gets created in scp

String s7=s6+”change me”;//This is done during run time, hence the object gets created in heap

Sysout(s4==s7);//false

Final String s8=”You cannot”;//Every final variable will be executed during compiler time

String s9=s8+” change me”;//So this is same as “You cannot change me”

Sysout(s4==s9);//true

|  |  |
| --- | --- |
| **Heap** | **SCP** |
| You cannot change me=>s1  You cannot change me=>s2 | You cannot change me=>s3, s4,s5,s9 |
| You cannot change me=>s7 | You cannot=>s6,s8 |

Advantages and Disadvantages of SCP:

Memory utilization is less and performance is high because, many references can be refered to single object.

Disadvatage is if a change needs to be done on that single object, then all other references would be effected with the change. Hence the concept of immutability for String and the new object with the change will be created and the reference will be given to that so that only required reference will be effected and all other references would be the same.

v1,v2,v3…v10000=>Hyderabad

If v3 wants to change the city from Hyd to Vijaywada, then a new object will be created with reference v3 and other references will be the same.

Constructors of String:

String s=new String();>>Empty string object

String s=new String(String literal);>>with a string

String s=new String(String buffer sb);

Strinf s=new String(String builder sd);

String s=new String(char[] c);

Eg: char[] c={‘j’,’a’,’v’,’a’};

String s=new String(c);

Sysout(s);

O/p: java

String s=new String(byte b);

Eg: byte b={96,97,98,99}

String s=new String(b);

Sysout(s);

o/p: abcd

Methods:

s.replace

s.substring(int initialindex);

s.substring(int initial,int lastindex);>>3,6 then it will be index 3 to 5

s.indexOf(‘g’);>>3

If the character is not present in string, it returns -1

If the character is repetitive, it returns first occurance of the character

s.lastindexof(‘g’);>>gives the last occurance

s.tolowercas();

s.toUpperCase();

String s1=new String(“divya”);

String s2=s1.toLowerCase();

String s3=s1.toUppercase();

Sysout(s1==s2);//True

Sysout(s1==s3);//False

|  |  |
| --- | --- |
| **Heap** | **SCP** |
| divya=>s1,s2 | divya |
| DIVYA=>s3 |  |

String s1=”divya”;

String s2=s1.toString();

String s3=s1.toLowerCase();

String s4=s1.toUppercase();

Sysout(s1==s2);//T

Sysout(s1==s3);//T

Sysout(s1==s4);//F

|  |  |
| --- | --- |
| **Heap** | **SCP** |
| DIVYA=>s4 | divya=>s1,s2,s3 |
|  |  |

Create own Immutable class:

Final class Test{

Private int i;

Test(){

This.i=I;

}

Public Test modify(int i){

If(this.i==i)

Return this.i

Else

Return new Test(I);

}

Public static void main(String[] args){

Test t1=new Test(10);

Test t2=t1.modify(100);

Test t3=t1.modify(10);

Sysout(t1==t2);//false

Sysout(t1==t3);//true

}

}

Can StringBuffer change to immutable? No

Final is declared for a variable or reference. When final is declared then the same variable/reference cannot be used for other object.

Immutability is the concept of object.

If an object is immutable, then object cannot be changed.

Class Test{

Psvm(String[] args)

{

**Final** StringBuffer s1=new stringBuffer(“divya”);

S1.append(“Study”);

**Sysout(s1);//divyaStudy**

**S1=new StringBuffer(“Other”);//compiletime error**

}

}

String can be used in applications where content is not changed like college name/address of the house etc. For every new change, the new object is created

StringBuffer is used when the content is changed frequently. For every change, the content will be saved in the same ref object

Constructors of StringBuffer:

StringBuffer sb=new StringBuffer();=>Empty stringbuffer, default capacity is 16

If the string is more than 16 characters, then the new stringbuffer object will be created with capacity as (current capacity+1)2=17\*2=34

StringBuffer sb=new StringBuffer(String Literal);=>StringBuffer with the string

Eg: StringBuffer sb=new StringBuffer(“divya”);=>capacity is string.length+default capacity=5+16=21

StringBuffer sb=new StringBuffer(int initialcapacity)=>More than 16

StringBUffer sb=new StringBuffer(String);

Methods of StringBuffer:

* Public int length()
* Public int capacity()
* Public char charAt(int index)

Eg: SB sb=new SB(“divya”);

Sb.charAt(3)//y

Sb.charAt(10)//StringIndexOutOfBoundsException

* Public void setCharAt(int index,Char newChar)

Eg: SB sb=new SB(“java”)

Sb.setCharAt(0,’y’);//yava

* Public StringBuffer append(String s)

(int i)

(float f)

(double d)

(long l)

(byte b)

This is method overloading

* Public StringBuffer insert(int Index, String s)

(int index, int i)

(int index, float f)

….

Overloading

* Public StringBuffer delete(int begin,int end)

Begin index to end-1 index, the chars/string will be deleted

* Public StringBuffer deletCharAt(int index)
* Public StringBuffer reverse()=>Reverses the string
* Public void setlength(int i)

Eg: SB sb=new SB(AiswaryaAbhi);

Sb.setLength(8)//Aiswarya

* Public void ensurelength(int i)=>increases the length
* Public void trimtosize()=>cuts down the extra spaces if not required

Difference between String, String Buffer and String Builder

String: If the content is fixed, then we go for String and this is synchronized

StringBuffer: If the content is nt fixed. Then we can go for stringbuffer. This is threadsafe and all methods are synchronized. Only one thread acts at a time.

StringBuilder: Same as stringBuffer but this is not threadsafe and methods are not synchronized

Method chaining: sb.append(“divya”).append(“devi”).reverse().delete(3,5)

Everymethod returns StringBuffer and these methods ca be used continuously.