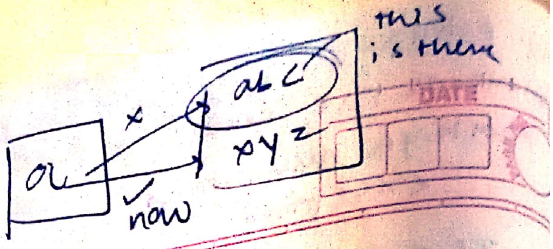


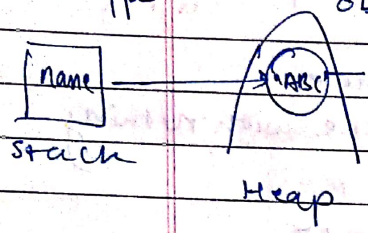
String

- a datatype
- final class in java.

String name = "ABC"

↓ ref datatype

instance/ obj



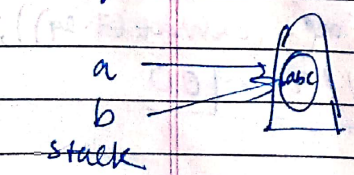
Pool :- Is a memory structure inside heap called as string pool.

Why pool?

→ No need to create another factor in heap rather point to the same in existing pool.

String a = 'abc'

String b = 'abc'



As string is immutable?

- for security reasons

String a = "abc";

Print(a); //abc

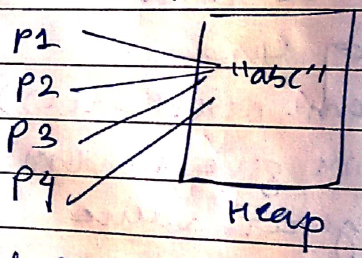
a = "xyz"

Print(a) //xyz

Here 'a' is same object. so you can modify it but

String a =

Why we can't modify string object?



If P1 decides to change P1 → xyz all other objects pointing to it will change to xyz.

So string cannot be modified.

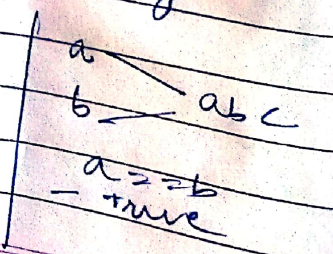
Comparison of strings.

a → "abc"

b → "abc"

a == b

- false



a == b

- true

== → comparator

If a = "abc"

a.charAt(0) → as it uses internally char[]
= a

==
↳ checks if any variable are pointing to same object.

Now there is one question
Q How to create diff object of same value.
→
String a = new String ("abc")
String b = new String ("abc")

now objects are created outside pool but inside heap due to "new" keyword.

Q when need to check only value use .equals to method.
String a = "abc"
b = "abc"
String c = a;
(c == a) // true
but (a == b) // false
(a.equals(b))
// true
because compared values.
also
"abc"
"abc "
↳ not equal

whatever you type in Sysout i.e
System.out.println( )
any datatype internally call to string and prints value.

How does printf works?
printf("%.2f", a)
and float a = 453.12324f;
// 453.12

printf("Pie: %.3f", Math.PI);
// Pie: 3.142

printf("Hello %s and %s", "abc", "xyz");
→ O/P → Hello abc and xyz.

String concatenation operators
sop('a' + 'b')
// 195 — why?
adding ascii of a + ascii value
sop("a" + "b")
OP → ab.

printf("a" + 1);
// Integer will be converted to Integer and it calls to string
So O/P = a1

Internal working

Print("abc"+new Integer(56));

OP → abc56;

'+' operator can be used any with primitives & complex object but the condition is that one of them must be string.

Sysout(new Integer(56) + new ArrayList<>());

→ // compile time error.

(new Integer(56) + "" + new ArrayList<>();

// OK

OP = 56[]

this is called as operator overloading.

New string performance:-

do when

String series = "";

for(int i=0; i<26; i++)

char ch = (char)('a'+i);

series = series + ch;

so this is with print abcd.....yz.

but for loop first iteration will be

'a' + 'b' = 'ab'

'ab' + 'c' = 'abc'

→ 'abc' + 'y' + z

final result

but as string is immutable there will all of references created in heap in string pool

→ this is wastage of memory right?

complexity

→ $O(N)^2$

$\frac{1+2+\dots+N}{N}$

$\frac{N(N+1)}{2}$

$O(N^2)$

So we need a datatype that will not create new object but just add "abc" + 'd'

→ abcd

★ So we use StringBuilder which is class

String name = " abc xyz";
sop(Arrays.toString(name.toCharArray()));
→ [a, b, c, , x, y, z] space also set index.

StringBuilder builder
= new StringBuilder();
for(int i=0; i<26; i++){
char ch = (char)('a'+i);
builder.append(ch);
}
sop(builder.toString(ch));

It also has methods such as,
append, rename, replace

Methods are
- toCharArray()
- indexOf('a')
// ↑ returns index at string where 'a' is lying
name.strip
// → KBh
sop(" KBh ".strip())
→ KBL - spaces are removed.
- split()
(Array.toString(name.split(" ")))
' split into spaces.

Competetive Questions

P1 → palindrome logic

(1) reverse the string and just compare if original string is same or not

P2 approach to palindrome

a b c d e b a
s e

if s = e,
then s++,
e--

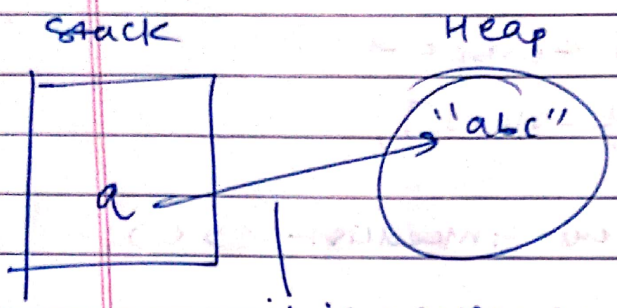
for(int i=0, i<=str.length()/2; i++)
{
char start = str.charAt(i);
char end = str.charAt(str.length()-1-i);
if(s != e)
return false

String Buffer Class:

Lets revise String:-

String a = "abc";

a = "xyz";



it is pointing to abc

but when we change literal value

it will point to new object.

Immutable

this is still in memory

As strings are immutable memory issue

So we use String buffer class

Advantages

1. Mutable.
2. Efficient
3. Thread Saye

If t1 t2

working on same object so t1 will ~~work~~ on object & prevent t2 to mess up with it.

StringBuffer
has constructor

// Constructor 1:
StringBuffer sb = new StringBuffer();
// constructor 2:
StringBuffer sb2 = new StringBuffer ("XYZ");
i.e can pass string as argument
*
// Constructor 3
StringBuffer sb3 = new StringBuffer (30);
// 30 → capacity set.

[Methods for constructor no 1.]

eg sb.append ("ABC is");
sb.append ("Good");
sb calling to parameterless const.
O/P → (ABC is Good)
i.e did not change anything and just appended that value in last.

sb.insert (2, "PQR");
cont A → 0
B → 1
C → PQR
O/P :- AB space PQR space C is good

sb.replace (1, 5, "XYZ")
ABC → XYZ replaced
5y
5 is index is not included.
0 → 4 — tak hi jaayega.

→ sb.delete (1, 5);
from 1 to 4 will be deleted.

* sb.reverse ();
// reverse the string

→ How to check capacity of StringBuffer
sb.capacity();
// 16 — default

```
package org.StringBuffer.exam;

import java.util.Arrays;

public class Program {
	public static void main(String[] args) {
		//Constructors
		StringBuffer sb = new StringBuffer();//default constructor
		//capacity is ig 16.
		StringBuffer sb2 = new StringBuffer("I am good");
		System.out.println(sb2);//I am good
		StringBuffer sb3 = new StringBuffer(50);

		//Methods
		sb.append("I am ");
		sb.append("so cool");
		System.out.println(sb);
		//I am so cool

		//sb.replace(0,"Parthavi");//error
		//need to specify upto what index

//		sb.replace(0, 5, "XYZ");
//		System.out.println(sb);//XYZso cool
//		//counted spaces too.

		sb.replace(0, 4, "XYZ");
		System.out.println(sb);
		//XYZso cool
		//XYZo cool -----> as referring to the same reference .
```

```
sb.delete(1,5);//Xo cool
sb.reverse();//looc oX
//can also use toString method.
String str = sb.toString();
System.out.println(str);

/*
Source --- Java docs
For example, if z refers to a string buffer object whose current contents are "start",
then the method call z.append("le") would cause the string buffer to contain "startle",
whereas z.insert(4, "le") would alter the string buffer to contain "starlet".


passing a null argument to a constructor or method
in this class will cause a NullPointerException to be thrown.
*/

        //replace
String sent = "he llo x y z how are y ou ";
System.out.println(sent);
//sb.replaceAll ..... error deta hai yeh
System.out.println(sent.replaceAll("h", "a"));
//ae llo x y z aow are y ou
//String s = sent.replaceAll("//s", " ");//nochange as // is used
String s = sent.replaceAll("\\s", "");
System.out.println(s);//helloxyzhowareyou

//split
String arr = "hi hello oh good bye sweet";
String[] b = arr.split(" ");
```

```
//String[] b = arr.split(""); ----- "" no space splits every element
System.out.println();
System.out.println(Arrays.toString(b));
//[hi, hello, oh, good, bye, sweet]
for(String st : b)
	System.out.println(st);
/*
hi
hello
oh
good
bye
sweet
*/

String s6 = "";
for(int i =1 ; i<26 ; i++ )
{
	System.out.println('a' + i);
}
	/*
	98
	99
	100
	101
	102
	103
	104
	105
	106
	107
```

```
                    108
                    109
                    110
                    111
                    112
                    113
                    114
                    115
                    116
                    117
                    118
                    119
                    120
                    121
                    122
                    */

            for(int i = 0 ; i<26 ; i++)
                    System.out.print((char)('a' + i));
            //abcdefghijklmnopqrstuvwxyz

        }

}
```

```
package org.StringBuffer.exam;

public class Palindrome {
```

```
	static boolean palindrome(String str) {
		//for(int i=0 ; i<=str.length(); i++) -------> error because out of bounds
		for(int i=0 ; i<=str.length()/2; i++)
		{
			if(str ==null||str.length()==0)
			{
				return true;
			}
			char start = str.charAt(i);
			char end = str.charAt(str.length()-1 -i);
			if(start != end)
			{
				return false;
			}

		}
		return true;
	}
	public static void main(String[] args) {
		//String s9 = "madam";//true
		String s9 = "";//true
		System.out.println(palindrome(s9));

	}

}
```

Difference of String , String buffer, String builder

SUMMARY TABLE:

Feature	String	StringBuffer	StringBuilder
Mutability	Immutable	Mutable	Mutable
Thread Safe	Yes (Immutable)	Yes (Synchronized)	No
Performance	Slow (new object on each modification)	Slower (due to synchronization)	Fastest (no synchronization)
Usage	Use when string doesn't change often	Use in multi-threaded environments	Use in single-threaded environments