(ii) Public Boolean Equals (Object o) :-

14 4 NOV class

Eg :-①
```
String s = "Java";
System.out.println(s.Equals("JAVA"));
                                  ↳ false
```

→ Equals helps to Compare Content in the string.

→ it Maps Care of letters.

Eg ②
```
String s = "Java";
S.o.p(s.Equals("Java"); // true
S.o.p(s.Equals("JAVA"); // false
                  ↳ it checks Care of letters.
```

→ checking only data not Case. of string object

(iii) Public boolean EqualsIgnoreCase(String s)

↳ ignore checking Care of data. it only checks. data.

↳ Return type of EqualsIgnoreCare is Boolean.

System.

Eg ①
```
String s = "java";
S.o.p(s.EqualsIgnoreCase("JAVA"));
```
o/p : true

→ it checks only data but not Care of data.

⇒ Practical Example of Equals() and EqualsIgnoreCase() Method

Credentials (gmail)

username : ashishpaul1199@gmail.com
↳ Case Sensitive

Password : * * * * * * * (Case Sensitive)

Code :-
```
String u = "ashishpaul99";
String p = "Ashish@777";
Scanner Scan = new Scanner(System.in);
// username
S.o.p("Enter username");
String username = scan.nextLine();
S.o.p(username.EqualIgnoreCase(p));
// Password
S.o.P("Enter Password");
String Password = Scan.nextLine();
S.o.p(Password.EqualsIgnoreCase(p));
```

④ Public String Substring(int begin) &

Public String Substring(int begin, int End)

Sachin IND MI
0 1 2 3 4 5 6 7 8 9 10

```
String s = "SachinIND MI";
s.o.p(s.length());
s.o.p(s.substring(9)); // MI
s.o.p(s.substring(0,8)); // SachinIN
```

search from index 9 to last

Starts at 0 and Ends End-1 Character

↳ it prints End-1 Characters.
↳ it stops at 7th index

```
s.o.p(s.substring(0,9)); // SachinIND
```

↳ it starts from 0th index and Ends at 8th index

⑤ Public String replace (char old, Char new)

① Replacing character in a string

```
String s = "sbchin";
System.out.println(s.replace('b','a'));
```

output :- Sachin (b is replaced by a)

② Replacing Every occurance with some Character.

```
String s = "abab ab";
s.o.p(s.replace('a','b'));
```

↳ all a's are replaced by b's

output :- bbbb bb

⑥ Public String toLowerCase() and toUpperCase()

① Converting Mixed Case data to upper Case :-

```
String s = "sAchIn";
s.o.p(s.toUpperCase());
```

object with no reference

→ here String is immutable After uppercasing it will not store in s variable.

→ till printing object is heap area with no reference.

→ object have no control by JVM after printing.

② Converting Mixed Case data to lower Case :-

```
String s = "sAchIn";
s.o.p(s.toLowerCase());
```

output :- sachin

⑦ Public String trim() :-

Eg ①

```
String name = "Sachin IND";
              0 1 2 3 4 5 6 7 8 9
s.o.p(name.length()); //10
s.o.p(name.trim());
```

output :- Sachin IND

→ trim will not remove Spaces in between.

Eg ②

```
String s = "  Sachin  ";
            0 1 2 3 4 5 6 7
             ↓ Space   ↓ Space
s.o.p(s.trim());
```

o/p :- Sachin
0 1 2 3 4 5

→ trim() method trims begging and End of the String.

→ trim() Method :- to the blank Spaces Present at the begging and End of String but not the blank Space present at the Middle of th string.

⑧ Public int indexOf(char ch) and Public int lastIndexOf(char ch) :-

```
String name = "hyder Abbas";
s.o.p(name.length());
s.o.p(name.indexOf('A'));
```

output : 5 (index of 'A')

hyder Abbas
0 1 2 3 4 5 6 7 8 9

→ it chek for character and gives it index and further it will not proced.

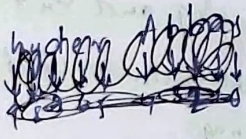
hyder Abbas
0 1 2 3 4 5 6 7 8 9

```
String s = "hyder Abbas";
s.o.p(s.indexOf('b'));
```

o/p :- 6 (index of b 1st b");

→ it will not Move to next (b).

(ii) index of ~~b~~ character from last



hyder Abbas
0 1 2 3 4 5 6 7 8 9 ←

```
String s = "hyder Abbas";
s.o.p(s.lastIndexOf('s'));
output:
```
↳ output: 9

```
s.o.p(s.lastIndexOf('b'));
```
↳ output: 7

```
String s = "hyder Abbas";
s.o.p(s.lastIndexOf('z')); // (-1)
```

→ z not there in string

→ -1 is default value it returns.

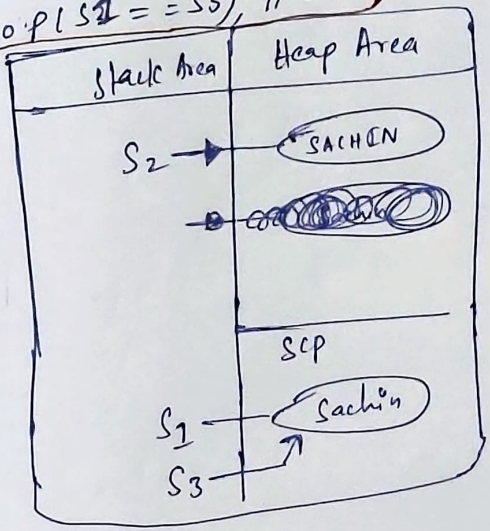
~~Scenario~~

① Predict the output

```
String s1 = "Sachin";
String s2 = s1.toUpperCase();
String s3 = s1.toLowerCase();
s.o.p(s1==s2); // false
s.o.p(s1==s3); // true
```

→ ~~no change~~

no need change it's already in lower case



## ⑨ Public String toString() Method

```
Class Student{
    String name = "sachin";
    int id=10;
}

Public Class Test{
    Public static Void main(String[] args){
        Student std = new Student();
        System.out.println(std);  // Student@Hexadecimal value
        System.out.println(std.toString());
    }
}
```

→ Same output

↳ call back Method

→ Whenever Variable is printed a Method Called toString automatically.

```
String name = new String("dhoni");
$.o.p(name);  // dhoni
$.o.p(name.toString());  // dhoni
```

Same output

→ Whenever Variable is printed a Method Called automatically.

Eg

```
Student std = new Student();
System.out.println(std);
System.out.println(std.toString());
```

→ gives same output.

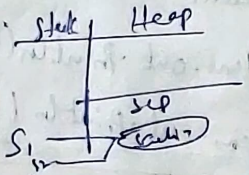
Method called automatically when we Print Variable.

→ Return type of toString() is String.

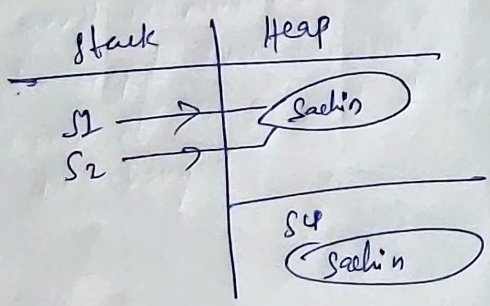
Note:- Whenever we print any reference, by default JVM will Call to toString() on the reference.

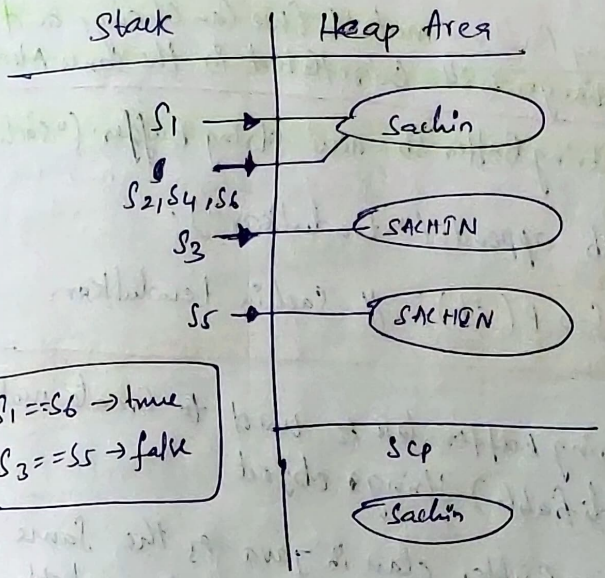
## Snippets

① String s1 = "Sachin";
String s2 = s1.toString(); → no change
System.out.print(s1 == s2); // true
output : true




② String s1 = new String("Sachin");
String s2 = s1.toString();
String s3 = s1.toUpperCase();
String s4 = s1.toLowerCase();
String s5 = s1.toUpperCase();
String s6 = s1.toLowerCase();
System.out.print(s1 == s6);
System.out.print(s3 == s5);




→ String builder (or buffer are clones of mutable strings.

① StringBuffer :- mutable (we can change and changes will be reflected in the same Memory)

Eg① StringBuffer sb = new StringBuffer ("sachin");
sb.append ("tendulkar");
S.o.P(sb); // Sachin tendulkar.

→ StringBuffer class is used to create mutable (modifiable) String object.
→ StringBuffer class in Java is the same as String class except it is mutable.
→ it can be changed.

Eg② → Compile time Constant

```
final int a = 10;
a += 10;
S.o.P(a);
```

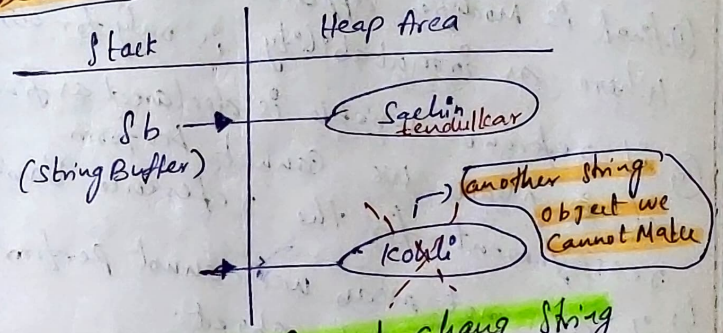
→ 'a' cannot be mutable.
→ final :- Compile time Constant
↳ once it is declared it cannot be changed.

Eg③ :-

```
final StringBuffer sb = new StringBuffer("sachin");
sb.append("tendulkar");
System.out.println(sb);
sb = new StringBuffer("kohli");
```

↳ sb will not point kohli because of final keyword

sb is final so can't be reused




→ final keyword cannot change String buffer nature
→ content of StringBuffer can be change because it is mutable.
→ sb cannot point to another object when we use final keyword for StringBuffer.
→ sb is final so can it be reused.

## ② final Keyword :-

it is a non-access modified used for Classes, attributes and ~~for classes and~~ Methods which makes them non-changable (impossible to Inherit (or) Override.

## ③ final Vs immutability

① final is modifier applicable for Variables, where as immutability is only for objects

② if reference Variable is declared as final, it means we cannot perform reassignment for the reference Variable
↳ it does not mean we cannot perform any change in the object.

③ By declaring a reference variable as final, we wont get immutability nature.

④ final and immutability is different Concept.

Note:
final Variable (valid) ✓
final object (invalid) → we don't have this Concept
immutable Variable (invalid) "
immutable object (valid) ✓

→ StringBuilder, StringBuffer and all wrapper classes are by default mutable classes

→ wrapper classes (Byte, Short, Integer, Float, Double Float, Boolean, Character) are default Mutable.

Mutable → Can be changed
immutable → Can't be changed

(4) Important Methods of StringBuffer/StringBuilder

Same Method applicable for both

1. Public int length()
2. Public int Capacity()
3. Public char charAt(int index)
4. Public void setCharAt(int index, char ch)
5. Public StringBuffer append(String s)
6. Public StringBuffer append(int i)
7. Public StringBuffer append(long l)
8. Public StringBuffer append(boolean b)
9. Public StringBuffer append(double d)
10. Public StringBuffer append(float f)
11. Public StringBuffer append(int index, Object o)
12. Public StringBuffer delete(int begin, int End)
13. Public StringBuffer deleteChar(int index)
14. Public String Buffer reverse()
15. Public void setLength(int length)
16. Public void trimToSize()
17. Public void ensureCapacity(int Capality)
18. Public StringBuffer insert(int index, String s)
19. Public StringBuffer insert(int index, int i)
20. Public StringBuffer insert(int index, long l)
21. Public StringBuffer insert(int index, double d)
22. Public StringBuffer insert(int index, boolean b)
23. Public StringBuffer insert(int index, float f)
24. Public StringBuffer insert(int index, Object o)

java.lang.StringBuffer → Command. ✓

(5) Constructors:-

1. Public java.lang.StringBuffer();
2. Public java.lang.StringBuffer(int);
3. Public java.lang.StringBuffer(Java.lang.String);
4. Public java.lang.StringBuffer(java.lang.charSequence);

## ① Public Java.lang.StringBuffer( ); Method

```
StringBuffer sb1 = new StringBuffer();
System.out.println(sb1.length()); // o/p=0
S.o.p(sb1.Capality());
```

↳ o/p = 16

↳ length is Method Even it is part String Buffer

① length() Method → no. of characters Stored

② Capality() → how many no of characters can be stored?

→ When extra character added Capality of StringBulder increases.

→ this way JVM got programmed for string buffer.

```
StringBuffer sb1 = new StringBuffer();
System.out.println(sb1.length()); //op:0
System.out.println(sb1.Capality()); //op:16
```

// Appending 16 characters

```
sb1.append("abcefghijklmnopq");
S.o.p(sb1.length()); //o/p = 16
S.o.p(sb1.capality());
```

↳ o/p = 16

// Append 1 character to 16 characters.

```
sb1.append("a");
S.o.p(sb1.length()); → o/p = 17   (16+1) char
S.o.p(sb1.Capality()); → o/p = 34
```

↳ capality formula:

Capacity increases when characters added More than given Capality

new capacity = (old cap+1) x2

New Capality = (16+1) x2
= 34

## ② Public java.lang.StringBuffer(int); Method

Eg①

```
StringBuffer sb2 = new StringBuffer(100);
S.o.p(sb2.length());  → output: 0
S.o.p(sb2.Capality()); → output: 100
```

→ While Creating String Buffer object if number is given as argument that number will be treated as Capality.

length = 0 (no characters)

Eg②

```
StringBuffer sb2 = new StringBuffer(19);
S.o.p(sb2.length()); o/p: 0
S.o.p(sb2.Capality()); o/p: 19
sb2.append("sachin")
S.o.p(sb2.length()) // o/p: 6 (no. of characters)
S.o.p(sb2.Capality()); o/p → 19
```

## ③ Public java.lang.StringBuffer(java.lang.String)

Eg①

```
StringBuffer sb4 = new StringBuffer("sachin");
S.o.p(sb4.length()); o/p: 6
S.o.p(sb4.Capacity()); o/p: 22
```

o/p: 22 = length of string + default capacity

↓

Capacity = length of String (6) + default capacity (16) = 22

→ if string is given as argument then Capality will be

Capacity = length of string (6) + default capacity (16)

# Methods of String Buffer / String Builder

13/12/

① Public int length() :- • gives no. of characters stored in String Buffer.

Eg ①:-
```
StringBuffer sb = new StringBuffer("Sachin");
S.o.p(sb);   output :- 6
```

② Public int Capality() :- it shows How many Characters are stored.

Eg ①:-
```
StringBuffer sb = new StringBuffer();
S.o.p(sb.Capality());
   ↳ output : 16 (default capacity)
```

⇒ default Capality of String Buffer is 16.
↳ 16 characters

Eg ②
```
StringBuffer sb = new StringBuffer(100);
S.o.p(sb.Capality());
   ↳ output : 100
```

→ While Creating String Buffer object if number is given as argument that number will be treated as Capality.

Eg ③ StringBuffer sb = new StringBuilder("Sachin");

String S.o.p(sb.length()); → o/p : 6

S.o.p(sb.Capality()); → o/p : 22

↳ Capacity = length of string + default Capacity
C = 6 + 16 = 22
C = 22

③ Public Char charAt(int index) :-

Char : Sachin
index : 0 1 2 3 4 5

Eg ①
```
StringBuffer sb = new StringBuffer("Sachin");
S.o.p(sb.charAt(4)); → o/p :- i
S.o.p(sb.charAt(-1)); → o/p
   ↳ o/p :- StringIndexOutOfBoundsException
```

→ Printing character At index 4 is i.

④ ~~Public char setCharAt(int index, char ch);~~

Char : S a c h i n
0/1 2 3 4 5 6 7 8 9 10 11 12 13 14

StringBuffer sb = new StringBuffer("SachinTendulkar");

④ Public char setCharAt(int index, char ch)

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| char: | S | a | c | h | i | n | t | e | n | d | u | l | k | a | r |
| index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

→ t is replaced by 'T'

```
StringBuffer sb = new StringBuffer("SachintendulkarJ");
sb.setCharAt(16,'T');
S.o.p(sb)
```

↳ o/p :- SachinTendulkar