Date: / / Page no:

• String and StringBuilder :-

Strings, which are widely used in java programming are a seqvence of character.

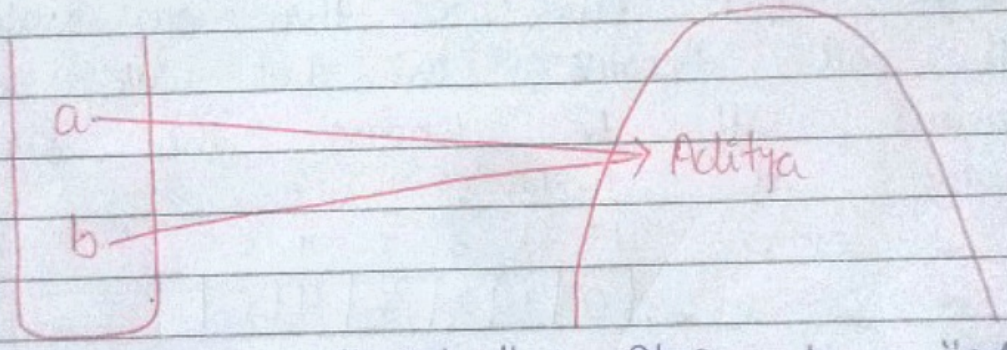
String is a type of class. Everything that starts from capital letter is class.

* String is class *

Example :-

- Choose the correct :-

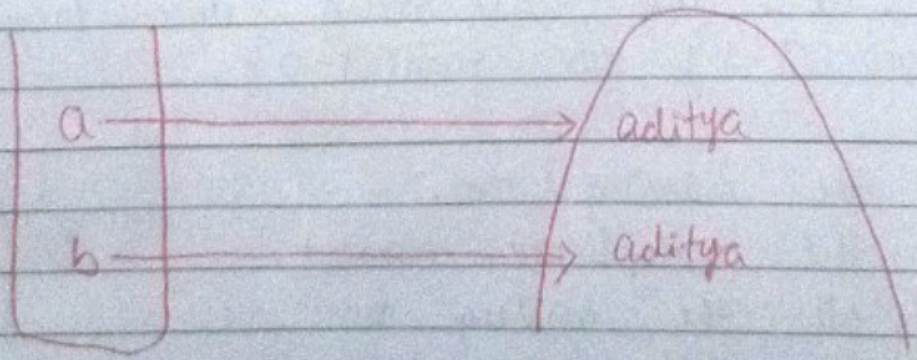
(a)



String a = "aditya" ; String b = "aditya" ;

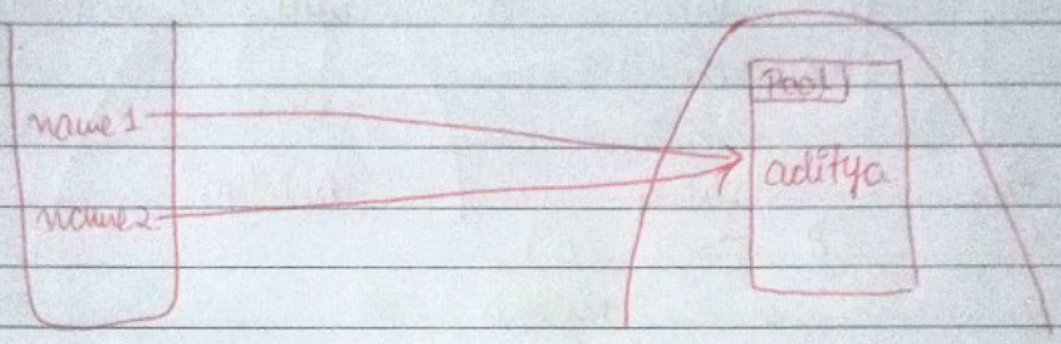
OR

(b)



Actually both are incorrect for the given Strings. The correct will be :-

```
String name1 = "aditya";
String name2 = "aditya";
```



pool is nothing but String pool.

- **String pool :-**

String pools are nothing but a storage area in java heap where String literals stores. It is also known as String internal pool.

"Separate memory structure in heap."

all the similar values of Strings are not recreated String.

Date: / / Page no:

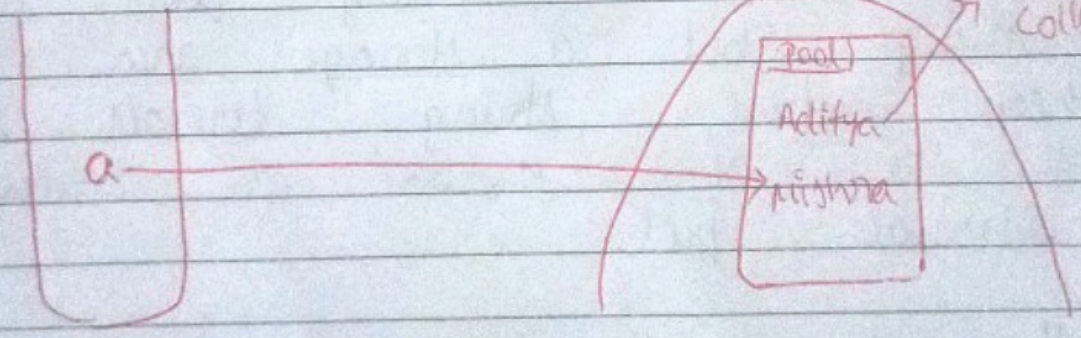
- **Immutable :-**

Strings are immutable
Immutable means unchangeble.

Ex:- Suppose a and b both are having the same value and if we change the value of b then a will not be changed.

⇒
```
String a = "Aditya";
// Sout (a);
a = "Mishra";
Sout (a);
```

output :- Mishra. → how?



* Now the value of a has not been changed but instead of that it has been assigned to new value.

- Reason why Strings are immutable :-

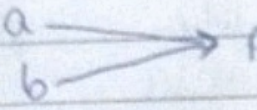
→ For security purpose Strings are immutable.

Suppose we have multiple user with same name and any one of the user wants to change the name. If the Strings are immutable then everyone's name will be changed in the database.

- **Comparison of String :-**

i) == operator :-

a → Kunal
b → Kunal
== will give false.



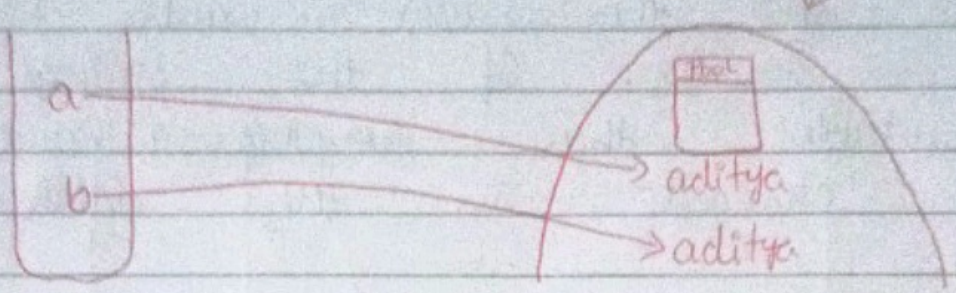
== will give true.

- How to create differ objects of same value :-

```
String a = new String ("aditya");
String b = new String ("aditya");
```

Here we are telling java that we want to create a new object by using new keyword.

// Creating these values outside the pool but in heap.



Now - if we do a == b
it will give false.
Because they are not pointing to the same object.

ii) .equals method :-

When you only needs to check value, we use .equals method.

Sout (a.equals (b));
⇒ true.

* In String if we want to print the value at perticular index so -

.charAt (index no);

⇒ Sout (names1.charAt(0));

- **Pretty printing :-**

Prettyprint is the process of converting and presenting source code in a legible and attractive way.

Ex:-
```
public static void main (String[] args)
{
    float a = 24.5678f;
    Sout (a); // Normal printing.
    Sout ("%.2f", a);
```

it will print the no upto 2 decimal points.

we use "System.out.printf".

⇒ System.out.printf ("Hello %S How are %S", "Aditya", "You?");

output ⇒ Hello Aditya how are You?

It is widely used in competitive programming.

- **List of placeholders :-**

i) %.c → Characters
ii) %.d → Decimal no
iii) %.e → Exponential floating-pointing no
iv) %.f → Floating point no
v) %.i → Integer (base 10)
vi) %.o → octal no (base 8)
vii) %.s → Strings
viii) %.u → unsigned decimal no (integer)
ix) %.x → Hexadecimal no (base 16)
x) %.t → Date / time
xi) %.n → New line

- **String Concatenation :-**

In java, concatenation forms a new String that is combination of multiple Strings.

Ex:-

```
public static void main (String[] args)
{
    sout ('a' + 'b') // will print the
                     Sum of ASCII value
                     of both.
```

Sout ("a" + "b") // output → ab (String)
Sout (char) ('a' + 3); // ASCII value of d because 'a+3 = d'

In java operator overloading is not supported.

String aus = new Integer (56) + " " + new ArrayList <> ();

Sout (aus);

// we are using empty (" ") String in the middle because one of the object has to be the string because only String is supported in java for operator overloading.

- **String performance :-**

It means we can replicate pointers without replicating objects.

Ex :-

```
public static void main (String [] args)
{
    String series = "";
    for (int i=0; i<=26; i++)
    {
        char ch = (char) ('a'+i);
```

Date: / / Page no:

```
        Sout (ch);
        series += ch;
    }
    Sout (series);
}
```

output :- abcdefghijklmnopqrstuvwxyz

* But this is not good practice because it is copying the previous output and adding the new value to it. It takes so much memory.

For n digits we are taking $O(n^2)$ space complexity.

$$a + b + c + \cdots + n$$

$$= \frac{n(n+1)}{2} \Rightarrow \frac{n^2 + n}{2}$$

$$= O(n^2)$$

which is bad.

To overcome this we use String builders

=> => =>

• String builder :-

StringBuilder class is wed to create mutable Strings.

- Important Constructors of StringBuilder :-

i) String builder () :- It creates an empty String builder with initial capacity 16.

ii) String builder (String str) :- It creates a String builder with specified String.

iii) String builder (int length) :- Empty String builder with specified capacity.

- Methods of StringBuilders :-

i) append ()
ii) insert ()
iii) replace ()
iv) delete ()
v) reverse ()
vi) capacity ()
vii) substring ()

Ex:-
```
public Static void main (String[] args)
{
    StringBuilder name = new StringBuilder ();
    for (int i=0; i<=26; i++)
```

Date: / / Page no:

```
{
    char ch = (char) ('a' + i);
    name.append(ch);
  }
  Sout(name);
 }
}
```

Ques ⇒ W.A.P. to check palindrome String.

Ans ⇒

```
public class palindrome {
  public static void main(String[] args)
{
    String name = "Naman";
    Sout(ispalindrome(name));
  }
  static boolean ispalindrome(String name)
{
    name = name.toLowerCase();
  for(int i=0; i<=name.length()/2; i++)
  {
    char start = name.charAt(i);
    char end = name.charAt(name.length()-1-i);

    if(start != end){
        return false;
    }
  }
    return true;
  }
}
```