STRINGS & STRINGBUILDER

Strings are the type of objects that can store the characters of values.
A string acts the same as an array of characters in Java.

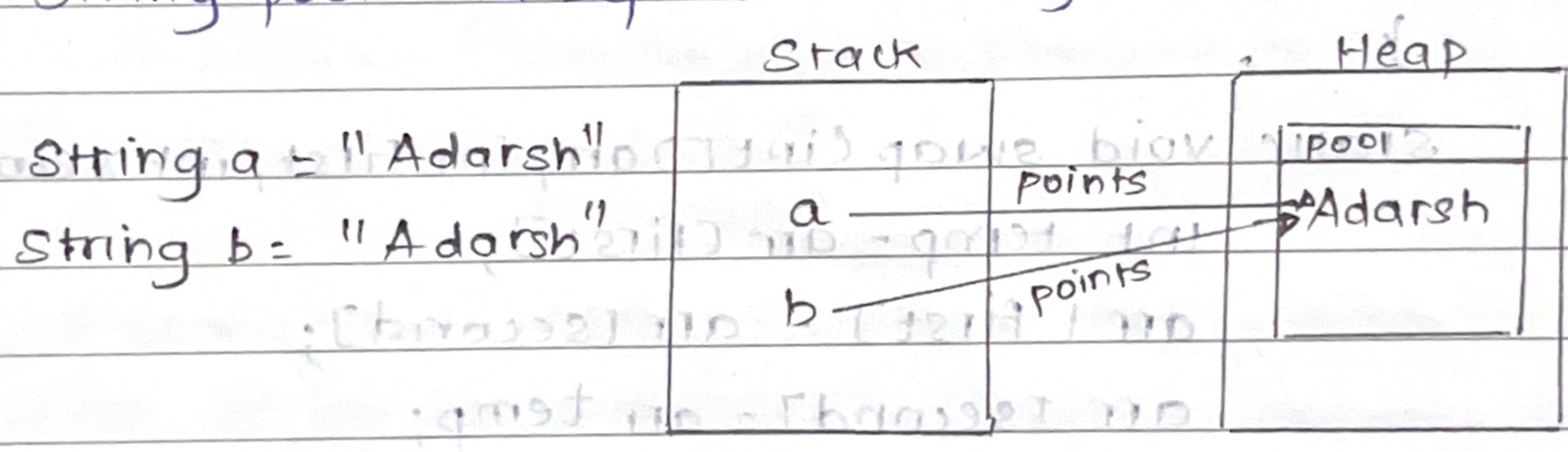
Syntax - (class of java)

class String

String name = "Adarsh";

datatype reference variable → object

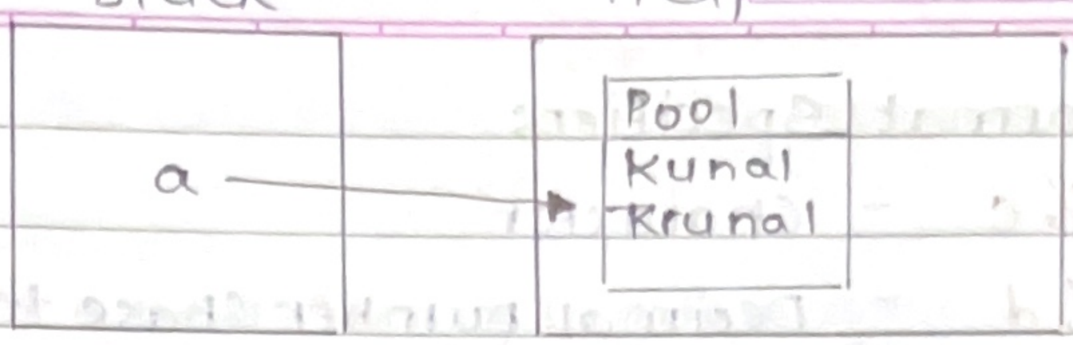
- String pool
  - String pool is a seprate memory structure inside the heap.



  - If already exist in the pool then it will not create it again and point it to existing one.
  - It makes program optimised

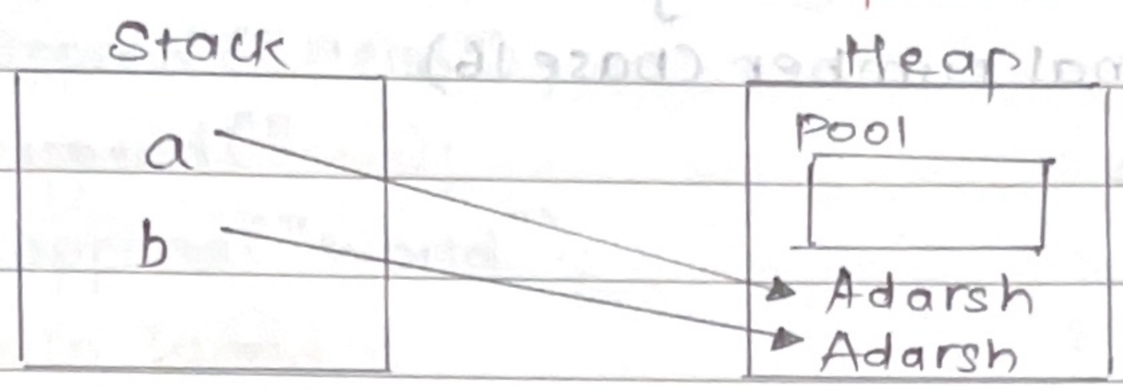
If you try to change the object via the reference variable it will not change because of Immutability.

String a = "Kunal"
a = "Trunal"



- Create different objects of same values

String a = new String ("Adarsh")
String b = new String ("Adarsh")
(It creates value outside of pool but inside the heap)



- a == b False as they are not pointing to same location.
- a.equals(b) True as the value of a and b is same.

- PreetyPrinting.
Printing output in specified format.

```
Float a = 453.1274F;
System.out.printf("Formatted number is %.2f", a)
```

%.2f → placeholder
The number of decimal values

- printf — Allows you to create formatted output by replacing placeholders with actual values from variables

Output

Formatted number is 453.13 → rounds off the value.

- Format Specifiers

%.c - Character
%.d - Decimal number (base 10)
%.e - Exponential floating-point number
%.f - Integer (base 10) floating-point number
%.i - Octal number (base 8) Integer number (base 10)
%.o - Octal number (base 8)
%.s - String
%.u - Unsigned decimal (integer) number
%.x - Hexadecimal number (base 16)
%.t - Date/Time
%.n - Newline.

```
System.out.println("Hello my name is %.s and I am %.s, "Adarsh",
                   "computer science enthusiast");
```

Output -
Hello my name is Adarsh and I am computer science enthusiast

- StringBuilder
- It is a class that provides a convinent way to manipulate and construct strings efficently.
- It is particulary useful when you need to perform multiple string concatenations, as it avoids the performan overheads associated with frequent string modifications
  - Using + operator for every string concatenation in Java creates new string object and can be inefficent while performing multiple concatenations or frequent modifications
  - + in java you can only use this with primitives and you can also use with complex object as well but the condition is atleast one object shouldbe of type string

- Creating StringBuilder
  ↳ creates StringBuilder with specified string.
  StringBuilder(String str)

```
StringBuilder sb = new StringBuilder();
```

- Methods to manipulate and modify strings - (With examples)
- append() - Adds the specified value at the end of the current sequence.
  -
```
StringBuilder sb = new StringBuilder();
sb.append("Hello");
sb.append(" ");
sb.append("World");
//Hello World.
```
- insert() - Inserts the specified value at the ~~end of current sequence or at the given index~~.
  -
```
sb.insert(5, "Awesome");
//HelloAwesome World
```
- delete() - Removes characters from sequence, starting from specified start index to specified end index
  -
```
sb.delete(5,12);
//Hello World
```
- reverse() Reverse the sequence of characters
- sb.reverse();
  //dlroW olleH
- toString() converts the StringBuilder object to regular String
- capacity() Returns the current capacity (size of internal character array)
- charAt(index) Returns the character at specified index
- length() Returns the number of character in sequence.