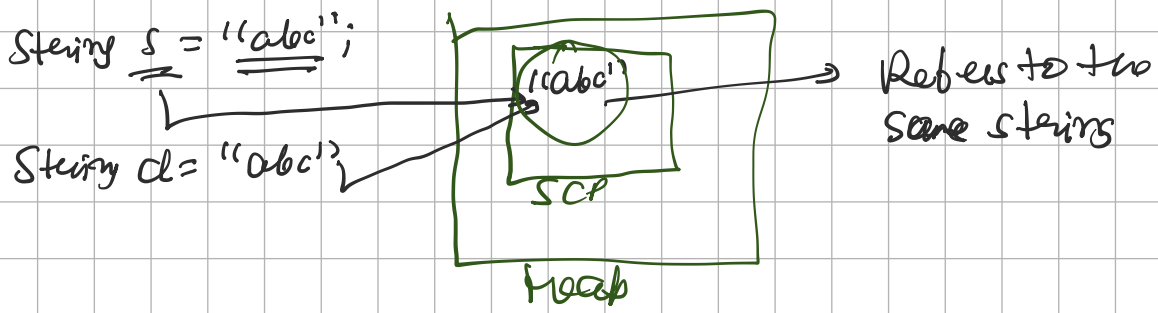
✡ String and Collections :-

⇒ Strings are immutable. (final class)
Once they are created its state cannot be modified.

String a = "abcd"
String b = a.concat("xyz"); → new String here
↳ extra memory

i) Thread safety :-
↳ No need to handle synchronizations.

(ii) SCP (String Constant Pool)



Because of immutability both refers to same reference.

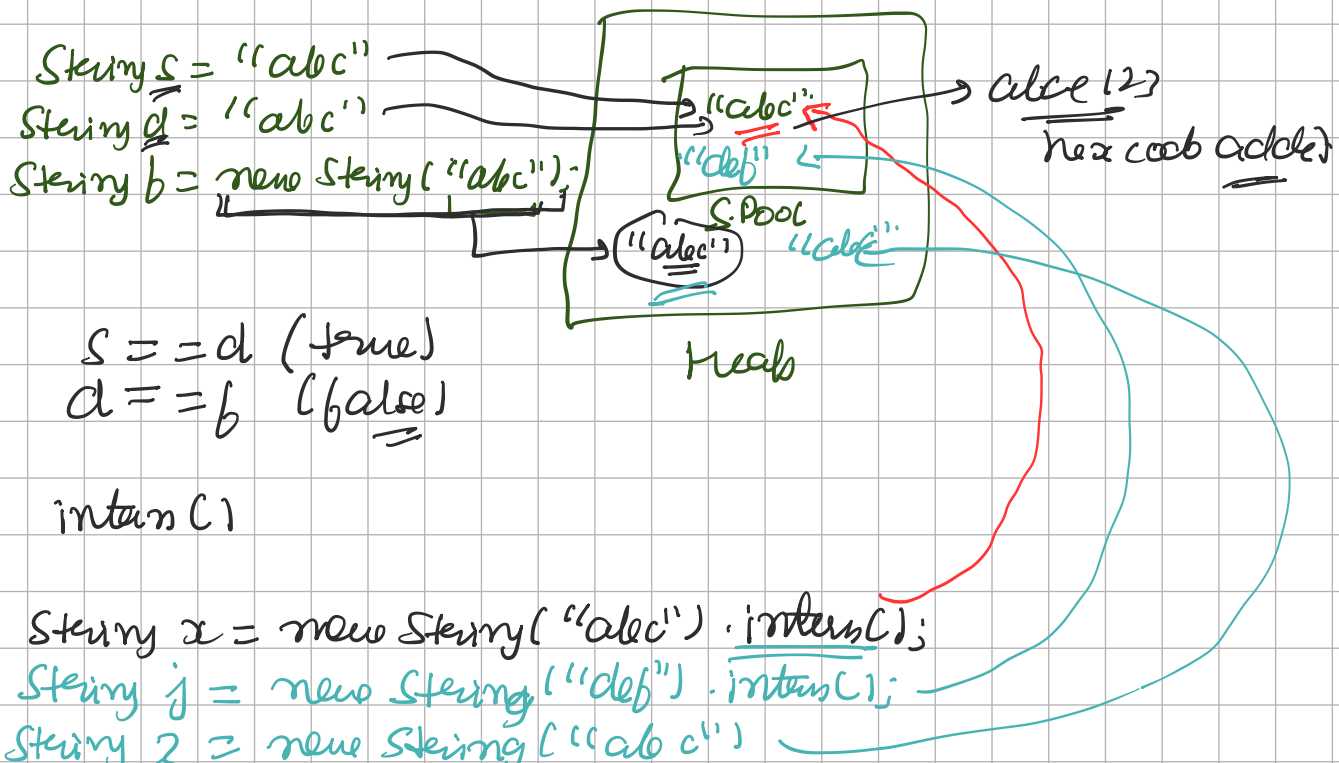
(String interning) → where multiple refs refer to same string pool. in same location.

(iii) Security
↳ ------ Password (Python) also has immutability
↳ 3rd party lib (sdk) may
↳ which could modify our string

(iv) Caching Optimization
↳ Caching it is also easy and can be reused

v) Hashcode
↳ For strings we don't need to override hashcode

"hash" → 123 "hash" → 123



s == d (true)
d == b (false)

intern()

String x = new String("abc").intern();
String j = new String("def").intern();
String z = new String("abc")

sout(z);

void sout(String s) {

Ref is passed as value

⇒

☆ String vs StringBuilder vs StringBuffer

	String	StringBuilder	StringBuffer
Mutable	X	✓	✓
Thread safety	✓	X	✓
Inplace modification	X	✓	✓

☆ Pseudo Code :-

psvm (String ... args) {

String s = "abc";

String d = "test" + "abc"; → new s → true

String e = "testabc"

is d == e ? True

"abc".equals(s);

→ s.equals("abc") → First one

Objects.equals(s, "abc");

↳ Good

→ Can give null pointer exception

String s = getValFromApi(---);

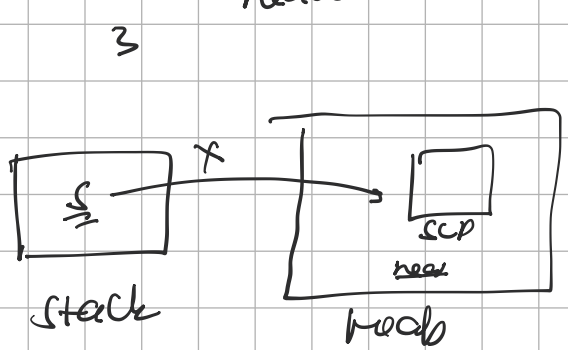
→ null.equals

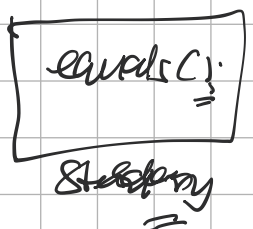
s.equals("Abc");

while(head != null) {

head = head.next;

}





s.equals

Objects.equals(-,-)

☆ String Buffer :-

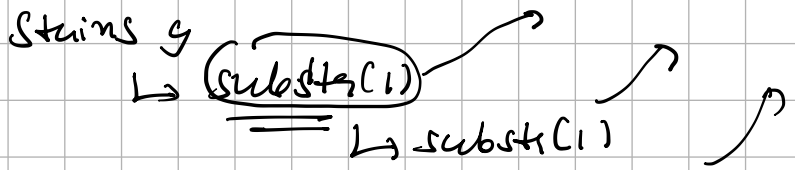
StringBuffer buffer = new StringBuffer("abc");

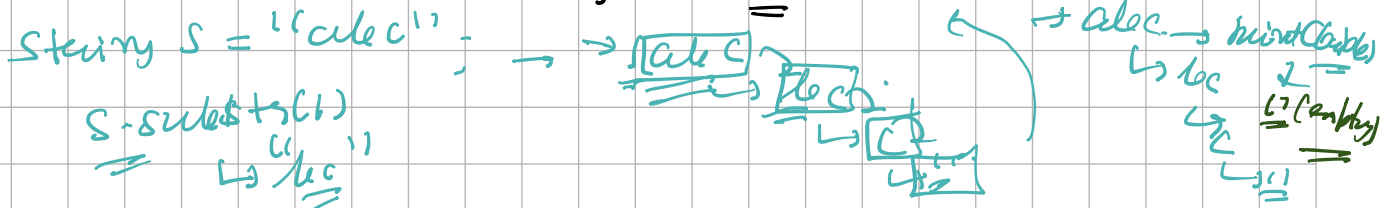
buffer.append("deb");

Thread Safe.

☆ StringBuilder :

```
StringBuilder builder = new StringBuilder();
builder.append("def");
```





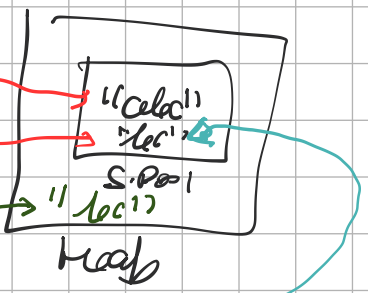
☆ String.format("Hello %s", world);

↳ String.format("Hello %d", 1);

⇒ String s = "alec"

String alec = "lec"

String x = s.substr(1);

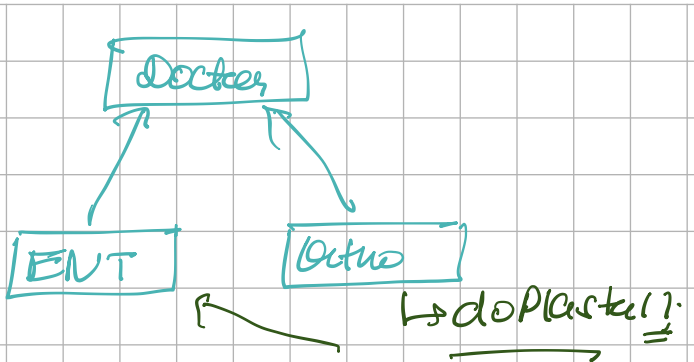


s.substr(1).intValue()

☆ Casting in java.

```
int i = 10;
double d = i; // This is allowed (implicit casting)
sout(d);

double x = 10.23;
int i = (int)x; // Explicit type casting
sout(i);
```



ENT e = new Ent();

Doctor d = new Ent();

((Ortho)d).doPlastic(); // Is this allowed

↳ Compile time it is fine.

↳ Class Cast Exception