Other Strings

Java have 2 others classes in Java, which makes String Objects mutable i.e

1) StringBuffer
2) StringBuilder

```
String s = new String("ABC")
```

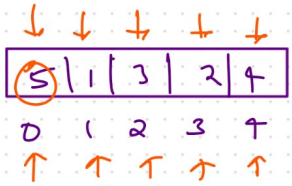
inbuilt reverse ← `StringBuffer s = new StringBuffer("ABC")`

```
StringBuilder s = new StringBuilder("ABC")
```

Note:

1) StringBuffer is "threadSafe" } StringBuilder is not a threadsafe

for Each Loop




```
for (int i=0 ; i<ar.length ; i++){
    S.o.p (ar[i]) ar[i] = ar[i]+1
}
```

→ works for only iterable objects

index value

```
for (int ele : ar){
    s.op (ele)
}
```

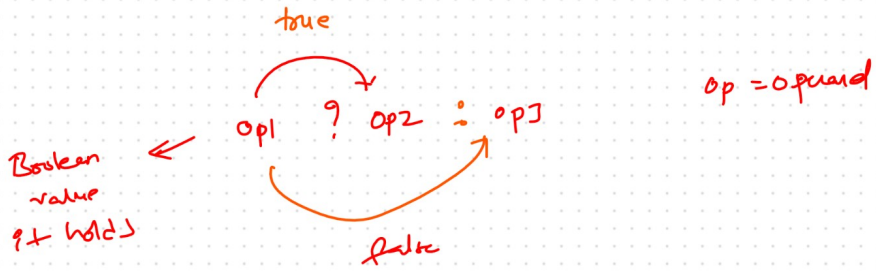
Note:

It is only useful for traversing elements

Ternary Operator

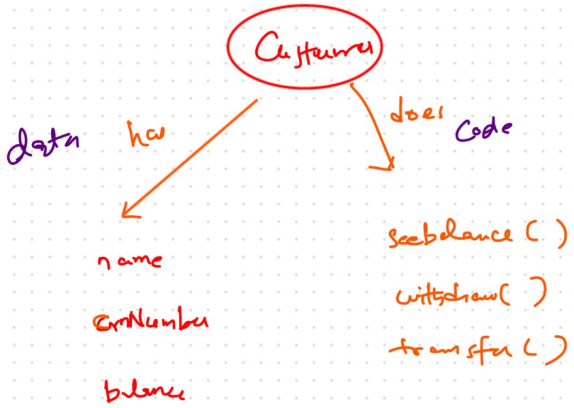
if-else Shorthand notation

1) WAP to find max of 2 numbers



1) OOP — Encapsulation

Def: The process of binding the data & code together is called encapsulation



Advantages:

We can achieve

1) code re-usability
2) abstraction
3) data security
4) maintainability
5) R/W protection (getters & setters in class)

Setters — mutators (changes)

Getters — Accessors

```
class Customer{
    String name;
    long crnNumber;
    double balance;
}
public class AlphaCustomer {

    public static void main(String[] args) {
        Customer x = new Customer();
        x.name = "Alex";
        x.crnNumber = 75124121231231;
        x.balance = -2000.00;
        System.out.println(x.name);
        System.out.println(x.crnNumber);
        System.out.println(x.balance);
        x.balance = 131231231.00;
    }
}
```

Note :

(1) In the above program, we are able to access the field directly. We can insert any values, which results in data insecurity

(2) In order to achieve data security, by the help of private keyword

[∵ making fields as private]

(3) private is a modifier in Java, which can be used for variables, method & inner classes

(4) If we make any thing as private it can be accessed only with in the class

(5) The process of making the fields as private & not giving direct access to the fields is called as "data hiding"

(6) Since direct access is not available, we have to give controlled access to the fields

(7) Controlled access can be achieved by the setter() & getter()

Notes

setter() ← local variable getter()

```
public void setName(String name){
    this.name = name;
}
```

this.name ↑ instance variable [fields]

```
public String getName() {
    return name;
}
```

(1) In the above program, the values for the fields are not assigned because the local variable & field have the same name, so priority will be given to local variable by JVM

(2) The practice of having the same name for the local variables & fields will arise a problem which is called as "shadowing"

super → String name

this → private String name Extending form of instance variable

(3) We can overcome this problem by using the keyword "this"