AGENDA

1.) Access Modifiers

2.) constructor

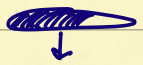
-) copy constructor

-) shallow / deep copy

start by 9:05 PM

Encapsulation :

Capsule



1.) bind Multiple drugs

2.) Protect it

is doing same thing what a capsule does.

1.) binds data together (attributes / behv.)

2.) Protecting Members from illegitimate access.

Acheive using class.

Class And Objects:

① class -

Blueprint of Entity.

syntax:

```
class student {
    // attributes
    // behaviours
}
```

② Object :

Real instance of class.

occupies space in memory.

ACCESS MODIFIERS:

help you to achieve 2nd Adv. of Encapsulation → Protecting.

1.) Private

All Members those are Private -

only self class can access.

Nobody else.

2.) Public

Anyone can access (any class)

3.) Protected :

can be accessed only within that Package. [child class can access from Anywhere]

4.) Default:

Allowed in same Package, even for child class as well.

	same class	same folder	child class same folder	c-c diff folder	Any other class
PRIVATE	✓	✗	✗	✗	✗
DEFAULT	✓	✓	✓	✗ ?	✗
PROTECTED	✓	✓	✓	✓ ?	✗
PUBLIC	✓	✓	✓	✓	✓

*) CONSTRUCTORS:

```
Student s = new Student ();
```

Student → data type

s → var

new Student () → constructor.

() → default constructor.

·) if No constructor is Provided-
default is executed.

benifits-

1.) create object

2.) initial default values.

default:

int → 0

String → NULL

boolean → false.

1.) Their can be many constructors

2.) Initially set default values
then override Given values

3.) Private constructors can exist
→ i.e. objects cannot be created

COPY CONSTRUCTOR:

Req: New object of class -
which have same values as that
of old one

student s1 = new student("yash",
21, "abc@gmail.com");

s2 → s1

way 1.)

```
s2 = new student();
s2.name = s1.name
s2.age = s1.age
.....
```

Not Good.

way 2.) COPY CONSTRUCTOR:

s1 → I have.

new student(a, b, c)

s2 = new student(s1);

```
Public student (student old) {
    name = old.name
    age = old.age
    .
}
```

↓

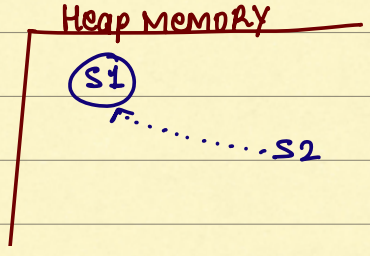
A New object with same values...

way 3.)

```
s1 = new student();
s2 = s1;
```

XXX

X X X X



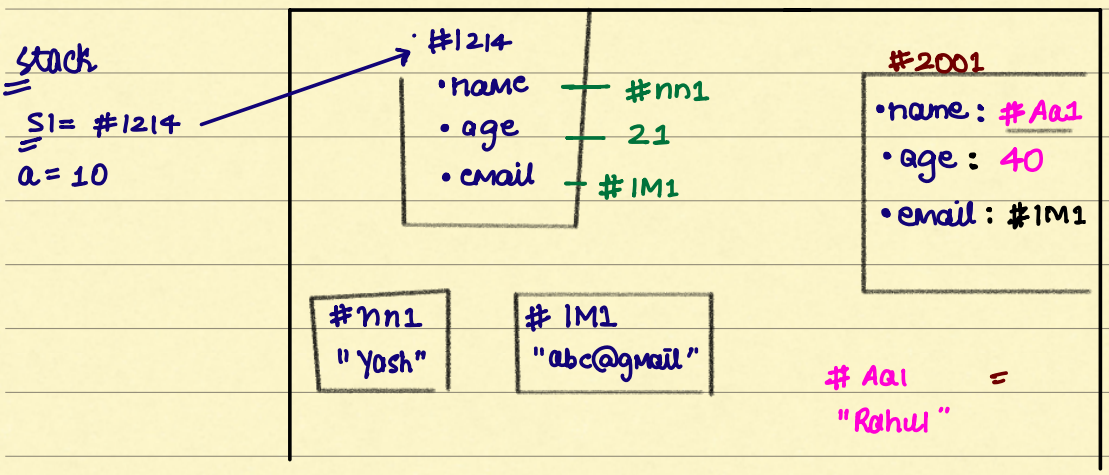
*) HOW COPIES WORK IN JAVA:

(Heap) ① NON Primitive (Object / String)
(stack) ② Primitive int / double / boolean

student s1 = new student();
int a = 10;

s1 ⟶ stored stack (as variable)

Heap (NON-PRIMITIVE)



// COPY CONS.

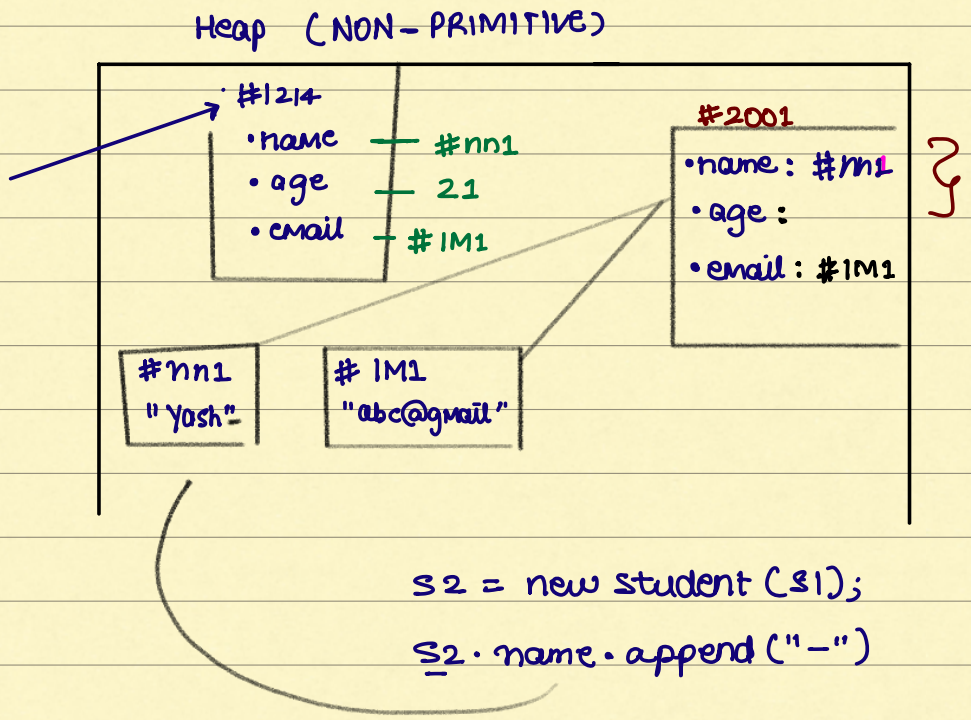
= S2 → Address is #2001

s2 = new student(s1);

s2.name = "Rahul"

s2.age = 40

✓ FINE.




```
s2 = new student (s1);
s2.name.append("-")
```

// This will change (s1) as well...
(STRING ARE Mutable - assump.)

SHALLOW COPY:

- You created New objects
- Internally they pointed to same locations for attributes.
  → They shared variables.

Deep COPY:

where old and New Object DON'T share data.

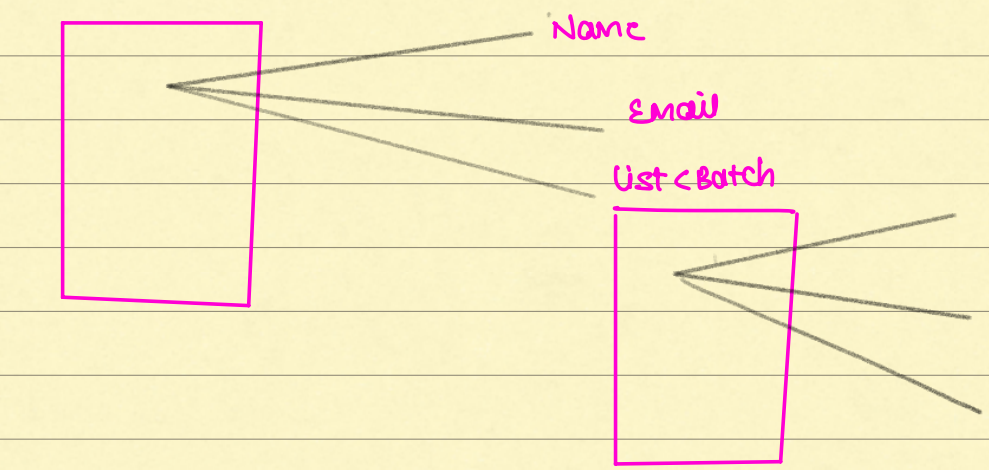
In Java → Easy to create deep copies?

Student
- name
- email
- list<Batches>

Batch
- name
- list<Instructors>
-

Instructor
- name
- list<Companies>

Heap



Very tough to create deep copies.

Q.) Is JAVA Passby value OR PassByRef ??

#1214 – As value

```
doSomething(student s1){
    s2 = new student(s1);

    s2.name="yash";
```

× Not affect s1

```
}
```

```
s2.name.append("-");
```

↳ will afect old object.

Note: Java is pass by value
the value that is passed is address.