Agenda
(0) Deep copy / PBV | PBR
(1) Inheritance
(2) Polymorphism
(3) Constructor Chaining ✓

(1) Deep Copy / PBV / PBR

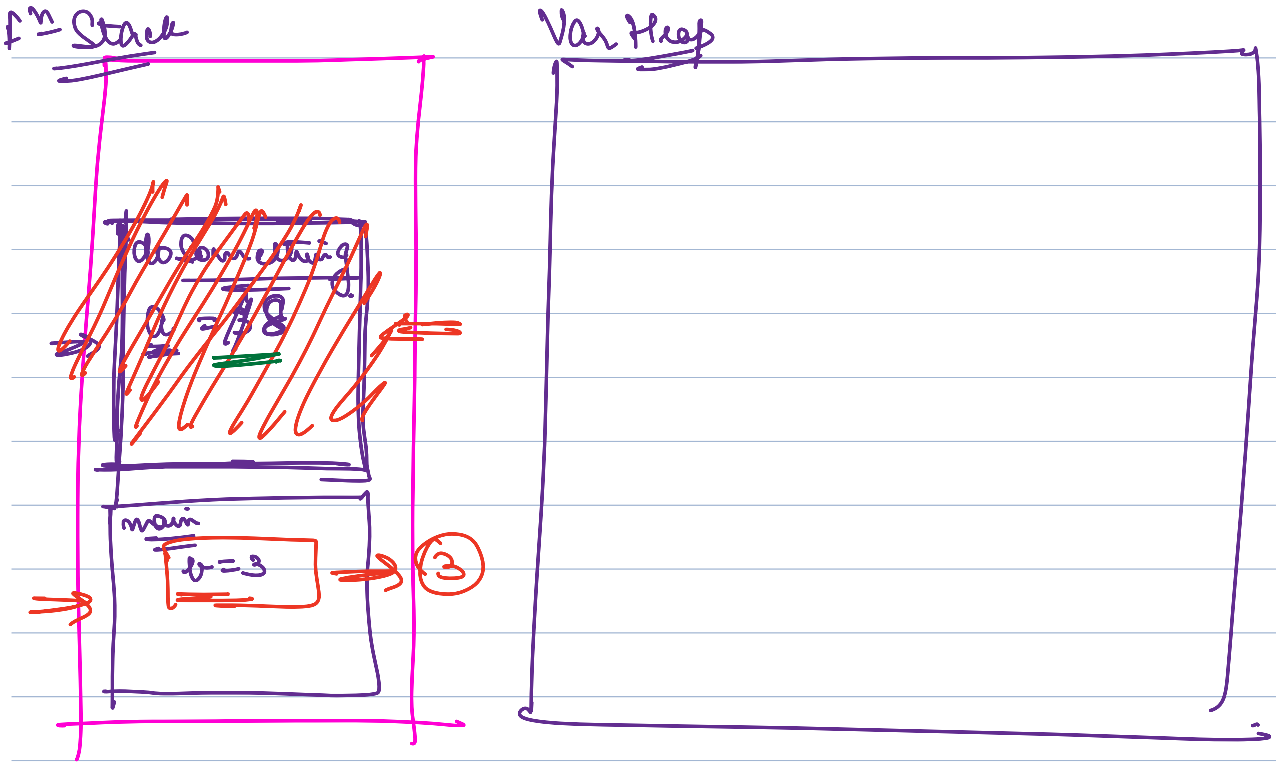
(a) → value if primitive
  → address if non-primitive

```
void doSomething(int b) {
        b = 7;          ←
}

int main() {
        int a = 2;
        doSomething(a)
        print (a);        → (2)
}
```

```
void  doSomething( list<int>  l){
        l.add(6);
}

int  main(){
    list <int>  l = {1, 2, 3, 4}
    doSomething(l)
    print(
    l.size()    => 5
    );
}
```

In C++, when we pass arguments to a f^n
they can be passed in 2 ways:

① Pass by value
② Pass by ref



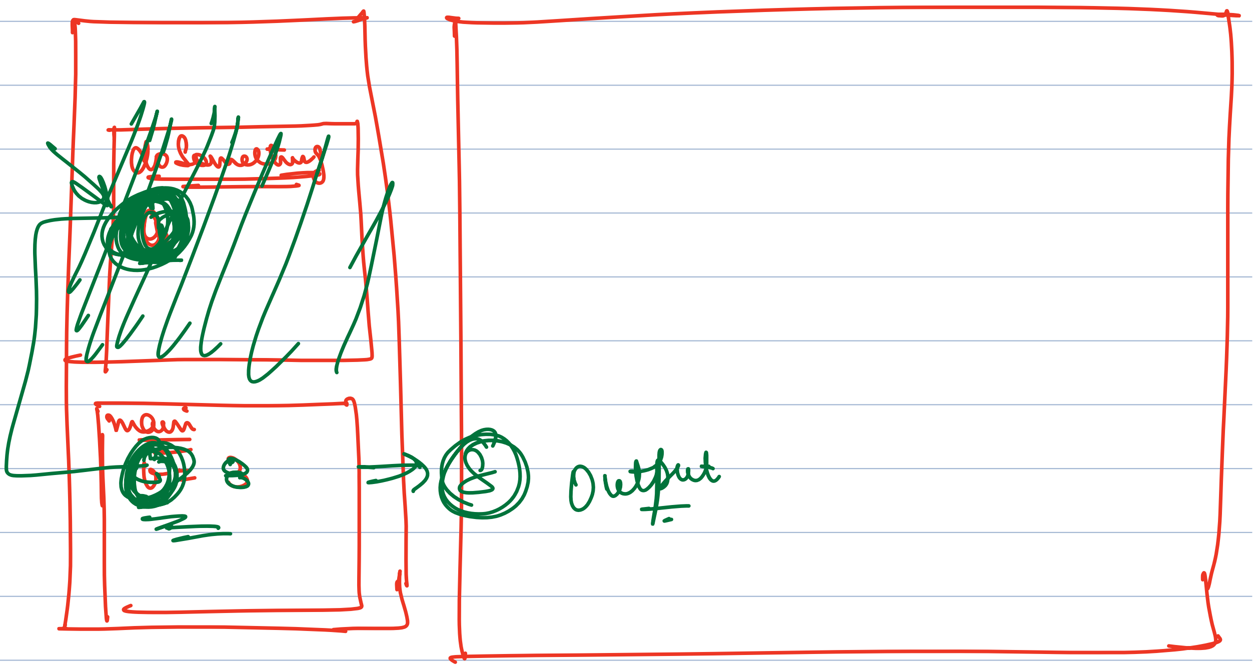
Pass By Value

```
void  doSomething ( int a ) {
          a = 8
}

int  main() {
     int  b = 3;
     doSomething (b)
     print (b)
}
```

<= Pass By Value

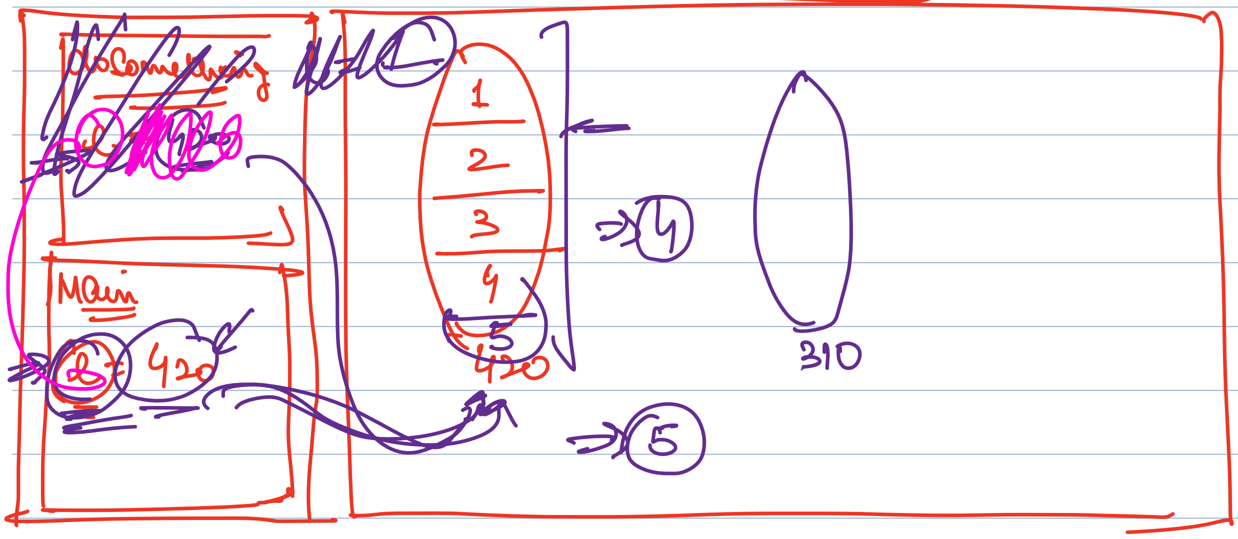
Pass By Ref

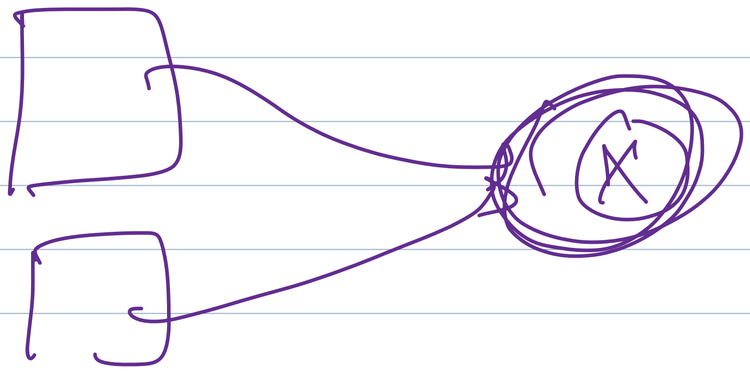
```
void  doSomething ( int &a ) {
          a = 8
}

int  main () {
     ( int  a = 3;
       doSomething (a);
       print (a)
}
```

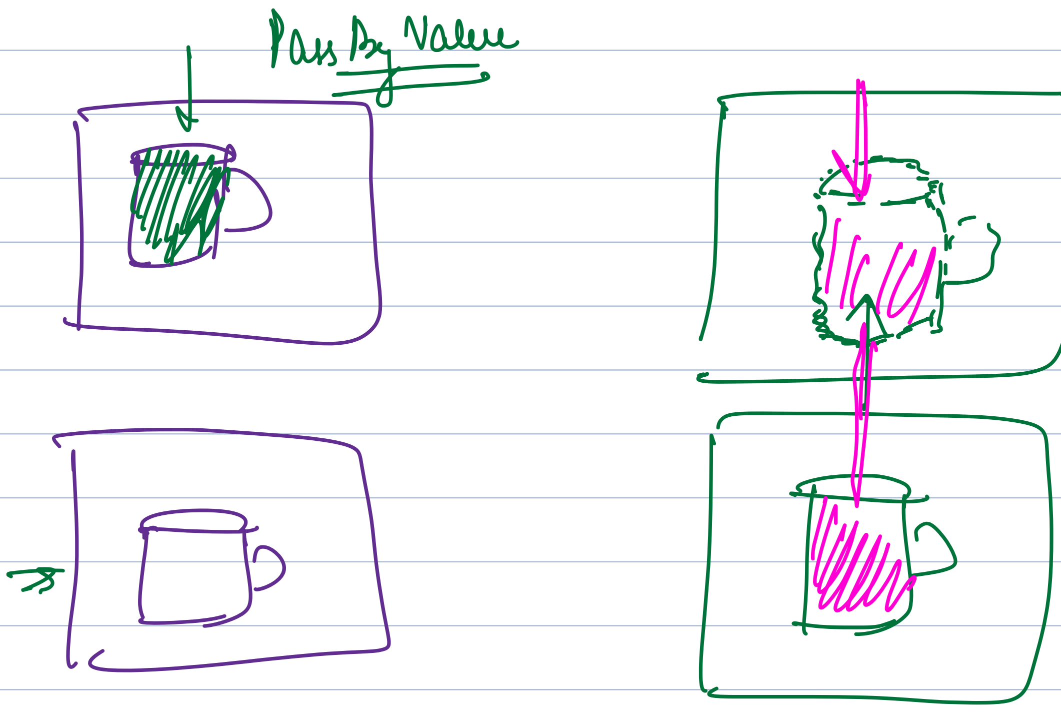


```
void doSomething(List<Int> l){
    l = new ArrayList<>();
}

int main(){
    List<Int> l = {1, 2, 3, 4}
    doSomething(l)
    print(
    l.size()    => 4
    )
}
```

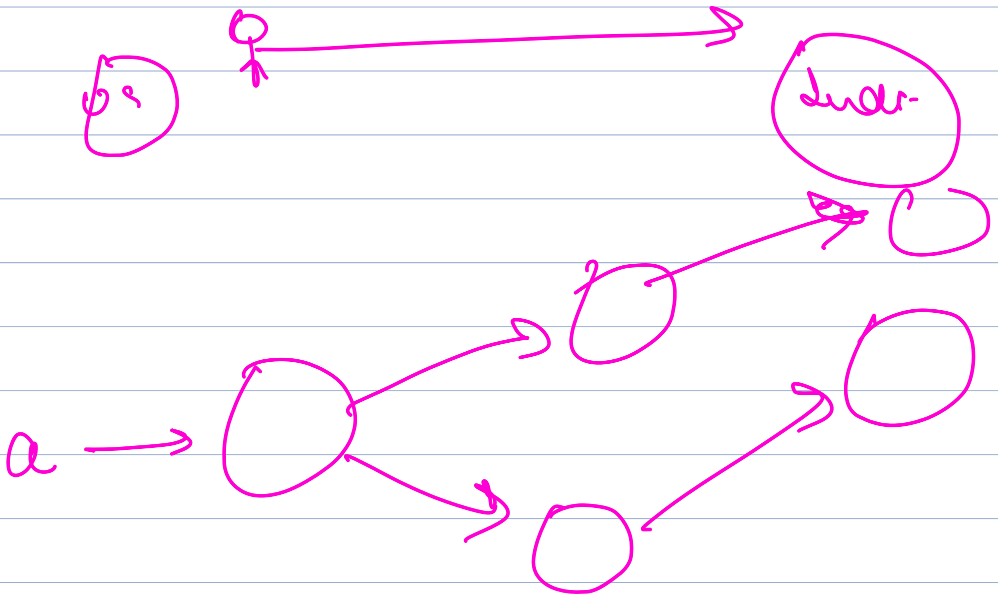
Java is always Pass by value

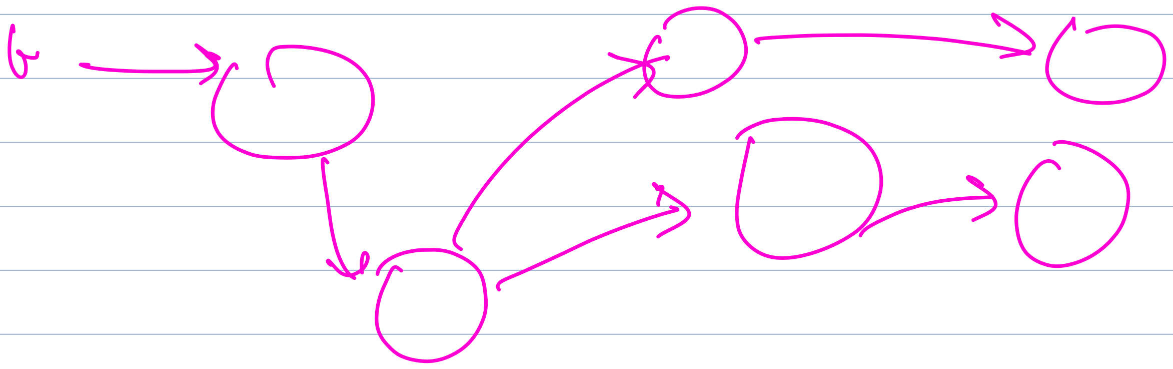





Pass By Value

US
India

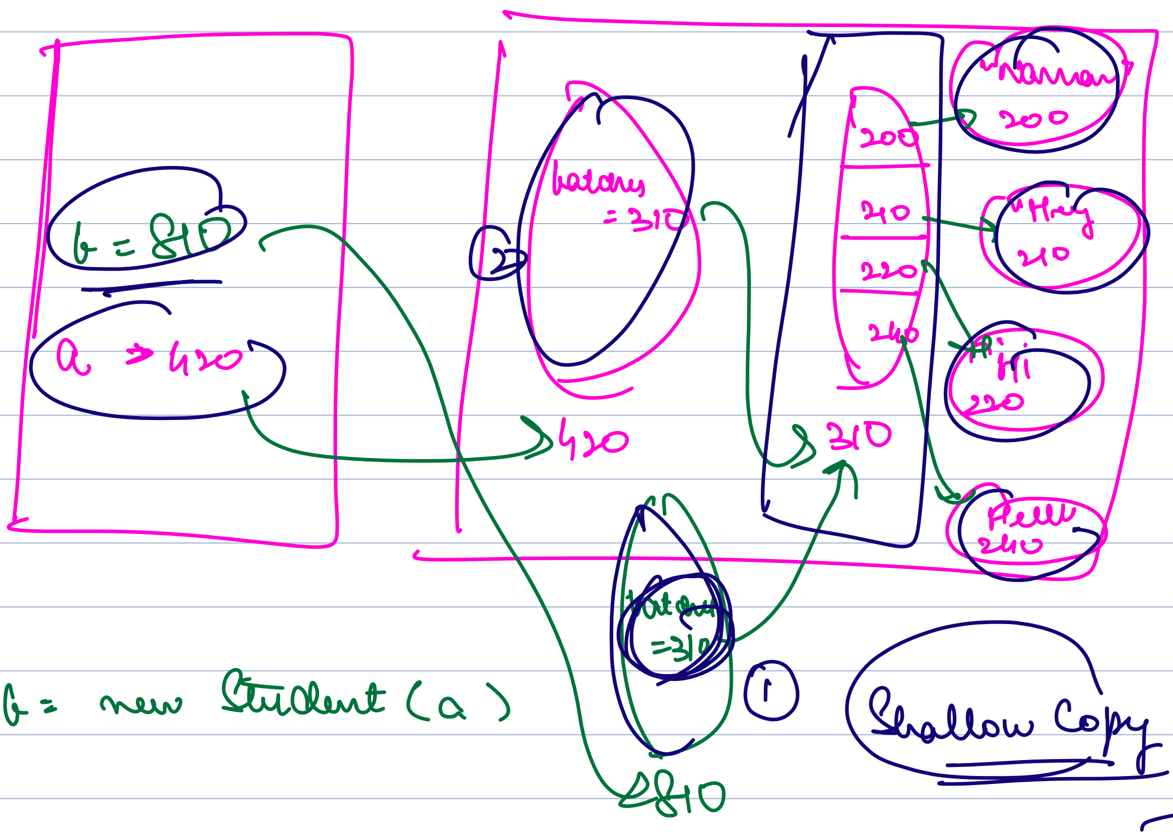

US
India
a


b

'class Student {

    list < String > batches

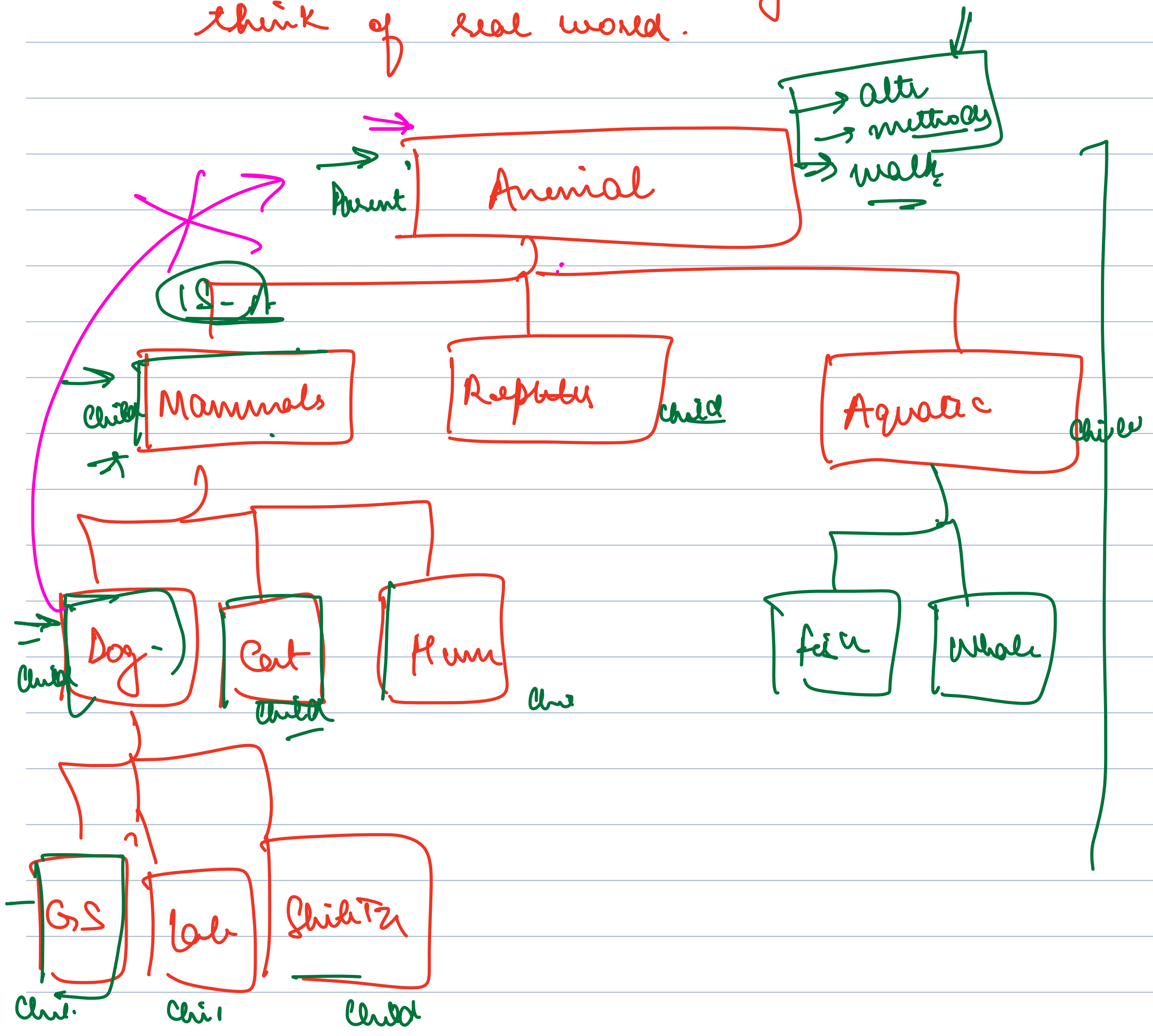
}



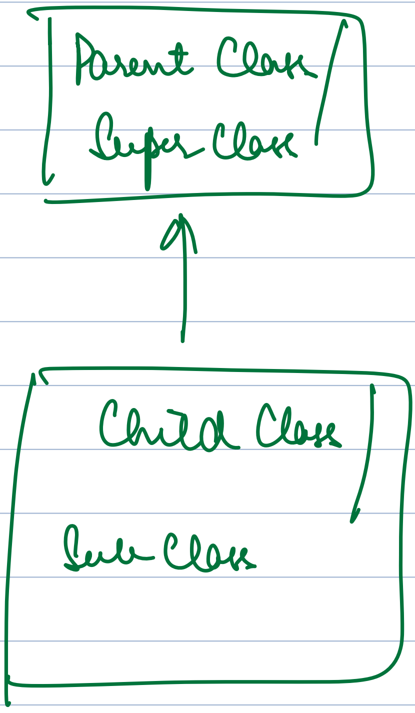
[Creating a deep copy requires to recursively copy all attributes.]

INHERITANCE

⇒ OOP is all about thinking how we think of real world.



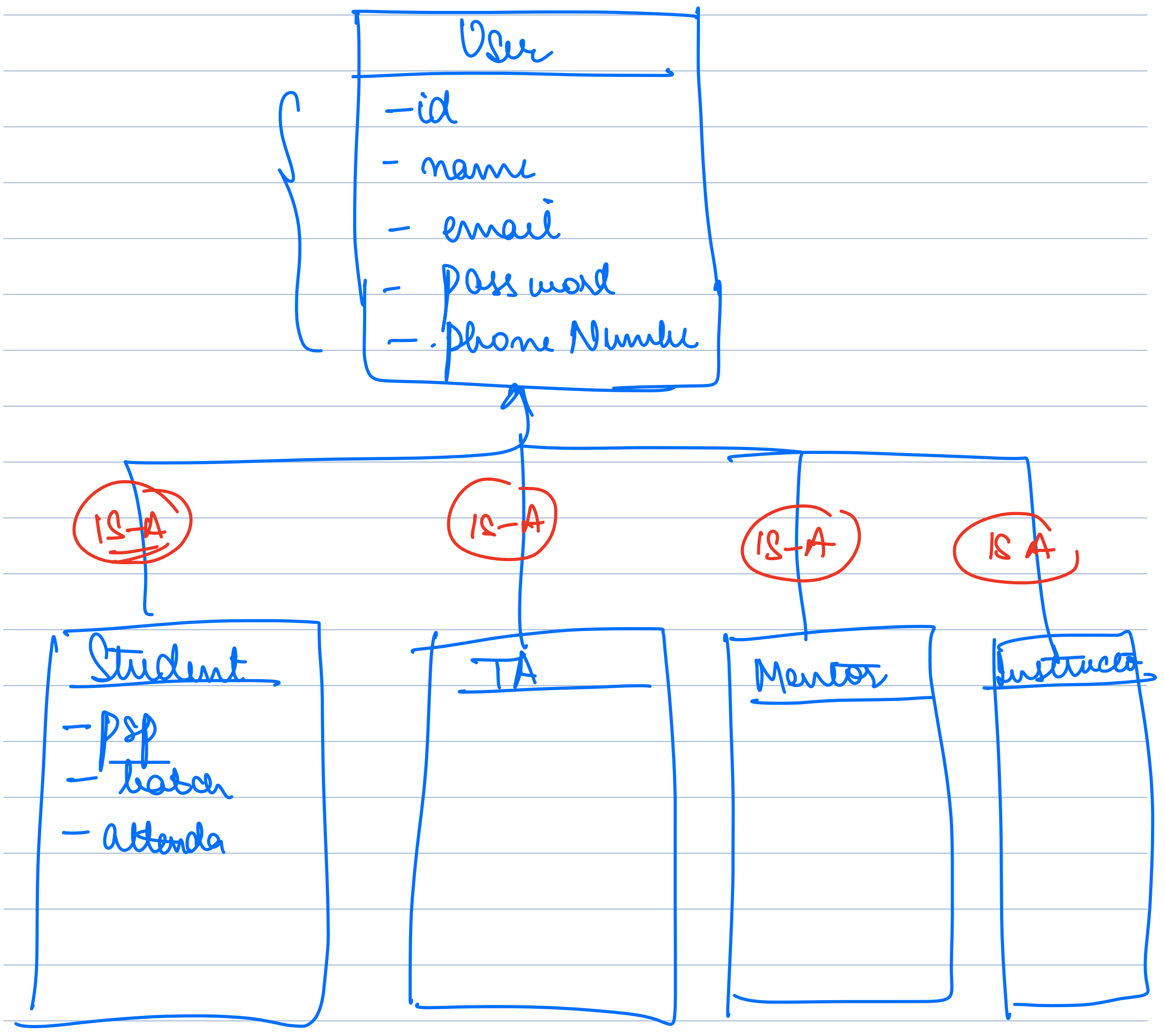
Inheritance → Rep of hierarchy in classes / entities



→ all of the attrs and methods of parent class are as-is copied to the child class

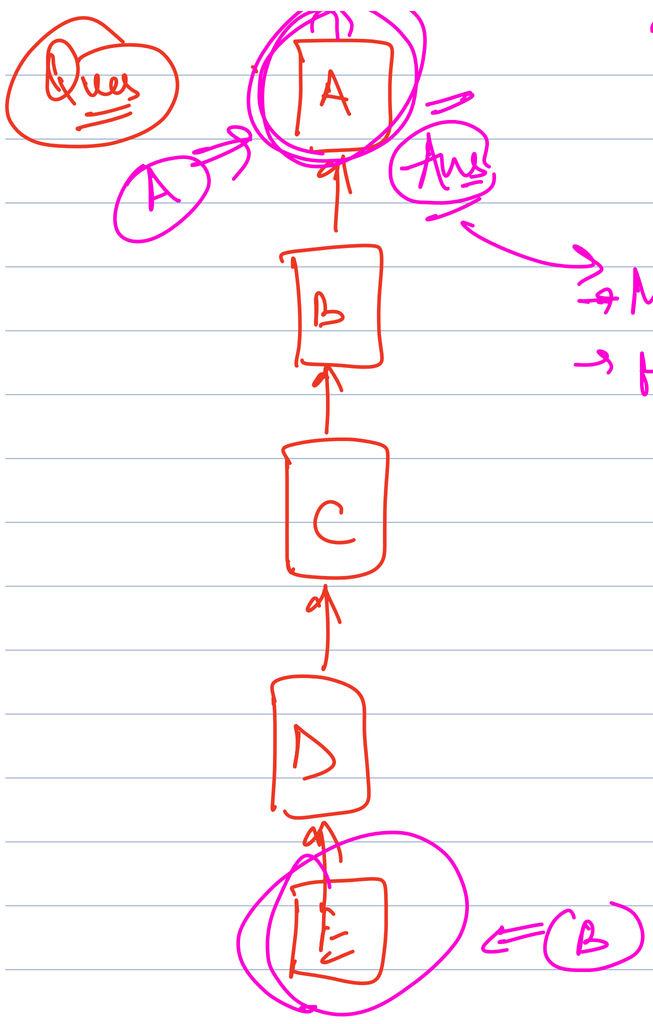
Child ⇒ Special type of parent

Parent ⇒ Generalization of diff types



⇒ all members (attr + method) of parent are present in child.

⇒ Reverse is not ~~true~~.



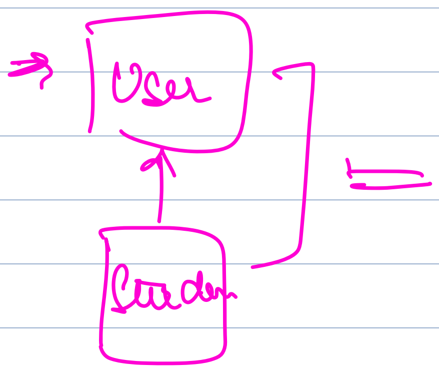
Which of these classes has highest level of abstraction ?.

↳ broadness

→ Most general

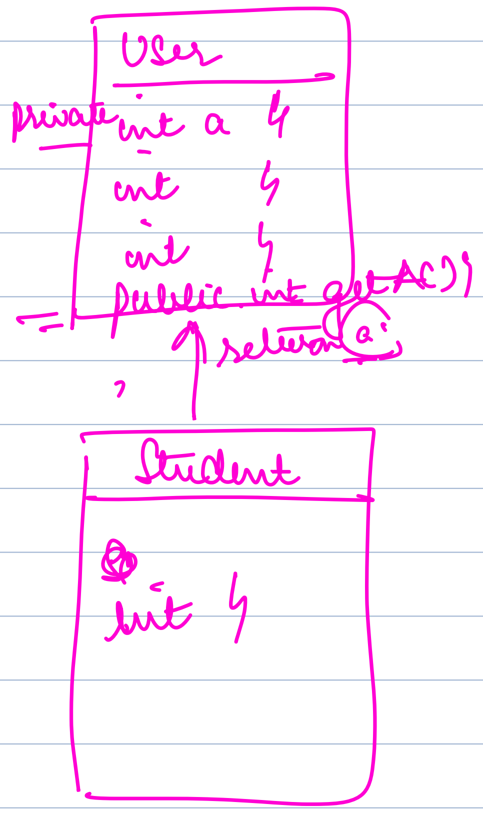
→ Highest level of abs.

⇒ How to use inheritance in practice



extends

```
class User {
        ────
        ────
        ────  } attrs
        ────
        ────

        ──
        ──
        ──    } methods.
        ──
}

class Student extends User {



}
```



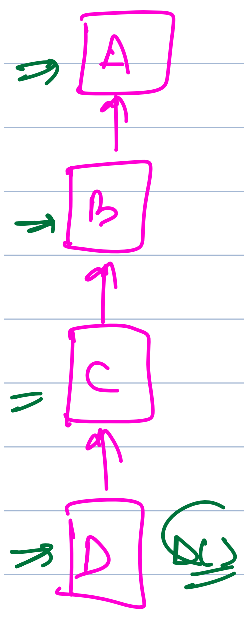
Student st = new Student()

St. getA()

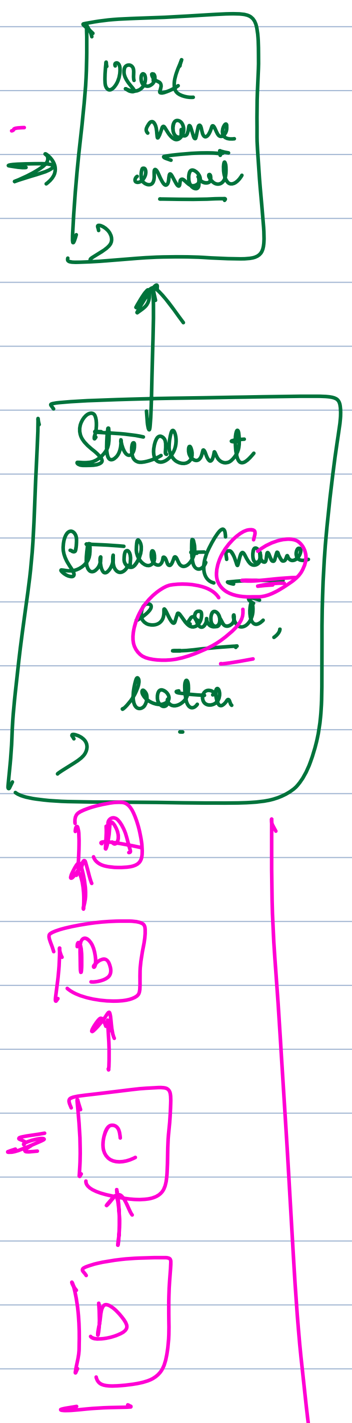
16 correct

12

HOW ARE OBJECTS OF CHILD CLASS CREATED



D d = new D()



```
Student st = new Student
        ( "Naman",
          "n@b.com",
          "ABC"
        );
```

```
Student() {                A()

   B() {

}
   }

   C() {

   D() {

   }
```

$$\begin{tikzcd} D() \arrow[r] & C() \arrow[r] \arrow[l, bend left] & B() \arrow[r] \arrow[l, bend left] & A() \arrow[l, bend left] \end{tikzcd}$$

⇒ When you create an obj of a child class, before its cons starts executing it calls default cons (cons with no param) of parent class.