Agenda

1. Access Modifiers
2. Constructors
   - Default
   - Copy
     - Deep copy V/S Shallow Copy
   - Manual / Custom
     - Parameterized
     - Pass By value V/S Pass by Ref

Encapsulation

→ bind attr/ behaviours together ⇒ class

→ protected attr/ behaviours from outside access.

```
Class Student {
    name
    email
    doIC()
    dose()
}
```

```
Client {
    psvm() {
        Student s = new Student()
        s.name = "ABCDE"
    }
}

doSomething(Student st) {
    st.name = ...
    st.age = ...
}
```

ACCESS MODIFIERS

- private ⇒ no one outside can access
- public ⇒ anyone can access.

Java

- private
- protected
- default
- public

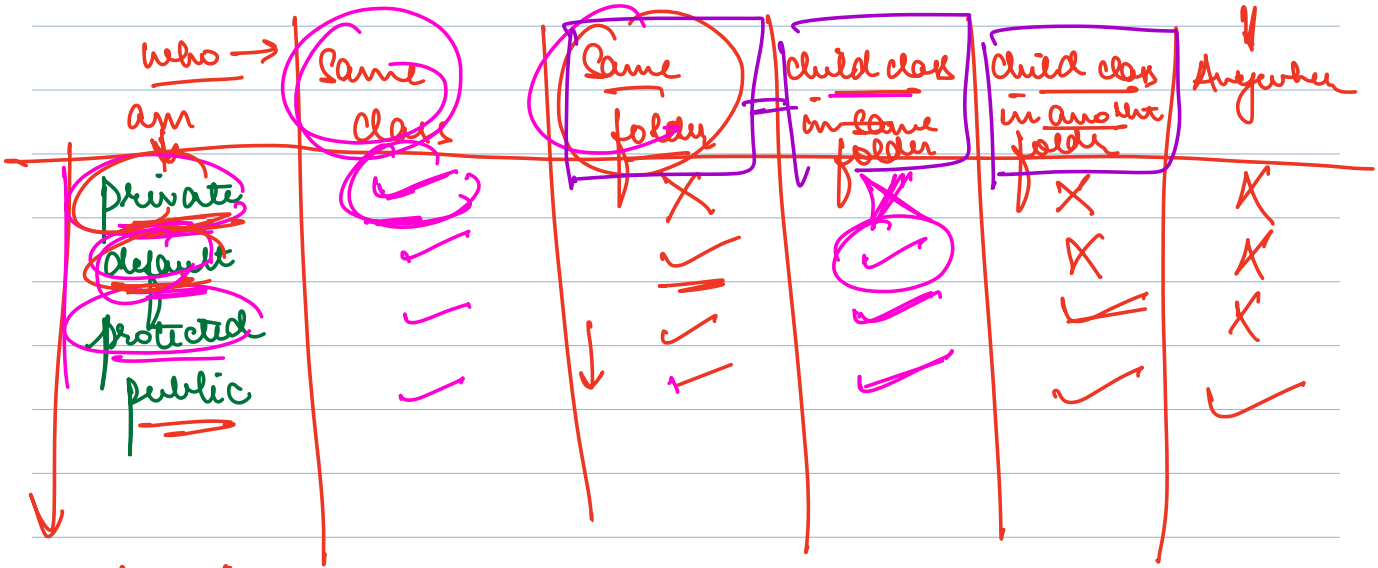
Client {

Student s;

s.name

s.age

}



most restrictive

to

least restrictive

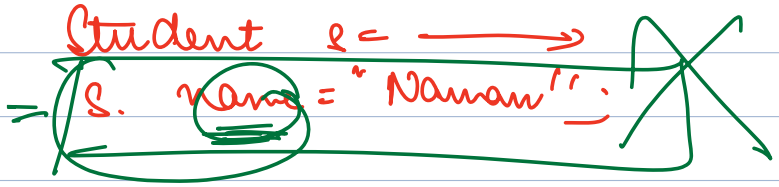
```
class Student {
    private String name;

    private void doZC() {}

    void changeBatch() {
        cout(name + " is changing batch.");
        doZC()
    }
```

class Client {

psvm() {

Student s = ⟶

s.name = "Naman";



}

⇒ Good practice to have private attr.
getters / setters

class Student {

private String name;

public void setName(String to) {

print(name is trying to set as.

name != to

if (to == Robot)

① → Logging
② → Validation
③ → Not tied to imp

}

public String getName() {

return name;

}

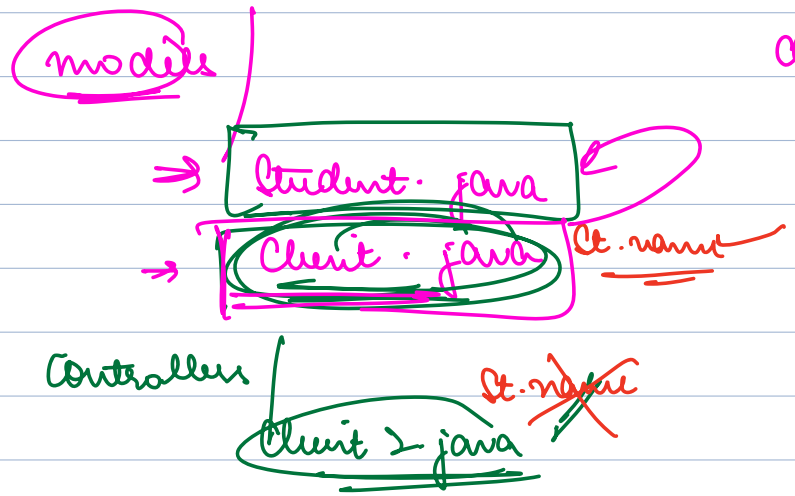
}

DEFAULT

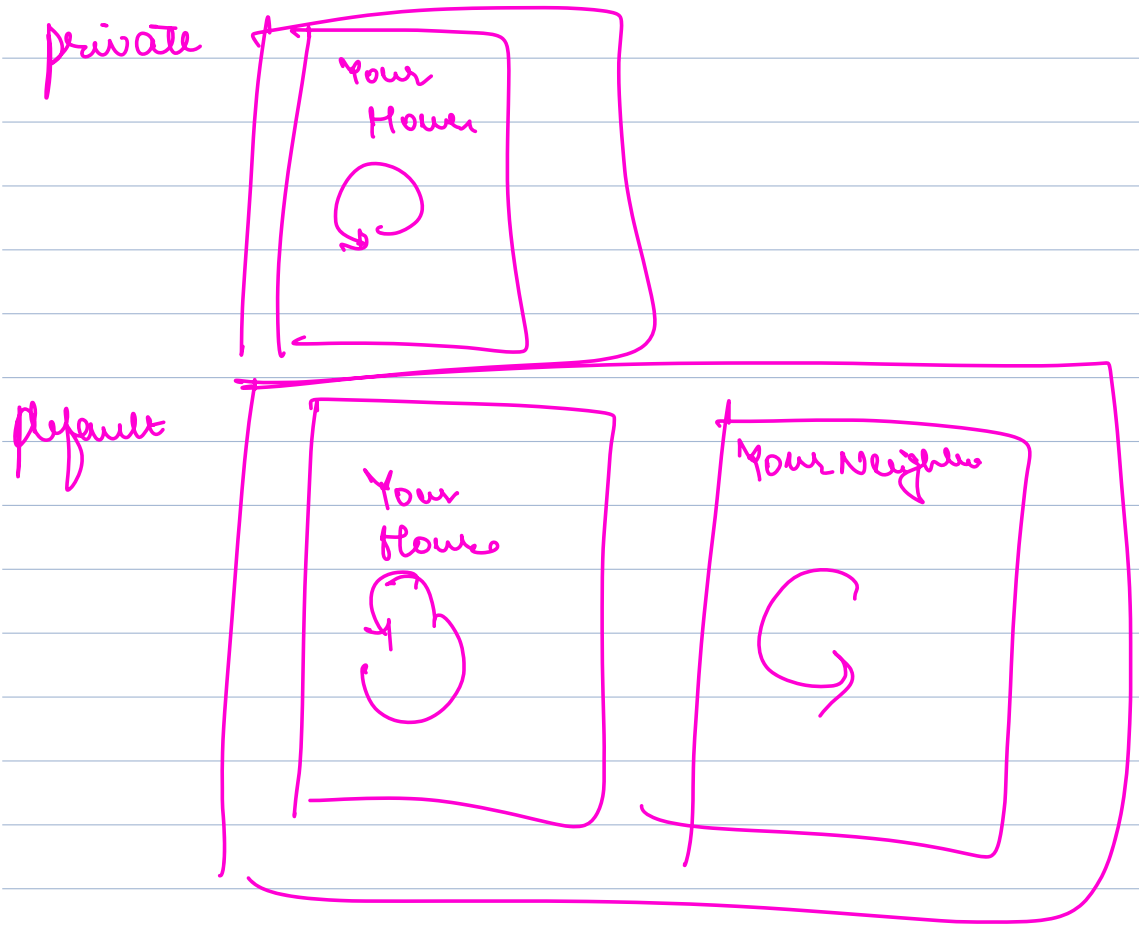
→ Same class

→ Any other class in the same folder.

→ Same package.




```
class Student {
    String name;

    void do1() {
        sout(name)
    }
}

Client {
    psvm() {
        Student st =
        st.name =
    }
}
```

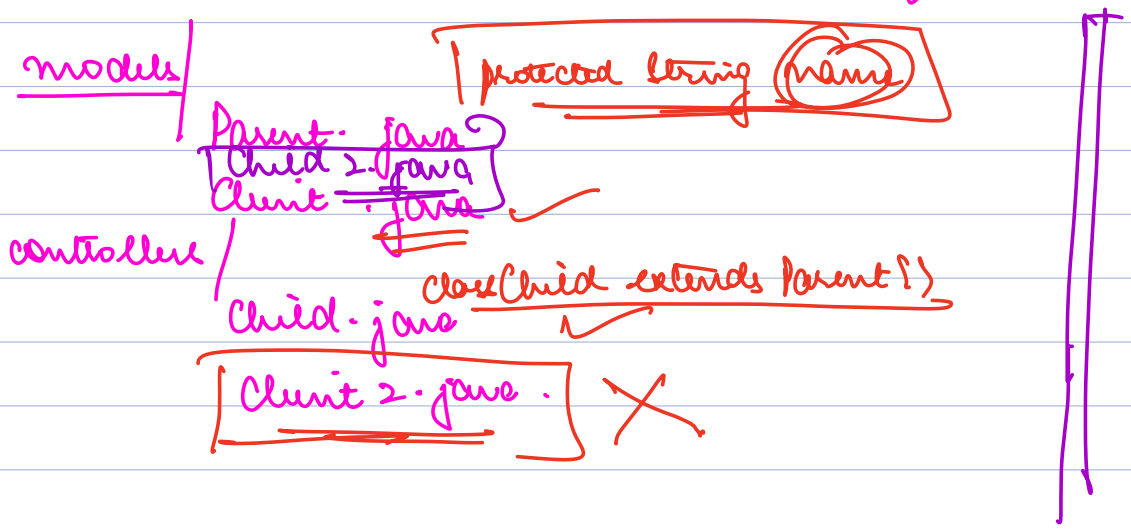

private
Your House
default
Your House
Your Neighbor

Protected

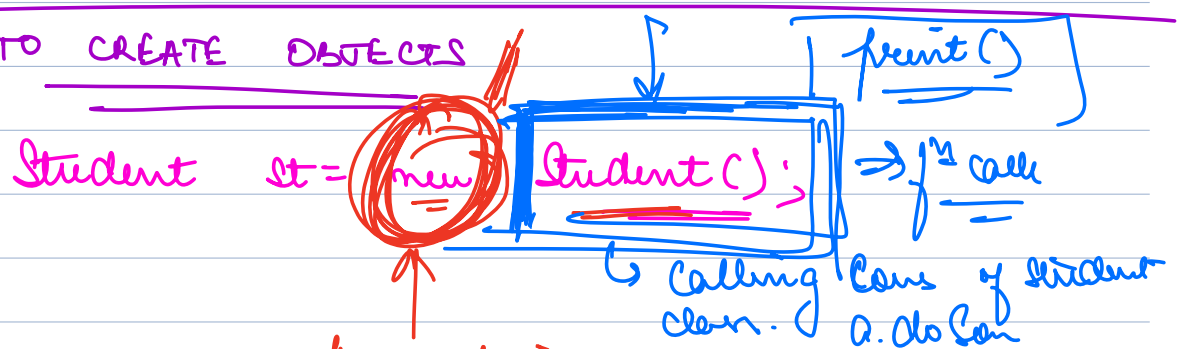
→ Same class

→ Any other class in same folder] default

→ Child class even in other folder



HOW TO CREATE OBJECTS



Keyword in java

⇒ used to allocate space in mem where a new obj will be stored.

CONSTRUCTOR : Special method of a class whose purpose is only to create an new object of the class.

Student {
    String name; → null
    int age; → 0
    boolean married; → false
}

Student s = new Student();

~~new Student("Naman");~~

⇒ Default Constructor is used only if we don't create our own constructor in a class, language by itself puts a default constructor in that class.

⇒ It sets each value of the attribute of the class to the default value of that data type.

→ (1) takes no params
(2) Sets every attr value as default value of that data type
(3) Public AM

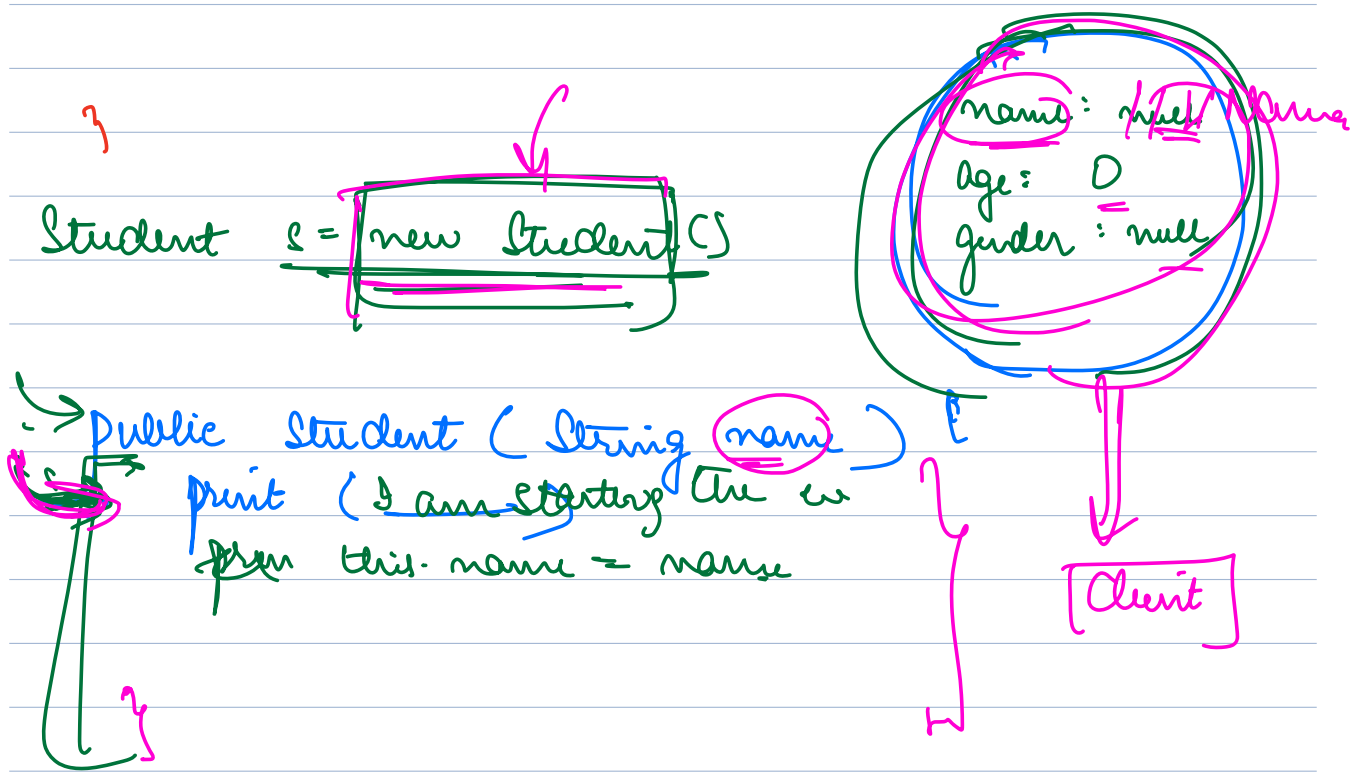
CUSTOM CONSTRUCTORS

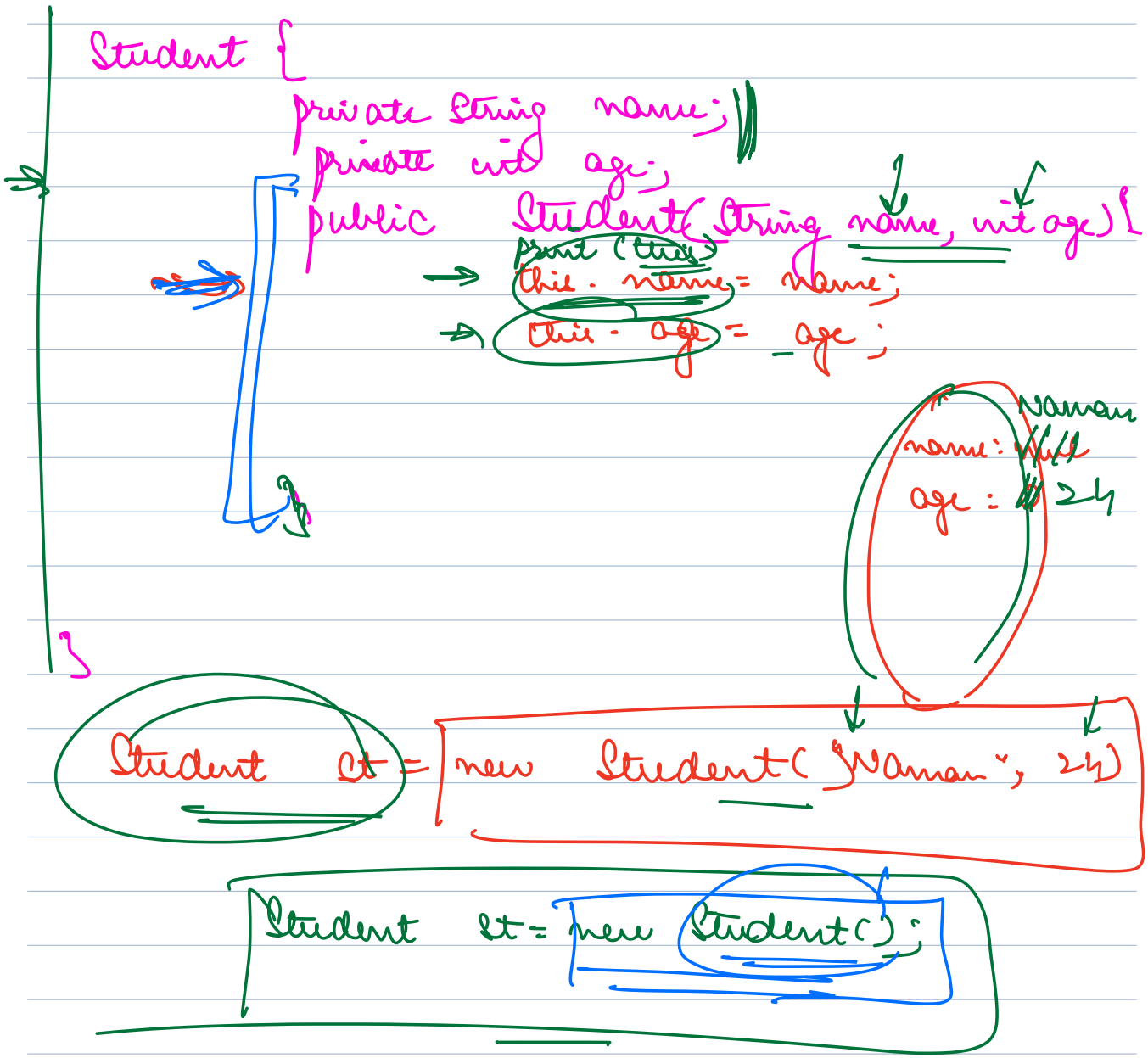
→ method with the same name as name of the class
→ no need to specific return type ⇒ RT is same as the class

```
class Student {
    String name;
    int age;

    public Student(String name, int age) {
        cout("I am the cons");
    }

}

public String getName() {
```

Break till 10:27PM

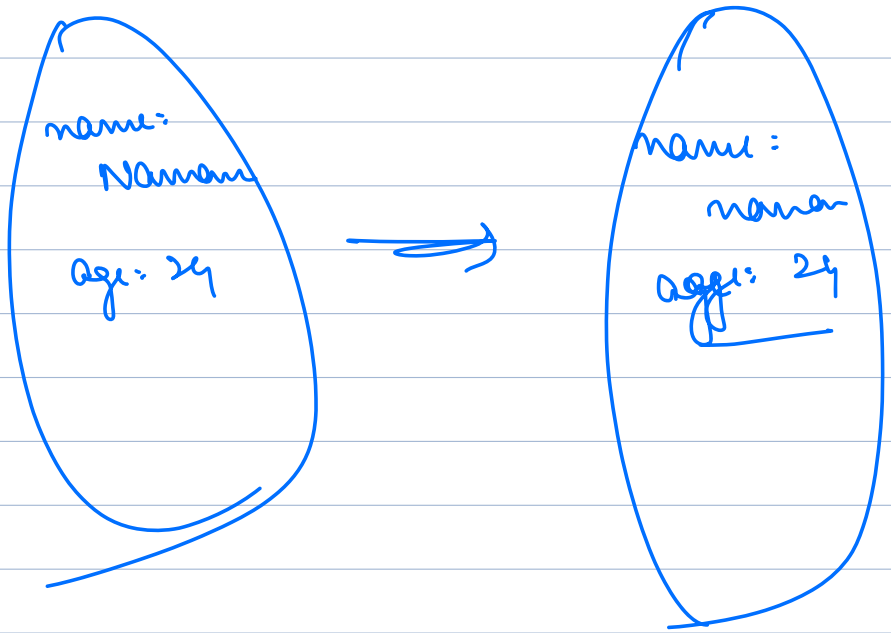
```
public Student(String name) {
    print("Student constructor")
    print(this.name) ? → null
    this.name = name;
}
```

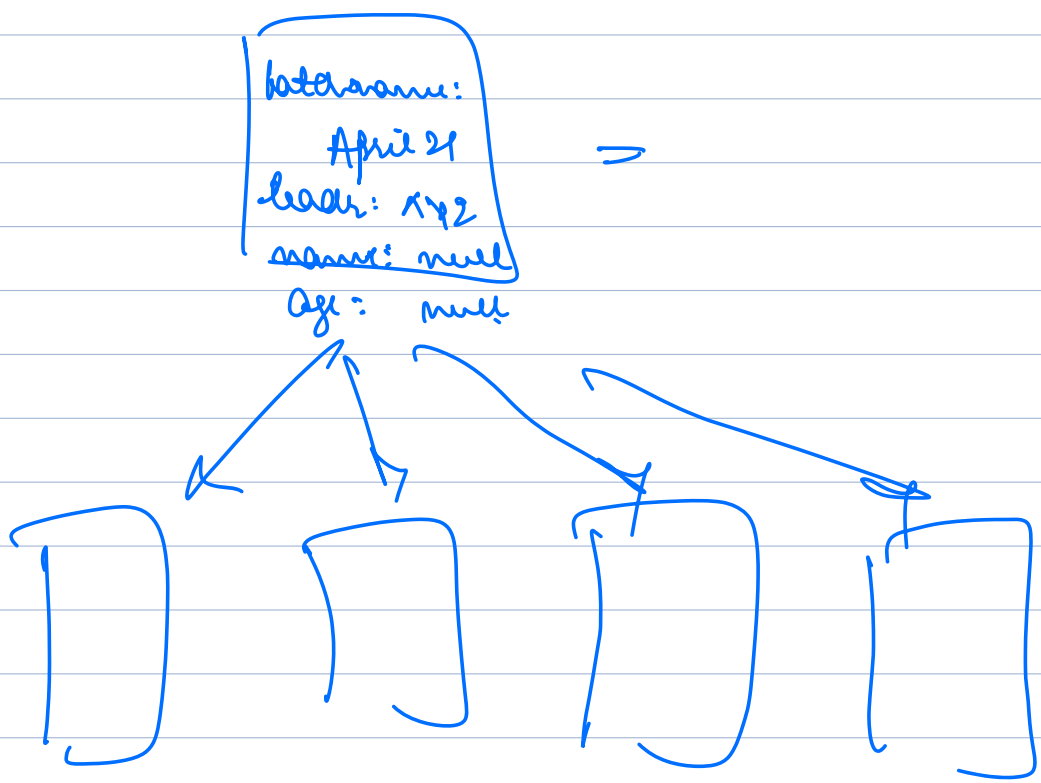




COPY CONSTRUCTORS

→ Sometimes you may want to create a new obj which looks exactly same as an already existing object.


name:
Naman
Age: 24
name:
naman
Age: 24

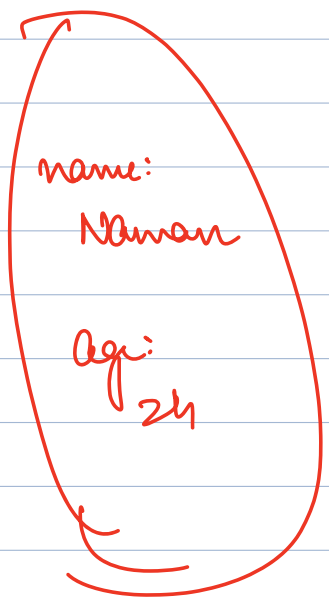

fatehname:
April 21
head: 1x2
name: null
Age: null

Student old = ;

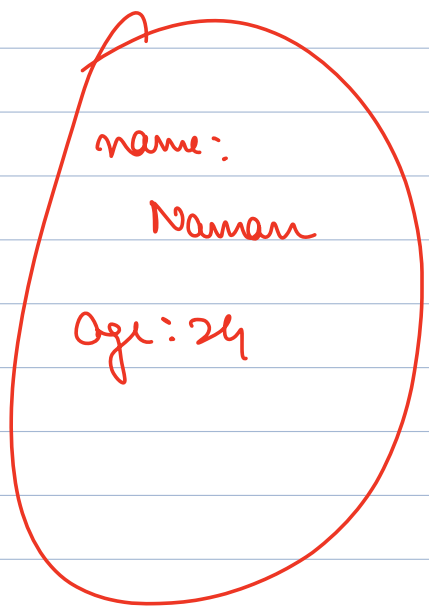
Student s = new Student (old .)

Student old =

Student n = old



100



240

Student {

name: Naman

age: 24

→ public Student (Student other) {

represents current obj → this . name = other . name ;

this . age = other . age ;

}

}

Student old = ____

Student st = new Student (old)

Copy Cons == Custom constructor whose purpose is to create copy of obj passed to it.

Is it deep copy or is it shallow copy

How variables are stored in Java

int age = 12

→ primitive
→ object

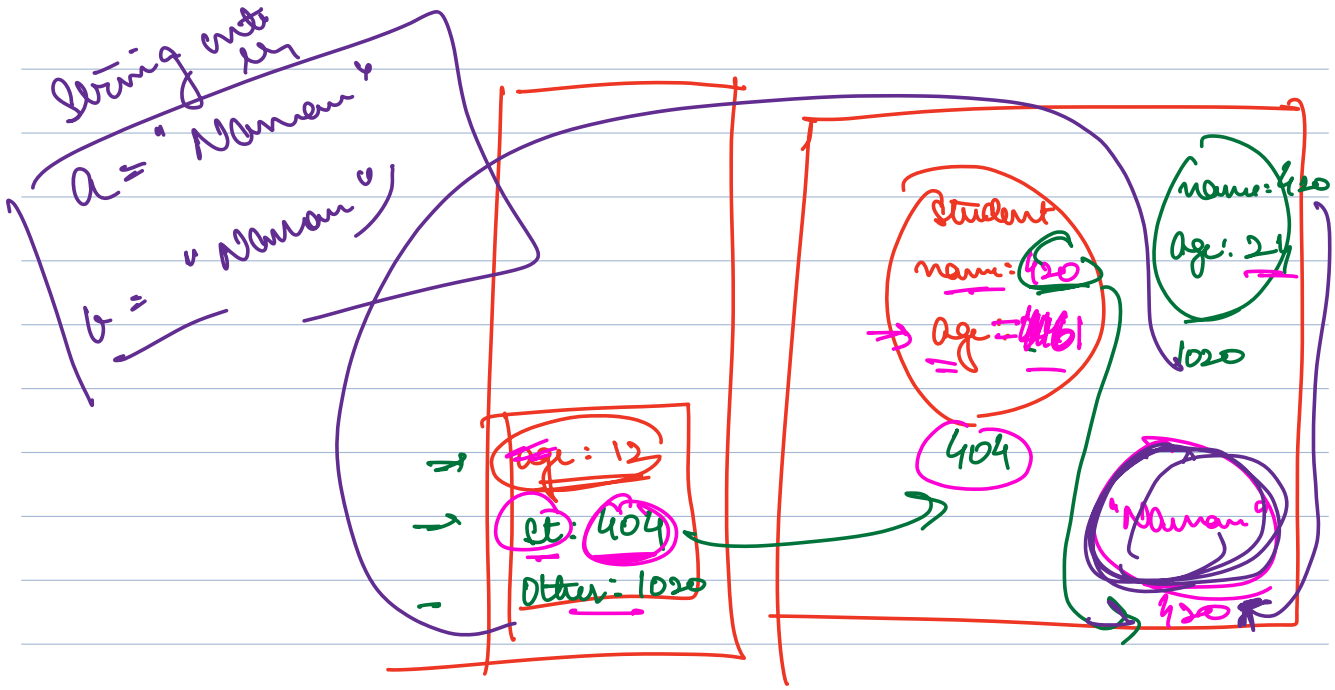
primitive data are stored with the variable names itself → variable name store value.

Student st = new Student()

→ variable name store the address of obj

st.name = "Naman"

st.age = 24



Student (Other) {

this . name = other . name

this . age = other . age

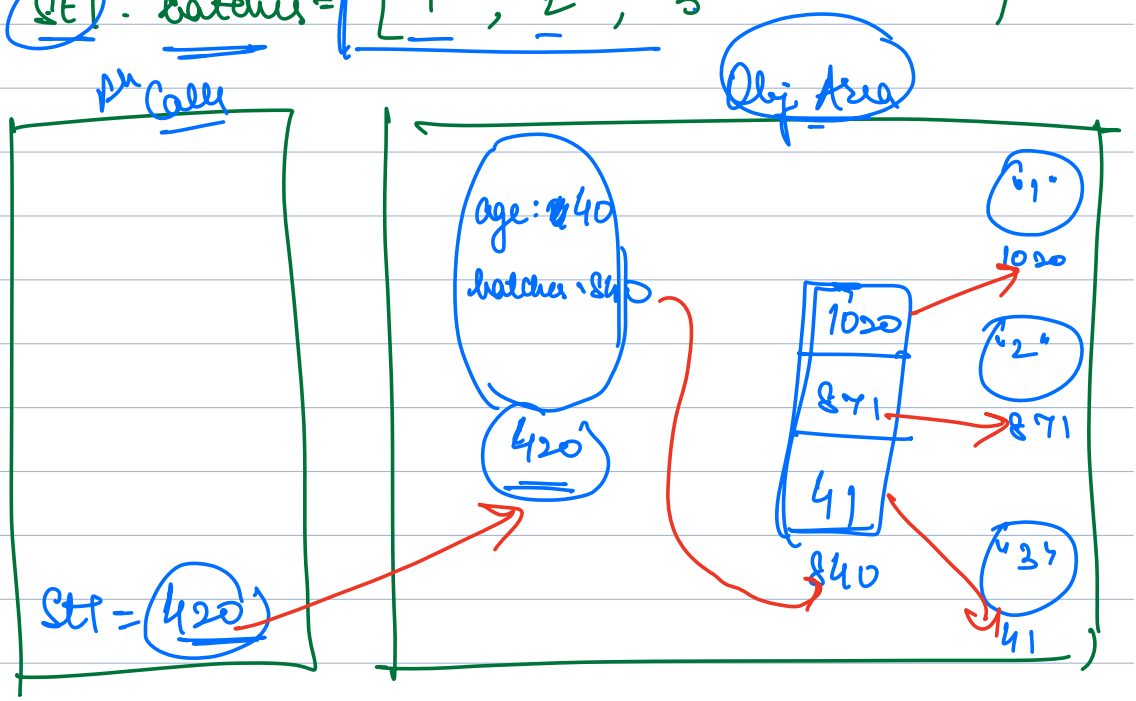
}

Shallow Copy

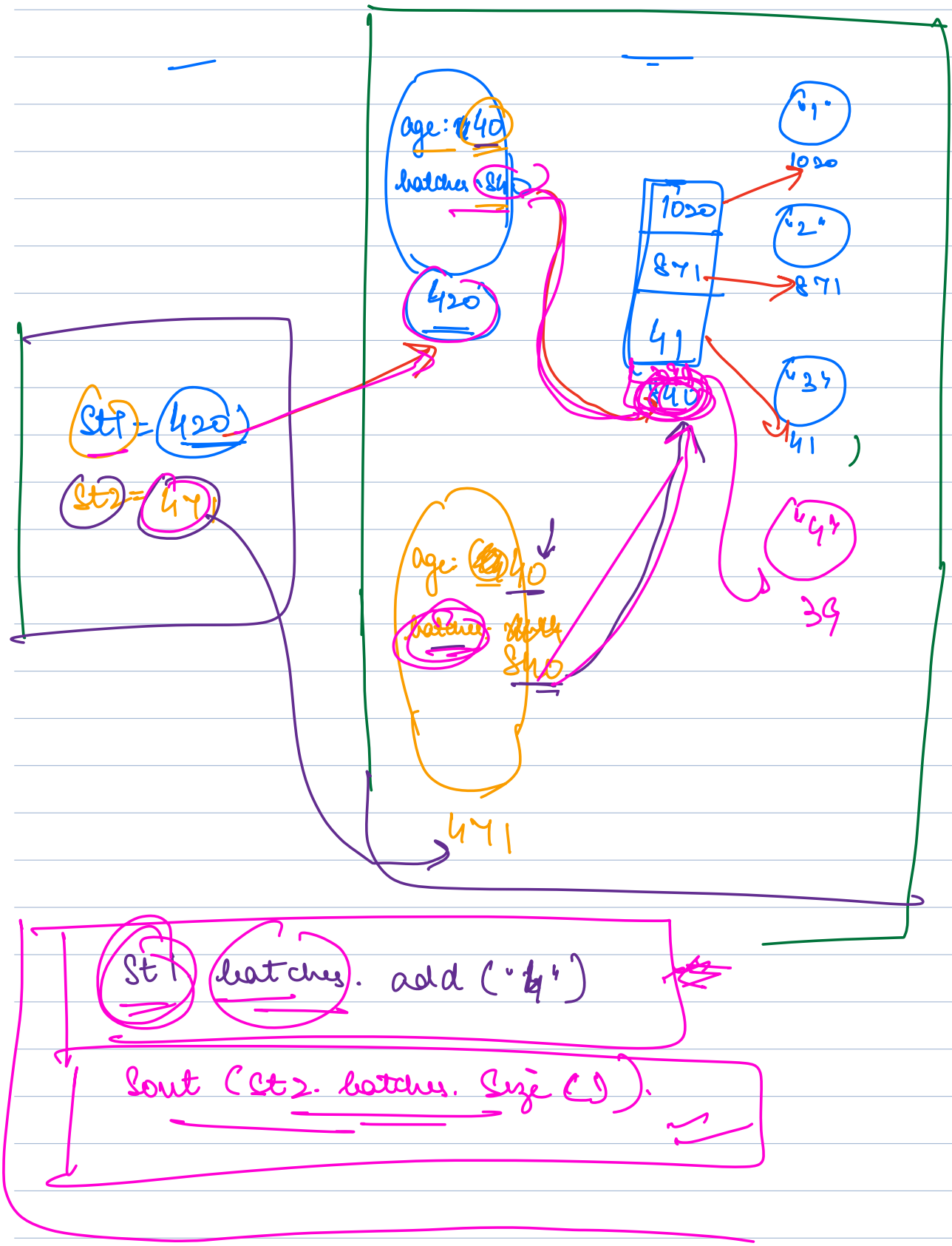
ot age = 61

```
Class Student{
    int age;
    List<Batch> batches;
}
```

```
Student st1 = new Student();
st1.age = 40
st1.batches = {"1", "2", "3"}
```




```
Student (Student other) {
    this.age = other.age
    this.batches = other.batches
}

Student st2 = new Student(st1);
```



st1.batches.add("4")

sout(st2.batches.size());

H/W → Understand how memory is managed in Java