Agenda

→ Access Modifiers

→ Constructors
- → Default
- → Copy
- → Manual.

→ Shallow vs Deep Copy

→ Pass by value vs Pass by Reference.

ACCESS MODIFIERS.

1. Public : Anyone can access from anywhere
2. Private : Can only be accessed within the class.
3. Protected : Can be accessed within the package & Child class outside the package.
4. Default : Can be accessed within the package only.

```
Class Student{
      String name;
      int age;
      double psp;
      String batch;
}
```

Memory Allocation

Default Constructor

```
Student st = new Student();
```

looking like a fun.

```
Student rohan = new Student();
```

Default Constructor.

If we don't create our own constructor in a class, a default constructor is provided.

⇒ Default constructor creates a new object of the class and initializes all the attrs of the class to their default values.

```
Class Student {
    String name;
    int age;
    double psp;
    String batch;

    Student() {
        name = null;
        age = 0
        psp = 0.0
        batch = null;
    }
```

⇒ Special function.

⇐ Automatically generated by JAVA but not visible to us.

⇒ Default Constructor.

1) No return type
2) Constructor name will be same as of the class name.
3) Initializes the class attrs with default values
4) public.

CUSTOM CONSTRUCTORS.

```
Student st = new Student();
    st.name = "Nikhil"
    st.psp = 85.0
    st.batch = "MWF 4PM";
    ______
    ______
    ______
```

```
Student st = new Student("Nikhil", 85.0, "MWF 9PM",
                                        ...);

Class Student{
    String name;
    int age;
    double psp;
    String batch;

    Student(String name, age, psp, batch){
        this.name = name;
        this.age = age;
        this.psp = psp;
        this.batch = batch;
    }
```

Belongs to the current object of the class ⇒ this.name

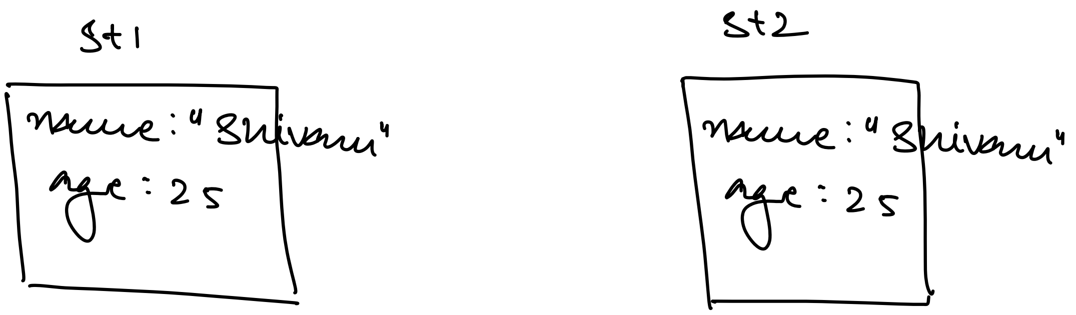
⇒ How custom constructor works.

1. It initializes the object with the default values
2. Starts executing further lines to set the attribute values

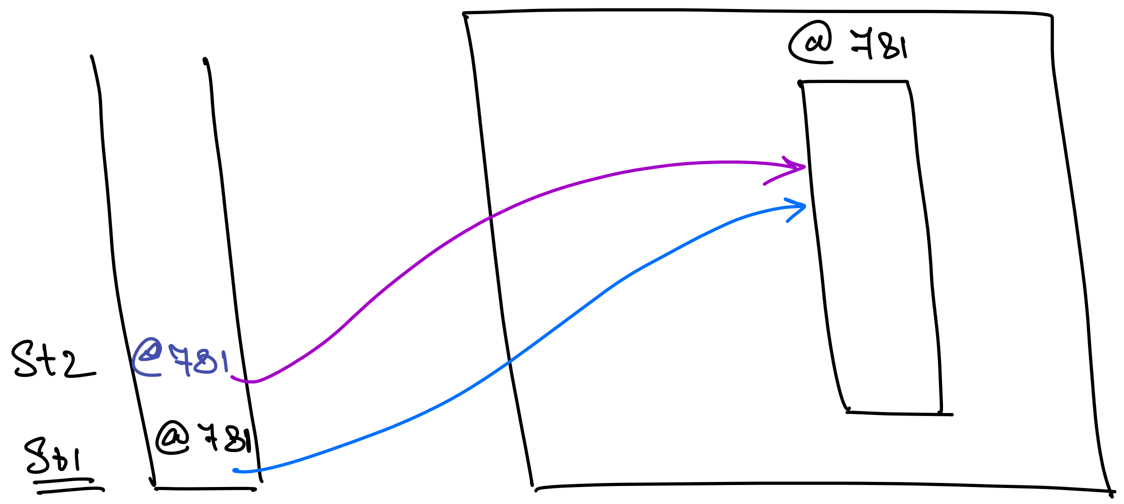
COPY CONSTRUCTOR.

→ We already have an object of Student.

→ We want to create a new object of Student that has exact same values of the existing object.



✓ Student St1 = new Student();



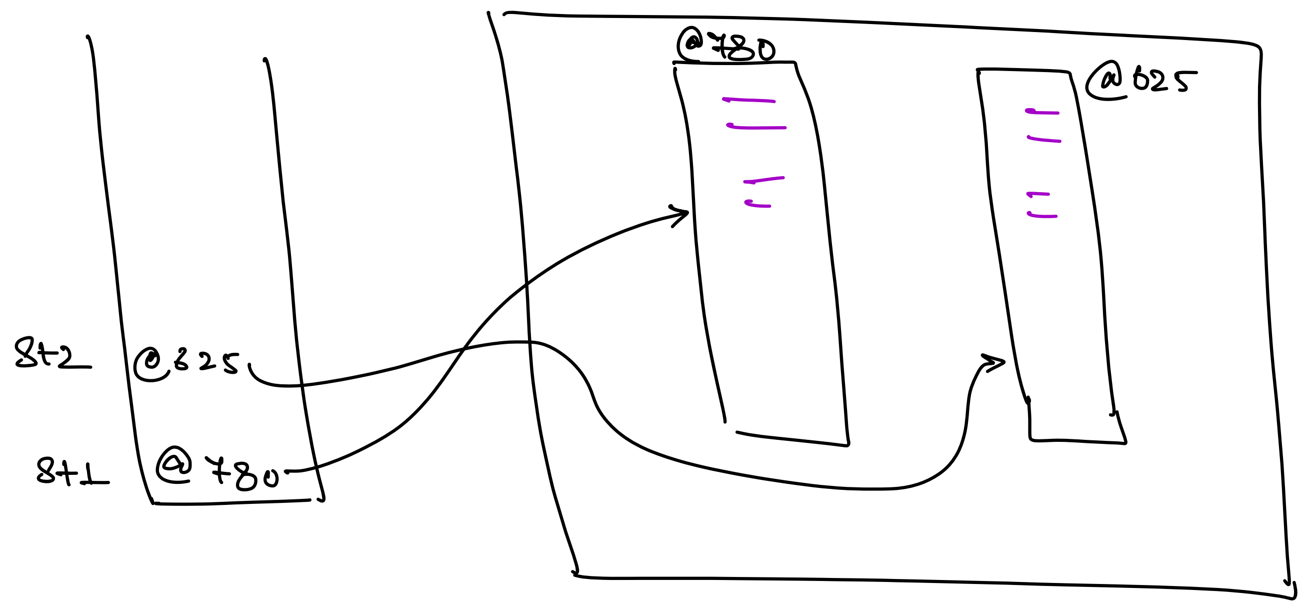
```
Student st2 = st1;
```

@781

Not even a Copy

#

```
Student st2 = new Student();
st2.name = st1.name;
st2.age = st1.age;
```



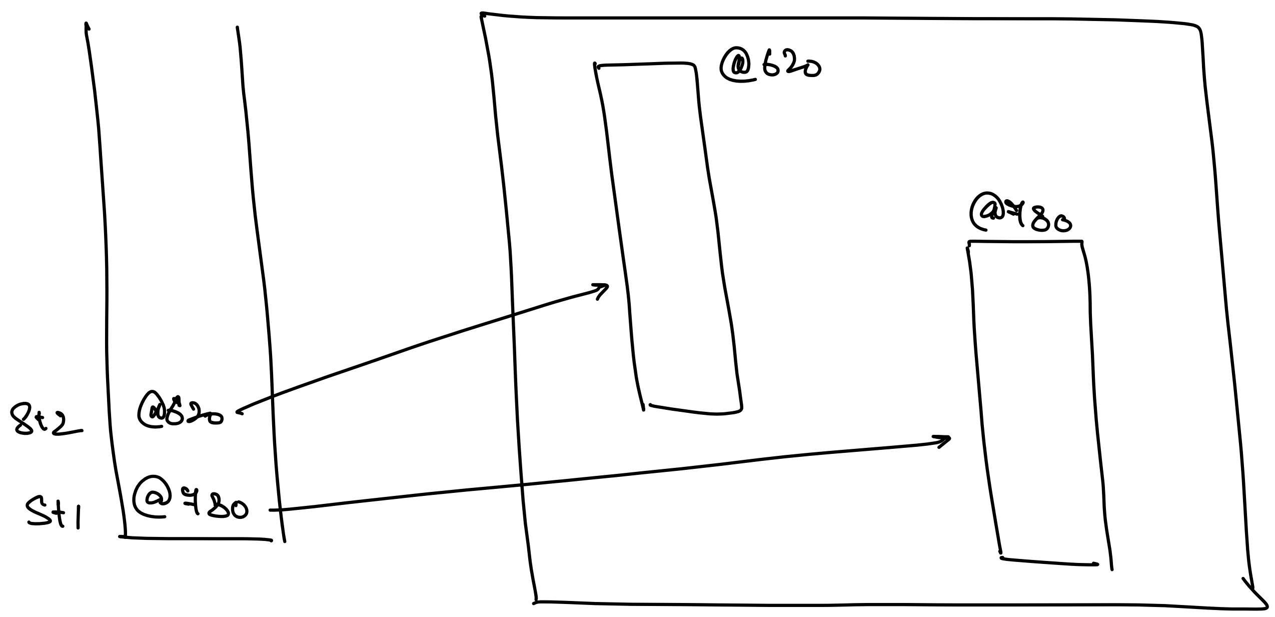
⇒
```
Class Student {
      String name;
      int age;
      double psp;
      String batch;

      Student (Student st) {
          this.name = st.name;
          this.age = st.age;
          ____
          ____
          ____
          ____
      }
}
```

Copy Cons.

Student St2 = new Student (St1);

Copy Object.

⇒



JAVA MEMORY.

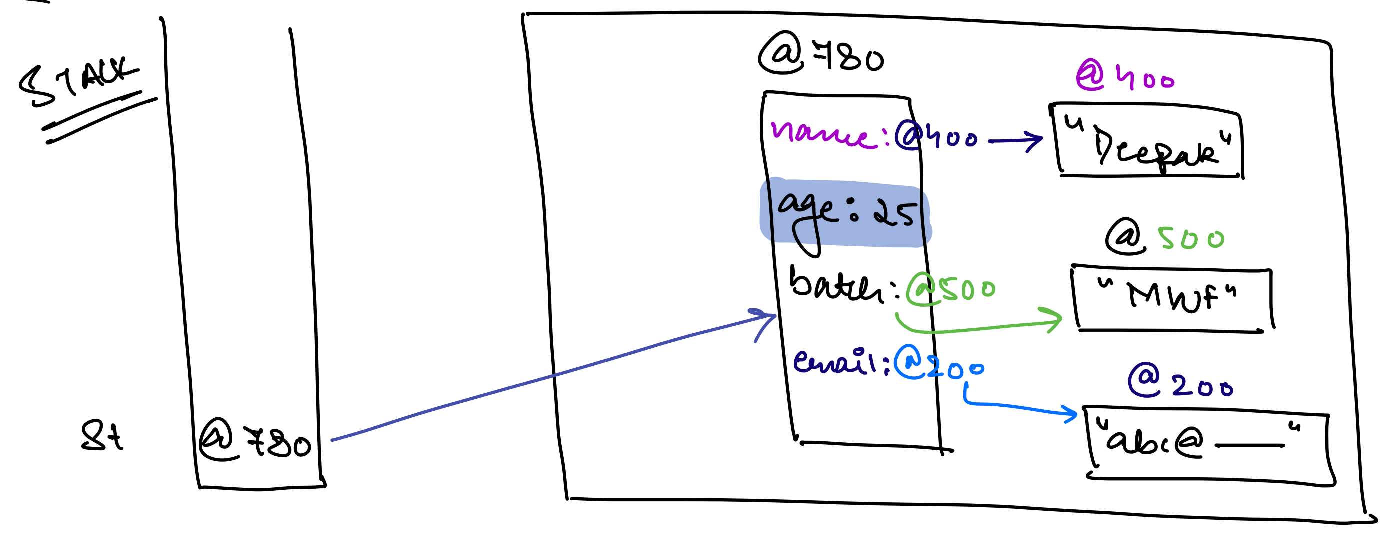
→ Primitive : Simple [int | bool | double]

→ Non Primitive : Object [String | Objects of any class].

Student St = new Student();

```
class Student {
    String name, => Non primitive
    int age;      ] Primitive
    double psp;   ]
    String batch; => Non primitive

    Student (Student St) {
        this.name = St.name;
        this.age = St.age;
        ___
        ___
        ___
        ___
    }
}
```

St.name = "Deepak"
St.age = 25;



Sout (St.name)

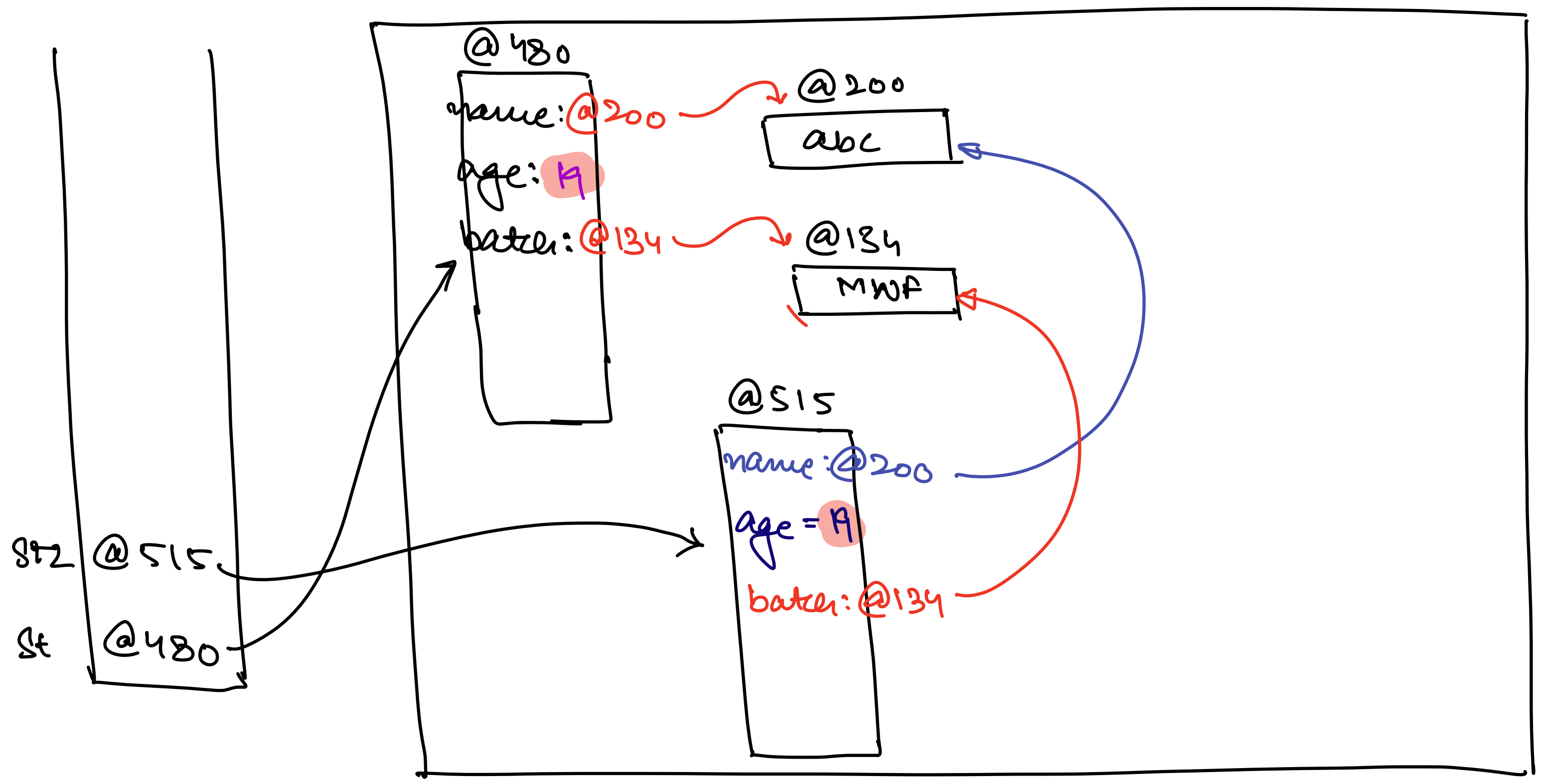
↓

780.name

↓

$\boxed{780.400}$ ⇒ Deepak

⇒



Student (St) = new Student()

Copy

Student St 2 = new Student (St);

St2.name = St1.name; = @ 200

St2.age = St1.age

St2.batch = St1.batch;

St2.name = "XYZ"

⇒ SHALLOW COPY.

⇒ DEEP COPY.

Class Student {

List<Lecture> Lectures

name

age

}

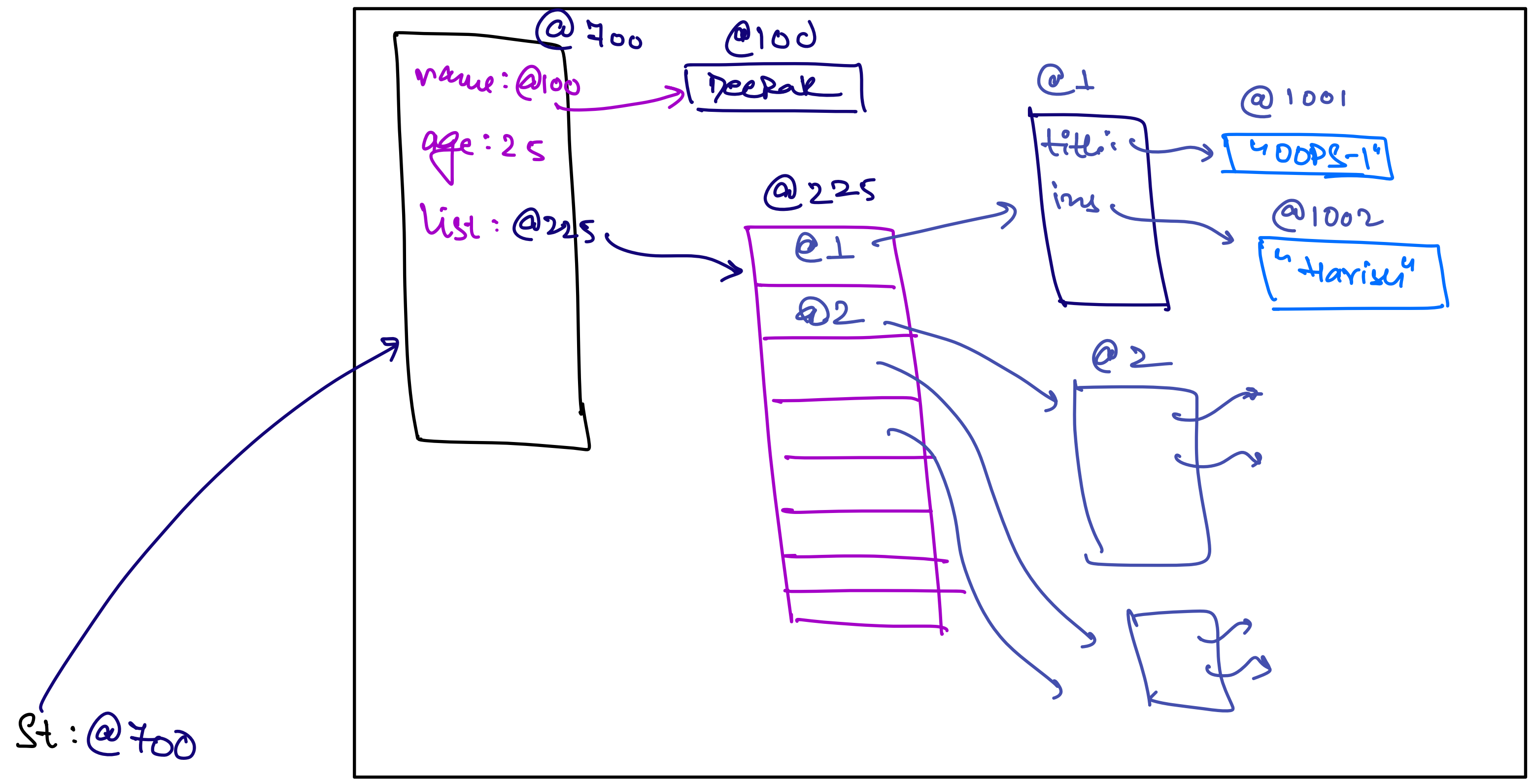
Lecture {

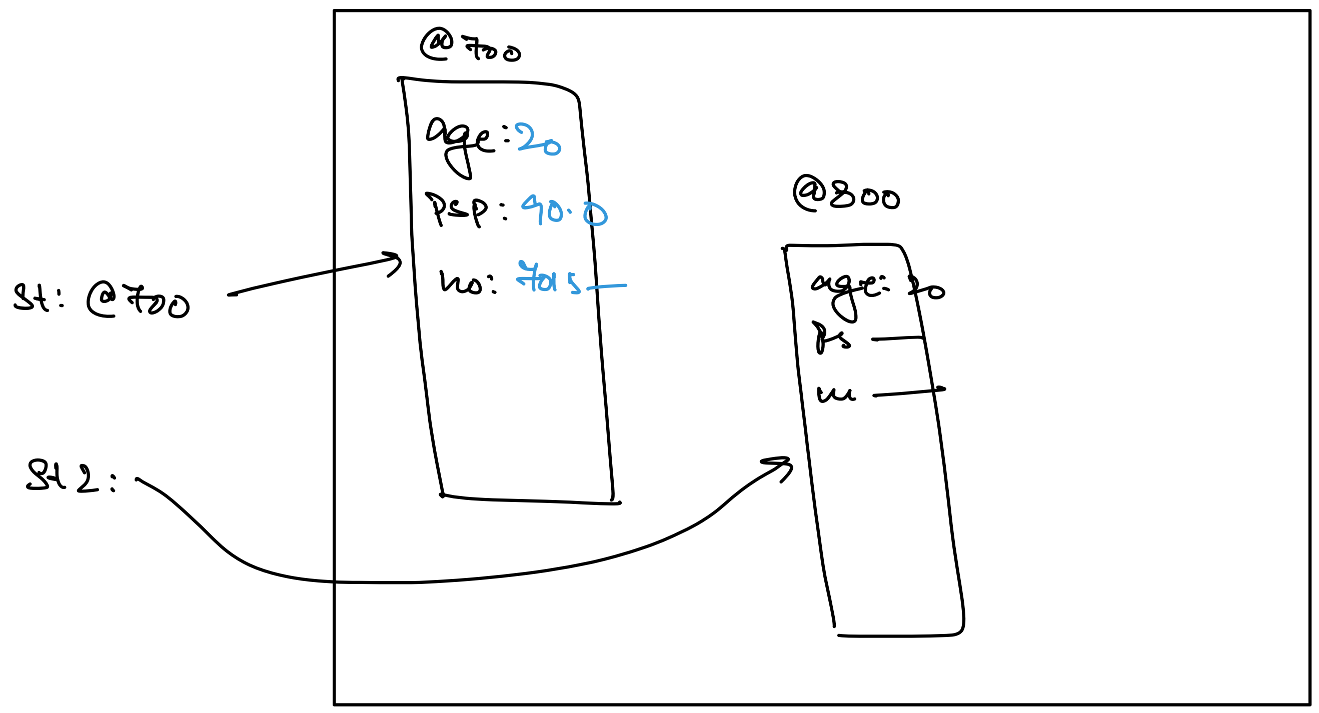
title

Instructor

}

⇒ Student st = new Student();





```
Student st2 = new Student(st)
```

@700

```
⇒ void doSomething(Student st){
   ________  st = new Student();
   ________  @400
   ________
   ________
  }
```

@700

```
Student st = new Student();
doSomething(st);
            700
```

⇒ Pass By Value : Passing Object address inside the fun.

Destructor.

- Destroys the Object.
- In Java, it is called automatically by GC to destroy unreferenced object.

—————— * ——————