Agenda

① Encapsulation @ 9.05
② Access Modifiers
③ Constructors

LLD | OOPS

↳ 1 Principle → ABSTRACTION
↳ Hide details
3 Pillars
↳ Encapsulation
↳ Inheritance
↳ Polymorphism

ENCAPSULATION

CAPSULE

1) Holds the medicine together
2) Protects medicine from contamination due external environment.

Encapsulation in OOPs

① It will store all the attrs & behaviors together
② Protect the data (attrs and behaviors) from illegitimate access.

Access Modifiers

① Public : Anyone can access the members of the entity

② Private : No one can access the member outside the entity

→ only accessible within the entity.

→ Even child can't access the members

③ Protected : Only entities within the same package | folder can access.

→ Only the child class can access outside the package

④ Default : Only within the same package

→ Even the child can access within the package

	Same Entity	Same folder (PKG) (Not child)	Child Entity (Same PKG)	Child Entity (diff PKG)	Anywhere
private	✓	X	X	X	X
Default	✓	✓	✓	X	X
Protected	✓	✓	✓	✓	X
Public	✓	✓	✓	✓	✓

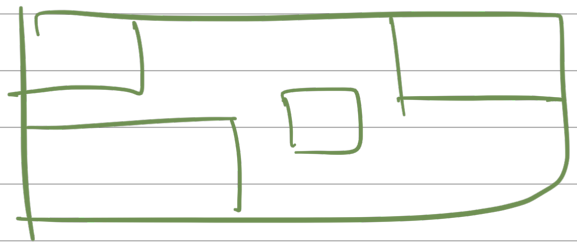
decreasing restriction

```
Class Student {

    String Name;
    int age;

}
```

Class is a blue print of Idea



on paper

Class represents a structure of idea

- attrs
- behaviors / func"

```
Student {
    String name;
    int age;

    void login();
}
```

2) Class by itself doesn't take any space in the Memory.

2) Class is not a real entity.

2) Multiple instances can be created

Object : Real entity

2) Occupies Memory.

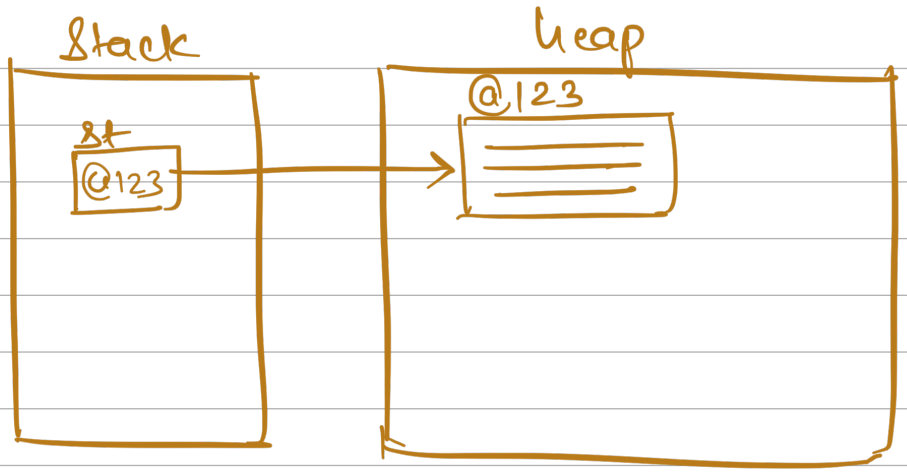
2) All the objects are completely independent

```
functionName();
```

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

- Student → data type
- st → Variable name
- new → Allocates the Memory
- student() → Create the object of the class → Constructor

How object gets created in Memory



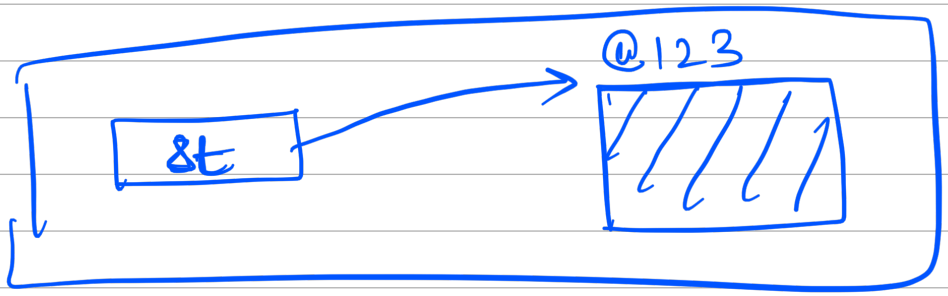
Stack
⇒ Variable Names

heap
⇒ actual object

→ Student st = new student();

const

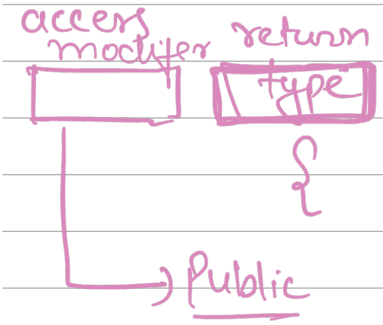
Class



st.name = "ABC"

Student st = new Student()

st.name → NULL



Student (String name, int age)

Constructor returns the object of the class

Ex:- new Student();

as it is fixed we don't need to specify

2) In general, constructor have public access

Constructor can be private

Singelton
- If constructor is private, we can't create object from outside
- Within class object can be created

this → ref to current object of class

```
Public Student (String name, int age)
{
    this.name = name;
    this.age = age;
}
```

→ Custom Constructor
Parametrised Constructor

How internally custom constructor

1) Creates an object with attrs set to default values

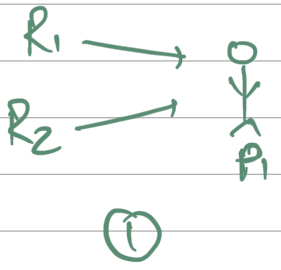
2) Start executing lines in the Custom constructor

```
Student st = new Student("Nikunj", 31);
```

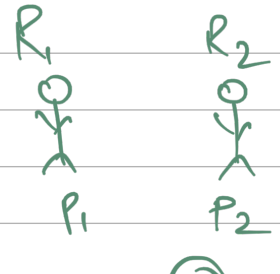
```
Student st = new Student();
St.name = "Nikunj";
St.age = 31;
```

The name of constructor is always same as the Name of the Class.

Clone



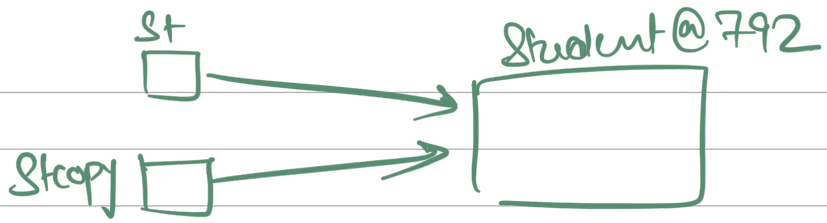
①



②

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

A New ref has been assigned. NO Actual Copy happened

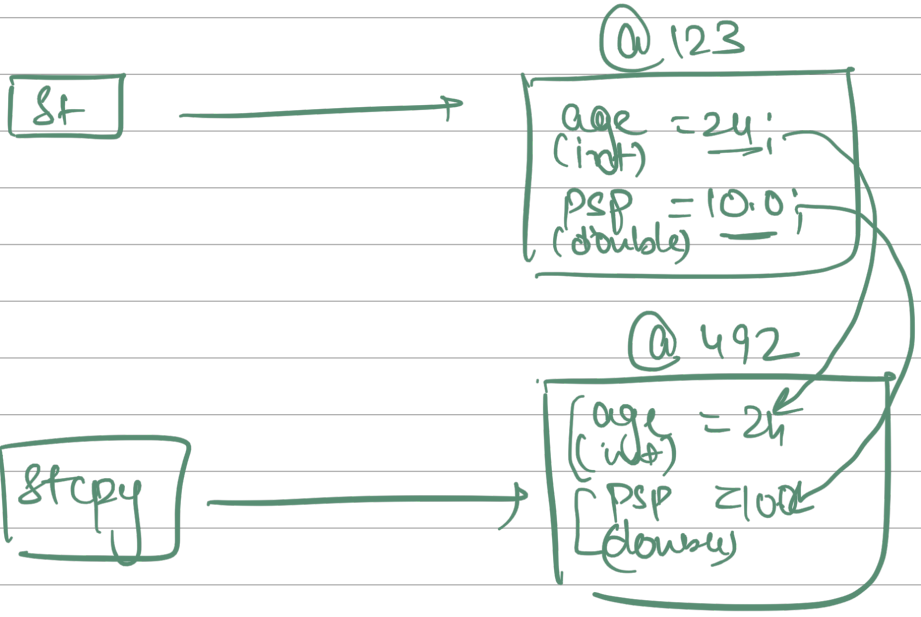



```
Student copy (Student st)
{
    Student Stcpy = new Student();
    Stcpy.name = st.name;
    Stcpy.age = st.age;
    return stcpy;
}
```

If age is private

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

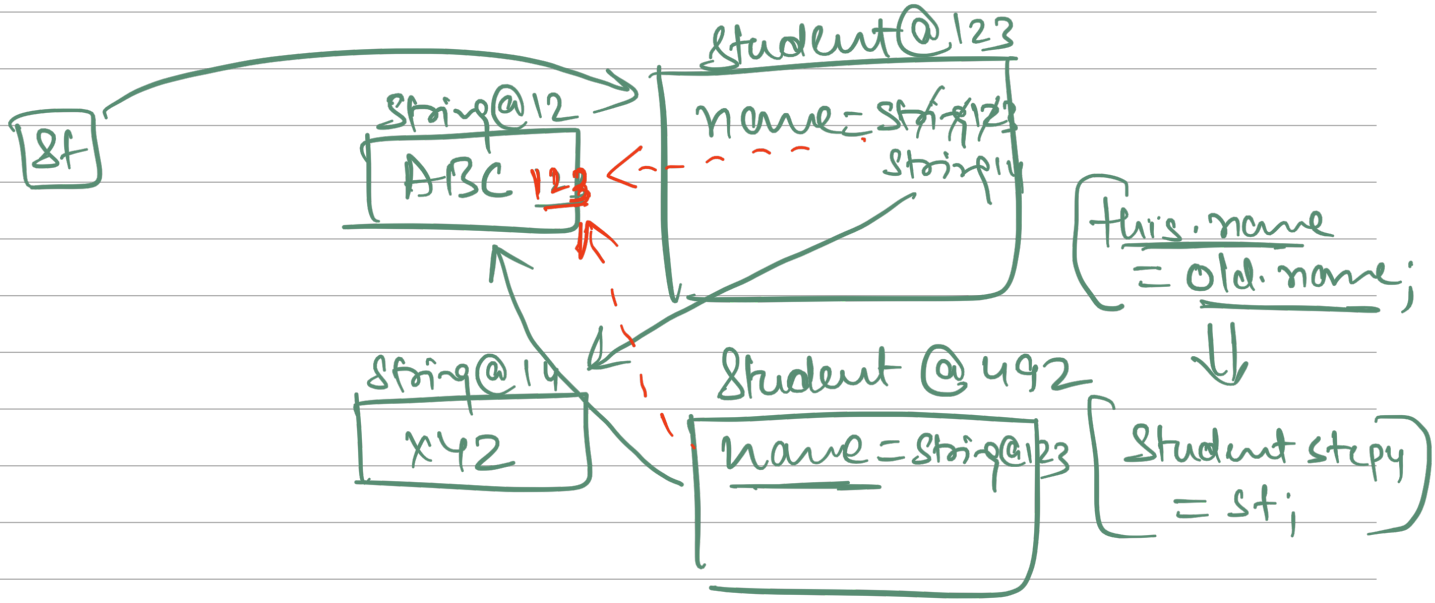
→ It will create object with default values



Datatypes ① Primitive
② Non primitives | Objects

For primitive, always a new memory will be allocated when object is getting created

For objects, always a new variable will be created



Shallow copy : When we create a copy of an object but behind the scences it still points to same attributes of old object

St.name = "X42"

St.name.append(123);

Deep Copy : No data sharing

this.name = old.name

instead

this.name = new String(old.name);

Fun fact

In all partical scenarios, you can't create a perfect deep copy.

Student
- List<Class> classes
  - Instructor Inst
    - List<Batches>
      - List<Mentors>

Once the work of the object is done it will automatically be collected by garbage collector

Destructor

opposite of constructor

Will automatically called by the GC

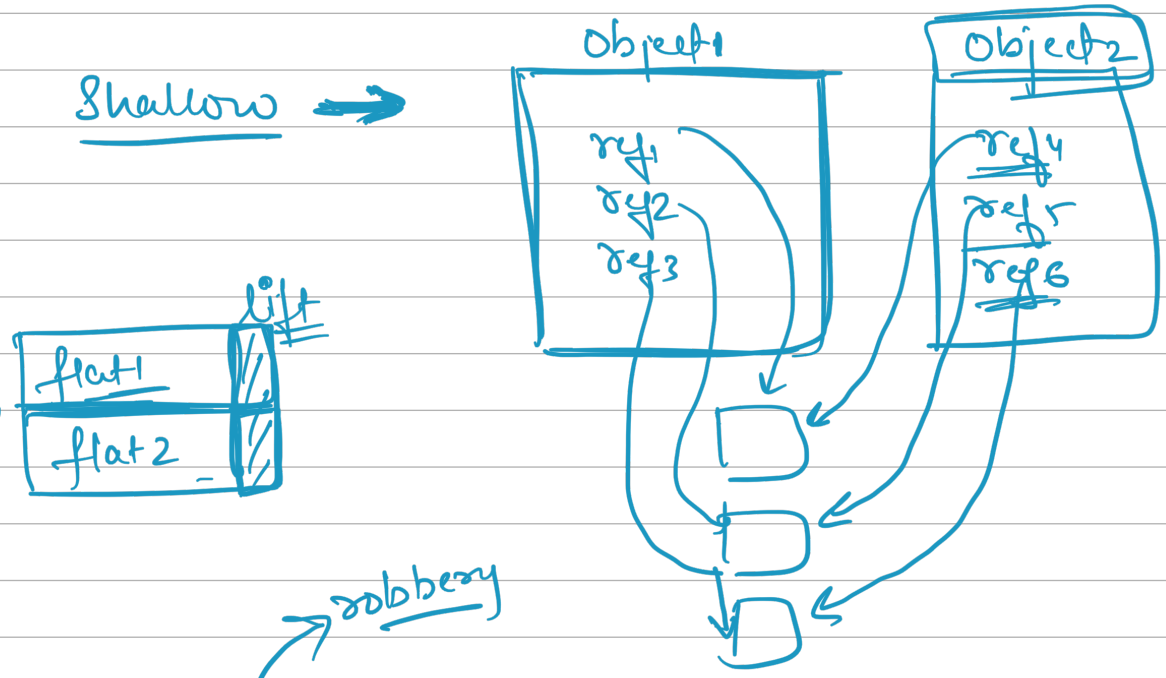
this.age = old.age → Private

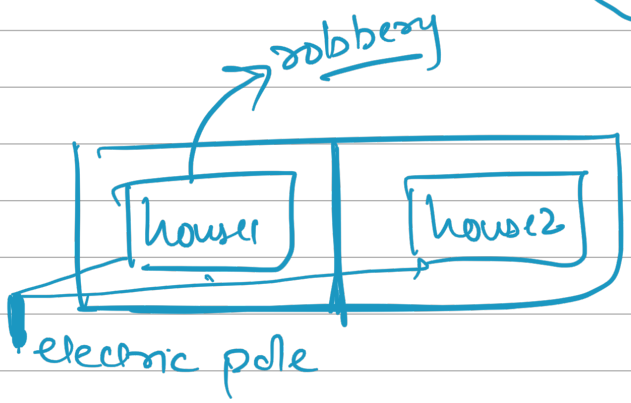
this

```
Student (String name)
{ this.name = name;
}
```

→ object created here will refered by this

Shallow →



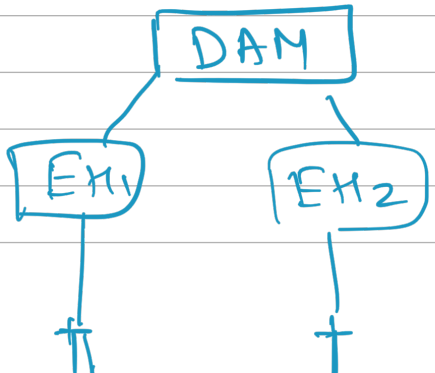


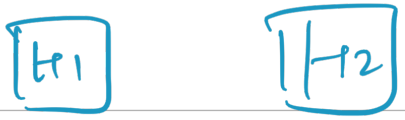
deep copy

Student S2 = new Student(S1);

→ shallow / deep

String




t+1
t+2