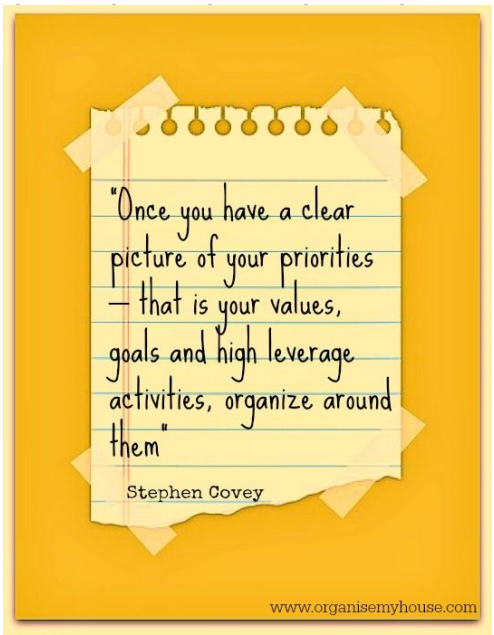
OOPs 2



Good Morning

01. Constructor
02. Types of constructor
03. Deep copy & shallow copy
04. Inheritance
05 Polymorphism
06. Method Overloading & Method overriding

* Class → Blueprint of an object.

* Object → Instance of class

* **Constructor**

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

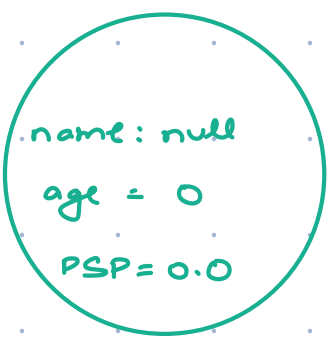
```
Student s1 = new Student();
```

constructor

* **Default constructor**

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

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

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



* Default constructor will not be created if we are going to create a manual constructor.

Summarising the default constructor:

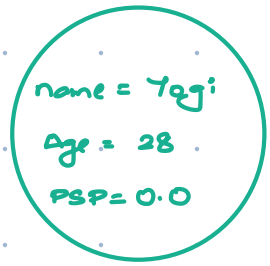
1. Takes no parameter.
2. Sets every attribute of class to it's default value (unless defined).
3. Created only if we don't write our own constructor.
4. It's public i.e. can be access from anywhere.

* Manual constructor

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

    Student (int age){
        this.age = age;
    }

    Student (String n, int a, double p){
        this.name = n;
        this.age = a;
        this.psp = p;
    }
}
```

```
Student (){
}
```

```
Student s = new Student ("Yogi", 28, 0.0 )
```




```
Student s1 = new Student (27);
```


```
Student s2 = new Student ( ); // Error
```

↓

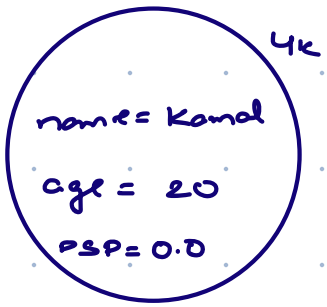
Default constructor is not created

* **Copy constructor**

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

    Student ( ) {    }

    Student ( student st ) {
```

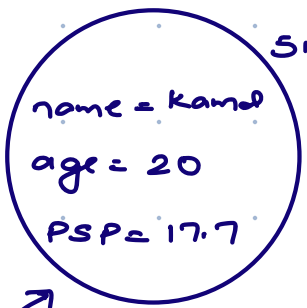
```
        this.name = st.name
        this.age = st.age
        this.psp = st.psp;
    }
}
```

```
Student s = new Student();
s.name = "kamal";
s.age = 20;
```



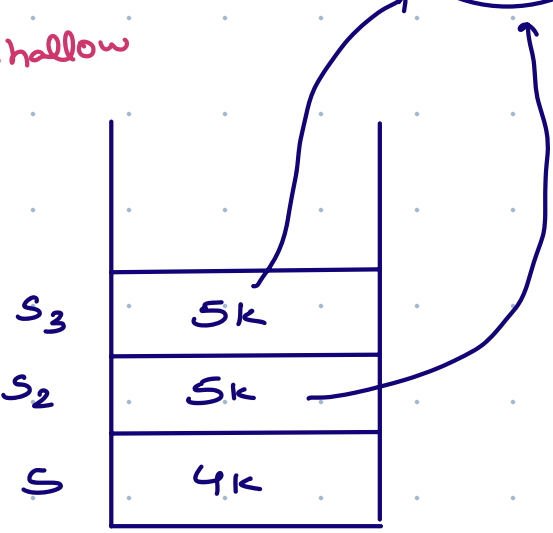
invokes the copy constructor & create a deep copy

```
Student S2 = new Student(s);
```



create a shallow copy

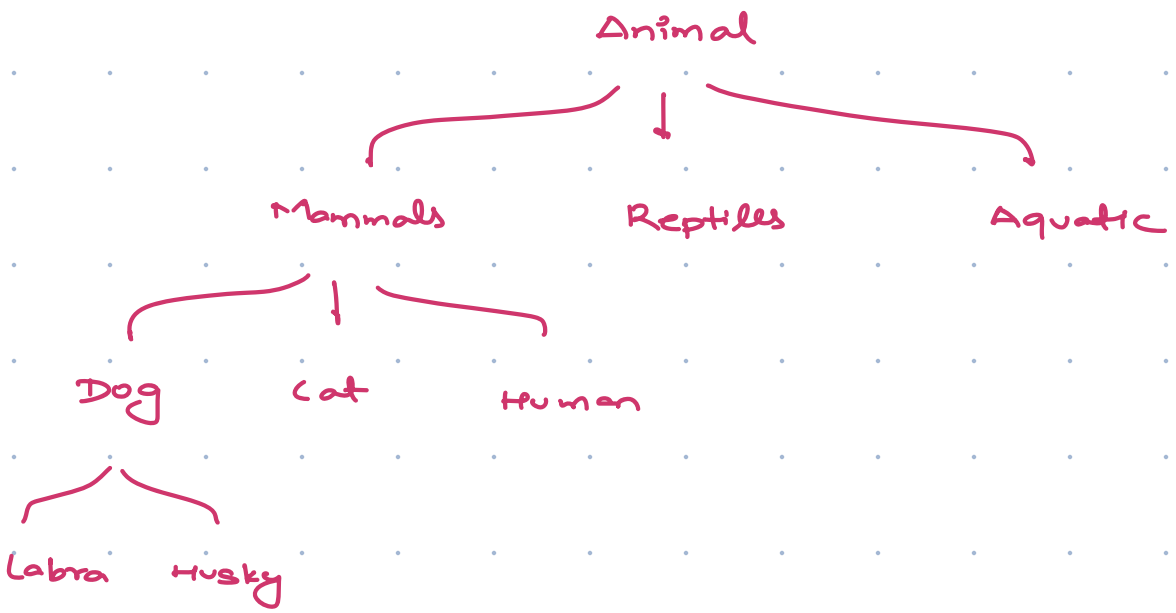
```
Student S3 = S2
S3.psp = 17.7;
```



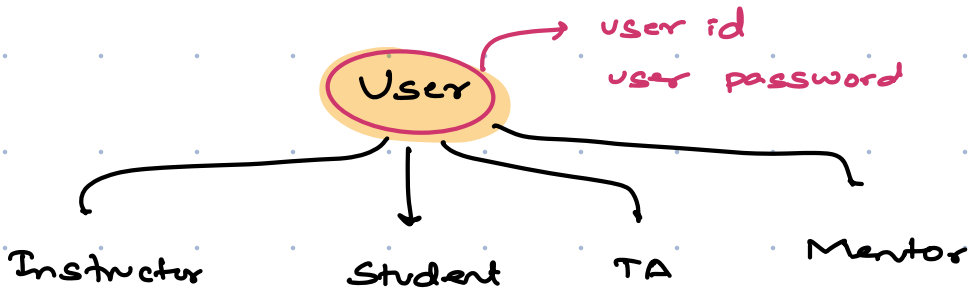
Deep copy → We are creating a new object through copy constructor

Shallow copy → when we are only passing the reference of the object to a new variable.

* **Inheritance**



Representation of hierarchies in OOPS is known as Inheritance



* Parent - child relationship

```
class User {
    String username;
    void login(){
    }
}
class Instructor extends User {
    String batchName;
    double avgRating;
    void scheduleClass() {
    }
}
```

- **In Python:** The inheritance is indicated using **parentheses**.
  For example: `class Subclass(SuperClass):`
- **In C++:** The inheritance is specified using a **colon**.
  For example: `class Subclass : public SuperClass { };`
- **In C#:** The inheritance is specified using a **colon**.
  For example: `class Subclass : BaseClass { }`

Instructor i = new Instructor();

i.username = "yogi"; ✓

i.avgrating = 4; ✓

username = null
batchname = null
Avg rating = 0.0

highest level of Abstraction

A
↑
B
↑
C
↑
D

D obj = new D();

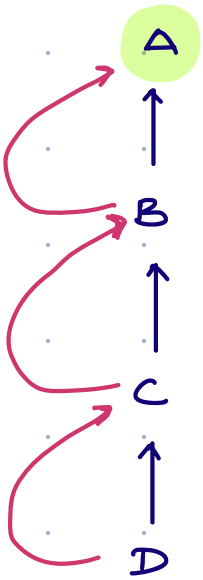
```
class abc {
    int x;
    private int y;
}

class user extends abc {
    int z;
}
```

User obj = new User();

x = 0
y = 0
z = 0

obj.x = 10
obj.z = 20
obj.y = 30

* Try to think how we can show that private members are also getting inherited.



D d = new D();

→ Every time we try to create an obj of child class, first it will invoke its parent class constructor & once that is finished, then only it will invoke its own constructor

```
public class C [redacted] {
    C() {
        System.out.println("Constructor of C");
    }
    C(String a) {
        System.out.println("Constructor of C with params");
    }
}
```

```
public class D extends C {
    D() {
        super("Yogi"); // this should be first line
        SOP("Constructor of D");
    }
}
```

D obj = new D();

Output
→ Constructor of C with params;
→ Constructor of D;

8:26 AM → 8:36 AM

* Poly morphism

Many form

Animal a = new Dog(); ✓

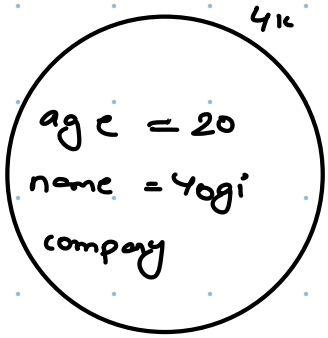
Dog d = new Animal(); ✗

```
A {
    int age
    String name;
}
```

```
B extends A {
    string univ;
}
```

```
class C extend A {
    string company
}
```

A a = new C(); → can't access the child class attributes



a.age = 20

a.name = Yogi

a.company = "ABC" → compile time error

C c = new A()

c.company = X

↳ Compile time error

* Two types of Polymorphism

01. Compile time polymorphism

02 Run time polymorphism

* Method Overloading

```
class A {
    void hello() {
        =
    }
    void hello(String name) {
        =
    }
}
```

→ Compile time polymorphism

Method Overloading – Function with same name but diff parameters

```
hello():
hello("Yogi");
```

01. `void printhello()` ✓
`void printhello(String s)`

02. `void printhello(String s)` ✓
`void printhello(Integer i)`

03. `void printhello(String s)` ✗ → Compile time error if written in same class
`int printhello(String s)`

Datatype & Name of method is same but diff parameters

Method Overriding

```
class A {
    void dosomething (String a) {
        ==
    }
}
```

```
class B extends A {
    String dosomething (String b) {
        ==
    }
}
```

* Is this allowed ? NO

Why ? - Because child class has everything that is present inside the parent class.

- This is why two same fnction with two diff return type is not allowed

```
class A {
    void dosomething (String a) {
        | =
        3
    3
}
```

```
class B extends A {
    void dosomething (String b) {
        | =
        3
    2
}
```

```
A a = new B();
```

```
a.dosomething("a");
```

Method overriding / run time polymorphism.