# OOPs 2 : Constructor, Inheritance & Polymorphism

## # AGENDA

- → Constructor
- → Types of Constructor
- → Deep Copy & Shallow Copy
- → Inheritance
- → Polymorphism
- → Method Overloading & Method Overriding.

## # CONSTRUCTORS

→ CLASS :- Blueprint of an entity

→ OBJECTS :- Instance of class.

Ex:-

```
class Student {                    <- CLASS
    String name;
    int age;
    double PSP;
}

Student St = new Student();
             ^
           OBJECT
```

⇒ Comparision b/w Simple declaration & Object Creation.

```
int a = 12;
```

Here Student is a data type

```
Student St = new Student();
```

Student() → constructor

QQ → Can you see any method named Student in class?

↳ No ! Right

↳ Symbolic constructor known Compile will Create for us & it is as Default Constructor

## (1) Default Constructor

↳ If we don't create our own Constructor in class, then a default constructor is created.

↳ Default Constructor creates a new object of the class & set the value of each attribute as the default value of that type.

Ex → null for String.
0 for int
0.0 for double.

Note :- Default constructor initializ value if not done by Creator.

Eg :-

```
class Student {
    String name;
    int age;
    double PSP;
    String uniName;
```

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

Hidden

**Observation** ⇒ we dont pass any value while Object creation.

**Q** In what condition a default constructor is not created?

↳ when we have created our own Constructor

**Q** Anything special About the name of Constructor?

↳ Same as class name.

**Q** what really is data type of the Constructor i.e. what data type does the ~~Comparator~~ Constructor retiern?

↳ class Name only.

**SUMMARY**

① Takes no parameter.

② Sets every attribute of class to its 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.

## ② Manual Constructor

Eg:-

```
Public class Student {

    String name;
    Private int age = 21;
    String univName;
    double PSP;

    Public Student ( String StudentName,
                     String universityName ) {

        name = StudentName;
        univName = universityName;
    }
}
```

Manual Constructor →

// Object Creation

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

QQ :- will it work or going to throw error?

↳ Error

// Correct way.

```
Student st = new Student ( "Chamandeep",
                           "IIT");
```

Q How Manual / Custom Constructors Works.

↳ It initialize the object with default value

↳ It will execute the Statements inside constructor.

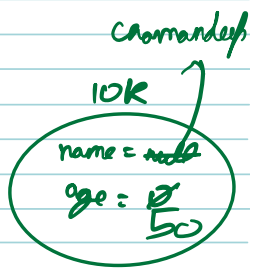
## ③ Copy Constructor

↳ We use this when we want to create object with exact same values as that of older object.

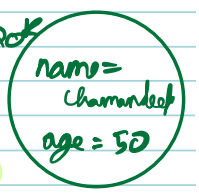
Eg:-

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

    Student ( ) {
        name = null
        age = 0
    }

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

Copy Constructor




```
Student st1 = new Student ( );
st1.name = "Chamandeep";
st1.age = 50 ;
Student st2 = new Student ( st1);
```




⇒ Use Case of COPY CONSTRUCTOR ?

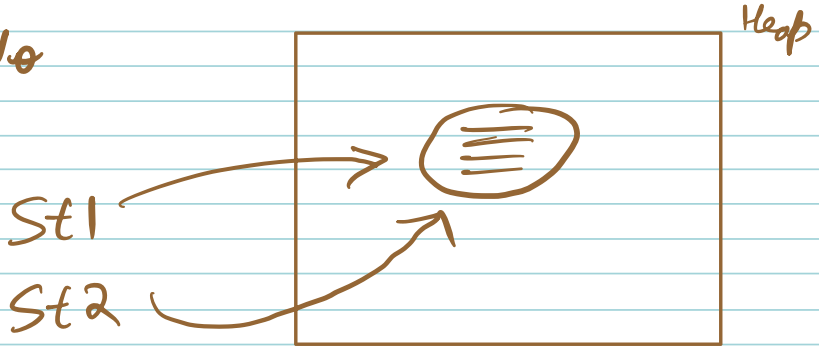
↳ when newly created object need same value. So instead of doing manually copy constructor could be used.

↳ when some attributes are private, their copy constructor helps to copy such attributes.

Quiz 1 :- Is copy constructor same as doing

Student St2 = St1;

↳ No




# SHALLOW COPY

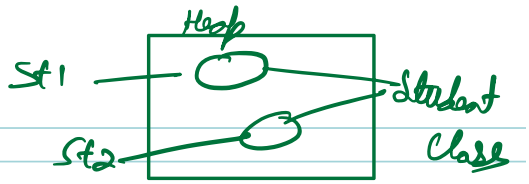
↳ when we create new object. But behind the scene the new object still refers to attributes of old object.

↳ Quiz 1 is Example of shallow Copy

Eg:- Student St1 = new Student();

Student St2 = St1;

Shallow Copy




# # DEEP COPY

↳ when we create new object. And behind the scene the new object do not refers to attributes of old object.

Eg:-

```
Student St1 = new Student();
```

Deep → `Student St2 = new Student();`

# # INHERITANCE

Q → what is the idea behind OOPs taking over procedural programming.

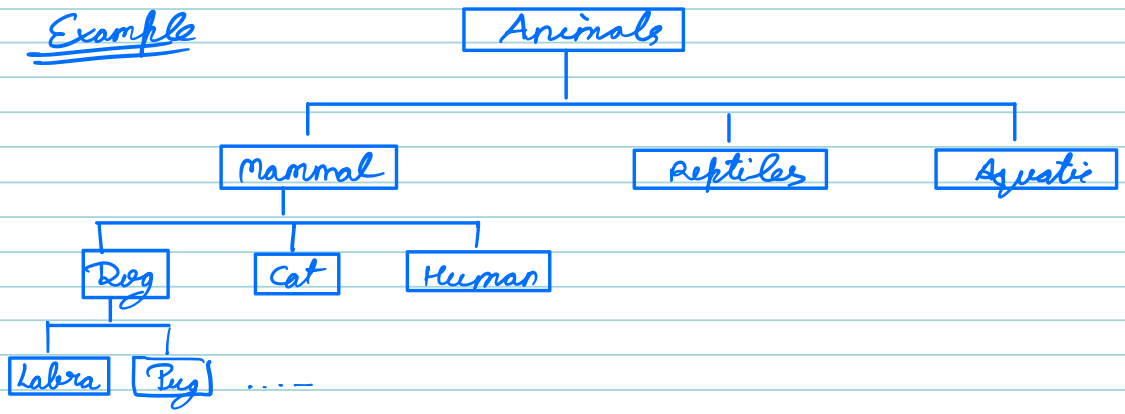
↳ Its more oriented toward how human think.

Q → Do we have any relation / hierarchy with animals or things around us?

↳ yes

⇒ so OOPs helps us in forming Hierarchy.

Example



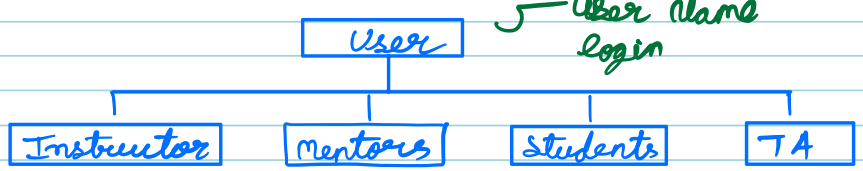

Q if Animal can Walk, does that Mean a Labra can walk?

↳ yes, Because

Labra → Dog → Mammal → Animals.

★ This States hierarchy allow us to share behaviours & attributes with Specific categories.

★ Representation of hierarchies in classes is known as Inheritance.




Q Will Instructors, Mentors, Students & TA all have two mentioned properties?

↳ yes - But each one of them can still have unique properties

Eg:- Student have PS P but teacher do not.

⇒ Parent – child Relationship in Code

Eg:-

```
class User {
    String userName;
    void login() {
    }
}
```

// Instructor inheriting User

```
class Instructor extends User {}
```

↓ used to create child class

⇒ In different languages Inheritance is done using different keywords.

Ex:- ① In Python

```
class subclass(superclass)
```

② In C++

```
class subclass : public superclass()
```

③ In C#

```
class subclass : BaseClass()
```

=> completing above code

child ↙ Parent ↙

```
class Instructor extends User {

    String batchName;
    Double avgRating;

    void scheduleClass() {
    }
}
```

Q → Does the Instructor class needs a username property ?

↳ yes

Q Do we need to code it ?

↳ No

**CONCLUSION**

↳ Extends means, keeping the original things & adding more to it.

## # CONSTRUCTOR CHAINING

```
Instructor i = new Instructor();
```

Q Did we created Instructor constructor ?

↳ No, Because of Default constructor.

Q Can we Do like :-

```
i.userName = "chamandeep";
i.login();
```

↳ yes, Instructor inherits this from User Class

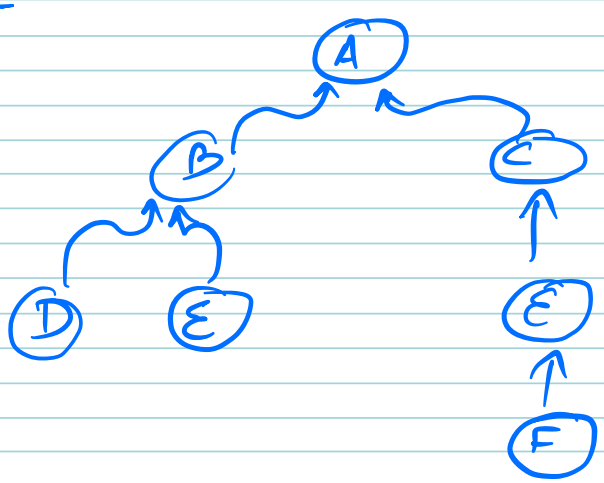
Q Can we Do this :-

i. avg Rating = 4

↳ yes

Note :- In Inheritance, a parent class is nothing but generalization & every child is a specification.

Remember




Quiz 2 which of these classes has the highest level of Abstraction ?

Abstraction ⟶ Idea

Bigger Idea more general

Hence A out of all will be having most generic Idea. So we can say A will be having highest level of Abstraction.

Quiz 3 Do the child class contains all the attributes of parent class ?

↳ No, because private attribute will not be

inherited.

Q How are they initialized? who initializes them?

```
class User {
    String name
    public User() {
        this.name = null;
    }
}
```

```
class Instructor extends User {
    String batch;
    Public Instructor() {
        Name = null;
        batch = null;
    }
}
```

Instructor i = new Instructor()

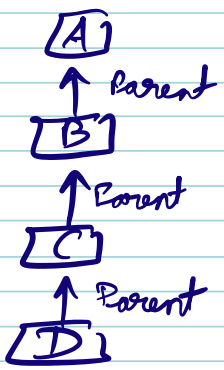
Q Do you think We always be able to know parent class attributes

↳ No,

## ⇒ Steps to Create Object of child

Assumption :- only default Constructor is here

Ex :-




Q what will happen when we write?

```
D d = new D();
```

⇒ Constructor D → Invoke → Constructor C → Invoke → Constructor B → Invoke ↓ Constructor A

## Constructor Execution Flow

A → B → C → D

Q → what if Parameterized constructors are there.

Added Manual Constructor in C :-

Eg:-

```
class C extends B {
    C() { }
    C(String a) { Print(Hey) }
}
```

And

```
D d = new D();
```

↳ Nothing will change & constructors will be executed in above manner.

Q How to invoke manual constructor of C?

Ans Using SUPER Keyword

Eg:-

```
Public class D extends C {
    D() {
        super("Hello")
    }
}
```

```
D d = new D();
```

① This will invoke parameterized constructor of C

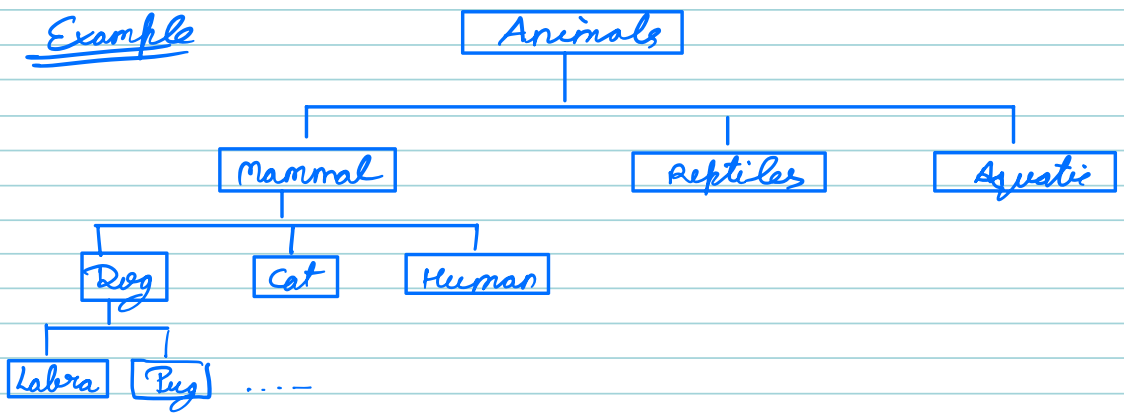
② It should be the first line of constructor

# POLYMORPHISM

Many Forms

→ It means Someone who have Many forms

Example




Q Can we write ?

```
Animal a = new Dog();
```

↳ yes, because it will create a object of Dog, which is a mammal, & Mammal is a Animal. So we can write it.

Q Can we write.

```
Dog d = new Animal();
```

No, we can't write because Animal's object is created & all Animals is not Dog.

CONCLUSION :- We can put an object of the child class in a variable that takes parent data type.

Eg:-

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

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

```
C extends A {
    String company;
}
```

```
A a = new C();
a.company = "ABC";
```

ERROR :- This will throw an error because a has a datatype of A, but A doesn't have any variable named company.

↳ Compiler only allows you access to members of the Data type of the variable

↳ Polymorphism help to make code generic which improves readability.

# Two types of Polymorphism

- Compile Time polymorphism
- Run Time polymorphism.

# METHOD OVERLOADING

Q :- Can a class have another method with same name but different parameters.?

↳ yes

Eg:-

```
class A {
    void hello() {
        sout("Hello World");
    }
    void hello(String name) {
        sout("Hello" + name);
    }
}
```

↳ This is known as method overloading

Q Here also, can you identify Polymorphism ?

↳ yes, same method in many forms.

Q Does the compiler know which method to invoke in below statements

→ hello();
→ hello("chamandeep");

↳ yes, using the parameters

Since the final form that will execute is known to compiler. That is why it is known as Compile Time polymorphism

Q which of the following is a Method overloading?

(A) void printHello();
void printHello(String s); ✓

(B) void printHello(String s);
void printHello(Integer s); ✓

(C) void printHello(String s);
String printHello(String s); ✗

↓ Method Signature

↳ Methods are known to be Overloaded when they same but different Signatures.

Qeej 4 :- Is this allowed in the same class?

void printHello(String s) {.... }

String printHello(String s) {.... }

↳ NO

# # METHOD OVERRIDING

Eg:-

```
class A {
    void doSomething (String A) {
        ..... .
    }
}

class B extends A {
    String doSomething (String) {
        .....
    }
}
```

Q Is this allowed ?

↳ No, Because to compiler Class B will look like.

Eg:

```
class B extends A {
    void doSomething (String A) {
        ..... .
    }

    String doSomething (String) {
        .....
    }
}
```

Since we can't have two Method with same signature. Hence not allowed.

Eg:-

```
class A {
    void doSomething (String A) {
        ..... .
    }
}

class B extends A {
    void doSomething (String) {
        .....
    }
}
```

Q what you think is it allowed?

↳ yes, If parent & child classes have

- → Same Method with same name.
- → Same return type
- → Same Signature.

then it is Method overriding

Eg

```
class A {
    void doSomething() {
        print(Hello);
    }
}
```

```
class B extends A {
    void doSomething() {
        print(Bye);
    }
}
```

```
main() {
    A a = new A()
    a.doSomething();   // hello
    a = new B();
    a.doSomething();   // bye
}
```

⇒ Points to remember

(1) The method that is executed is of the data type that is actually present at the time of code & not the type of variable.

(2) Do we know the exact code that is about to run in compile time?

↳ No, & that's why its called Runtime polymorphism

CONCLUSION

↳ Compiler relies on the data type of variables, whereas runtime relies on actual Object.