Constructors

Types of Constructor

Deep Copy & Shallow Copy

Inheritance

Polymorphism

Method Overloading & Method Overriding

Class : Blueprint of an entity

Object : Real life entity / class instance

class

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

→ Student st = new Student()

calling constructor

int a = 12

a
12
4B

st
reference
store address
of actual obj

st
10 K

Heap

10 K

name : null
age : 0
psp : 0.0

→ Student obj

st.name

Student() → calling a constructor

↓

initialize attributes of new object created

No constructor present in class def. calls the default constructor

Sets value of each attribute as default value of that type

```
class Student {
    String name
    int age
    double psp
    String uni
}
```

Default constructor → no arguments

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

Compiler does not always provide default constructor. Why?

Whenever user provides any constructor,

compiler will not provide.

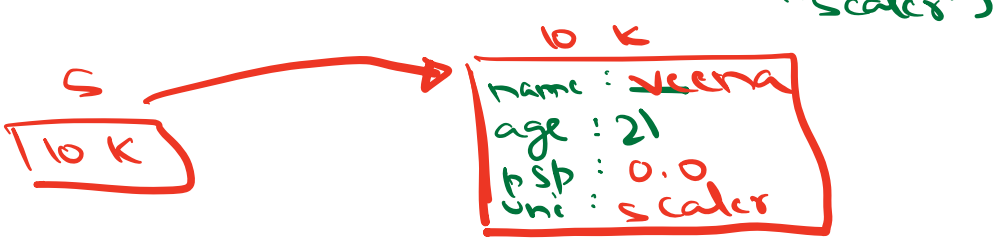
Summarising the default constructor:

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 / Parametrized constructor

```
class Student {
    String name
    int age = 21
    double psp
    String uni

    public Student(String sname, String univ) {
        name = sname
        uni = univ
    }
}
```

```
Student s = new Student("Veena", "Scaler")
```



Student S2 = new Student() → //Error

No default constructor supplied by compiler

Resolve error by passing 2 arguments

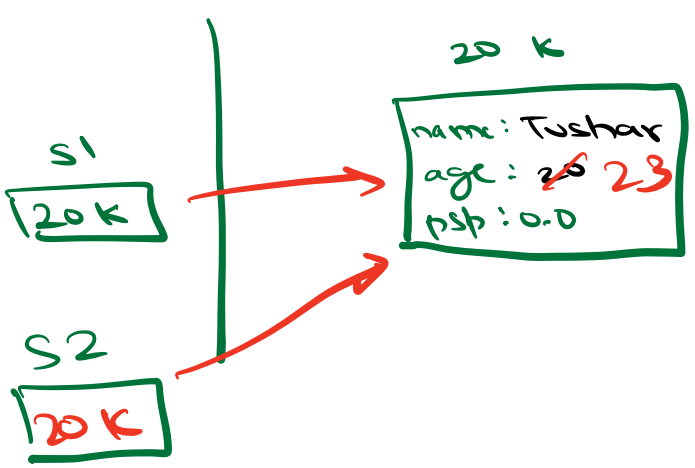
Summary of manual constructor

1. Starts executing code inside constructor
2. It initializes object with values passed for attributes

Copy constructor

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

Student S1 = new Student()



S1. name = "Tushar"

S1. age = 20

Student s2 = s1 // Incorrect

s2.age = 23

Shallow copy

Only a new reference s2 is created, s2 starts pointing to same memory location as s1

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

    Student() {
        name = null;
        age = 0;
    }
    Student(Student st) {
        name = st.name;
        age = st.age;
    }
}
```

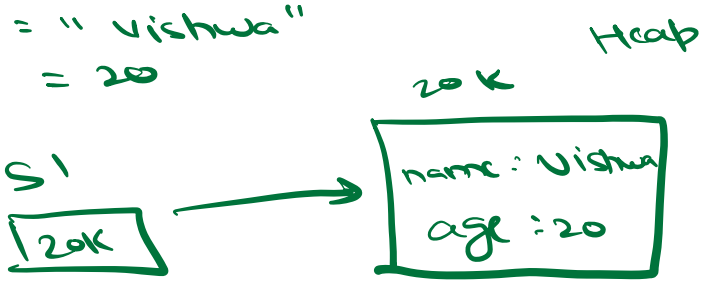
Default constructor

Copy constructor

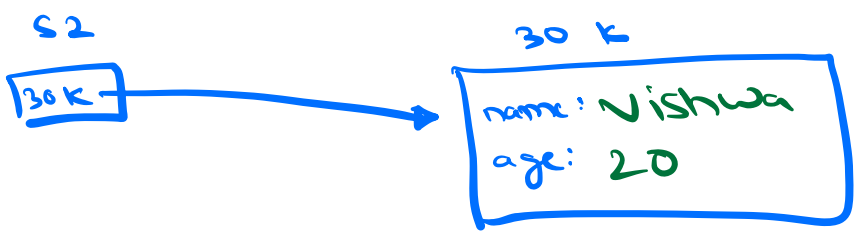
Student s1 = new Student()

s1.name = "Vishwa"

s1.age = 20



Student s2 = new student (s1)



Deep copy

↓

2 diff objects with same value of attributes

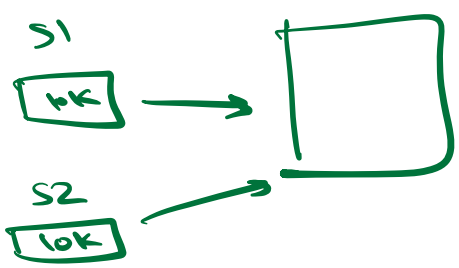
s2.name = "Apurva"

print (s1.name) ⟶ Vishwa

Is copy constructor same as student s2 = s1

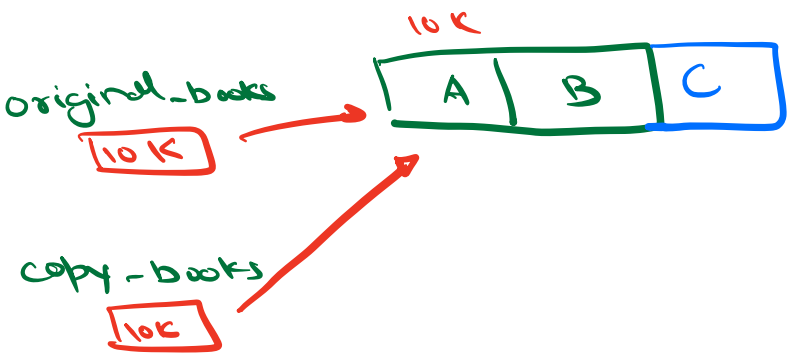
No

student s2 = s1 (shallow copy)



Shallow Copy

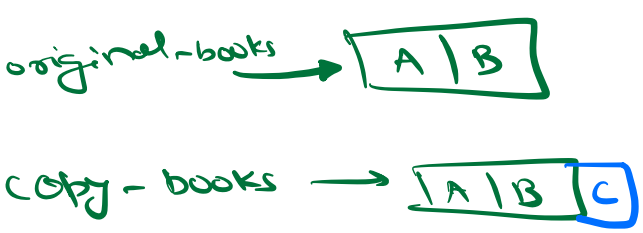
```
original_books = ["A", "B"]
copy_books = original_books
```




```
copy_books.append("C")
print (original_books)   → A, B, C
```

Deep Copy

```
import copy
original_books = ["A", "B"]

copy_books = copy.deepcopy(original_books)
copy_books.append("C")
```



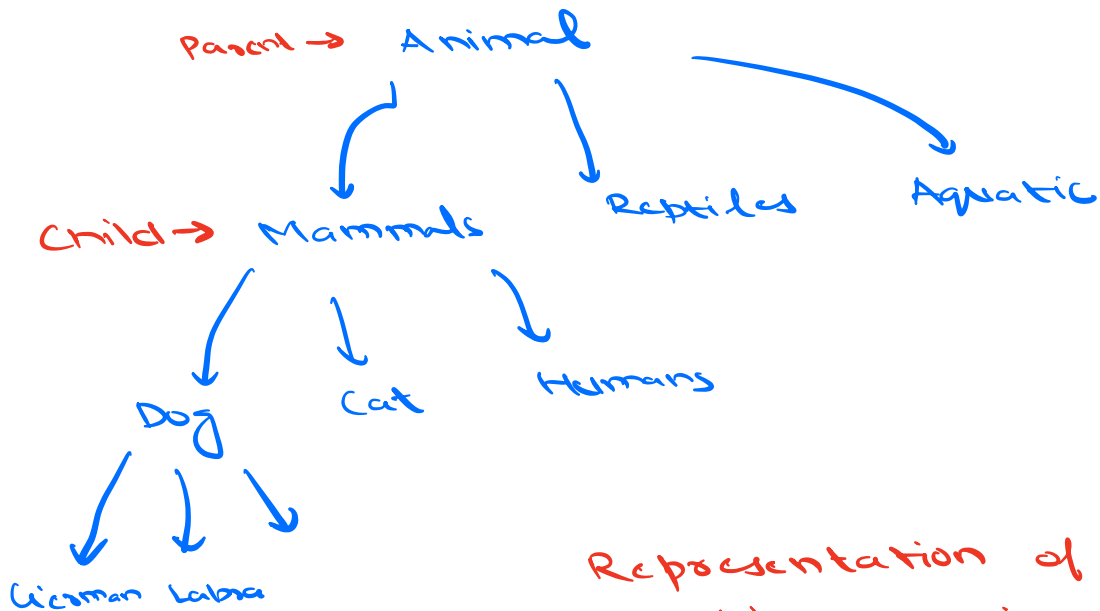

```
print (original_books) → A, B
```

Inheritance

What was idea behind OOPs ?

Entities perform action

Someone is doing something



Representation of this hierarchy is inheritance

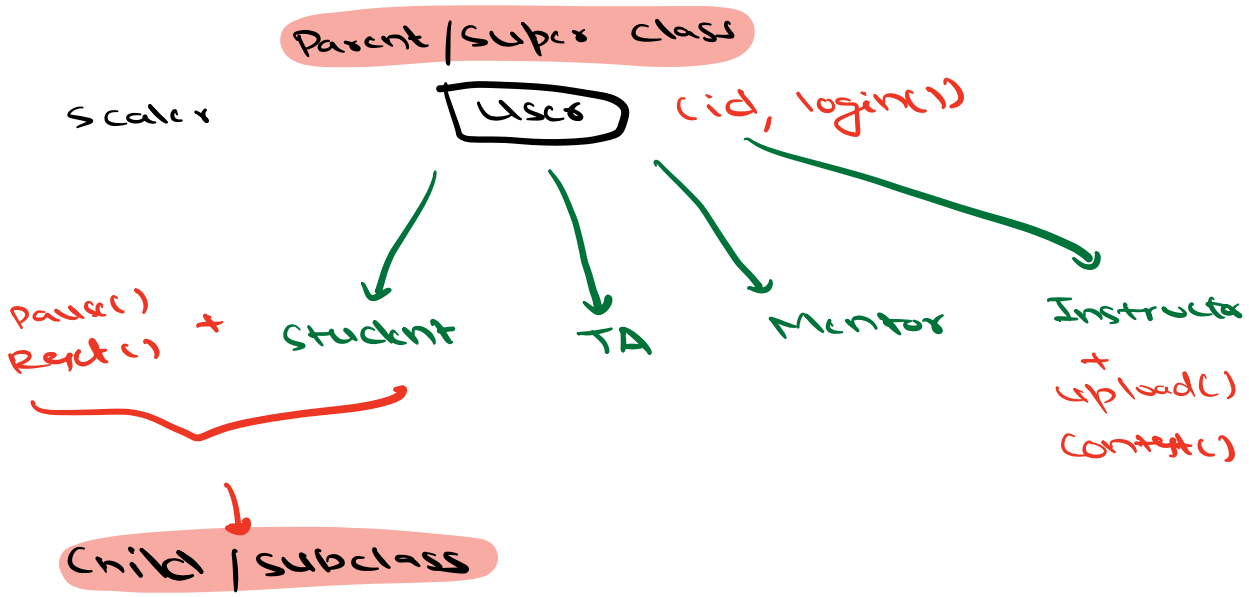
↓

Parent - child relationship b/w diff classes

Parent → x

↓

Child x + y



They share all members / attributes / methods of User class and may have some more of their own

User → Instructor

Parent — Child

```
class User {
    String id
    void login() {
    }
}
```

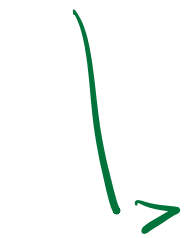
Java

```
class Instructor extends User {

}
```

Python class Instructor (User)

C++ class Instructor : public User < >

C# class Instructor : User <



---

class User <

    string id

    void login() <

    } ...

}

class Instructor extends User <

    string batchname

    double rating

    void schedule class() <

    } ...

}

User (Id, login())

Instructor (batchname, rating, schedule class)

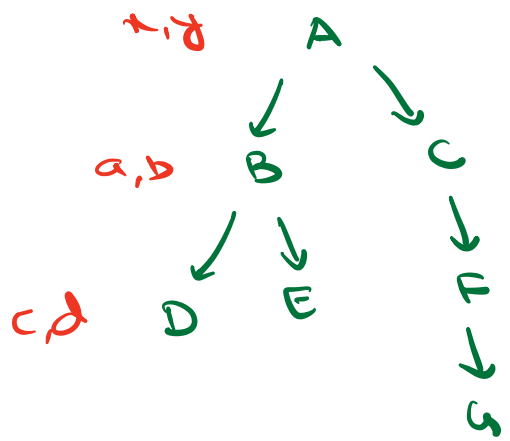
Extend → keeping original things and adding more things to it

Instructor i = new Instructor ()

batchname : null
rating : 0.0
id : null

i. batchname = "B7"
i. id = "abag"
i. login()

Parent class → Generalization
Child class → Specification

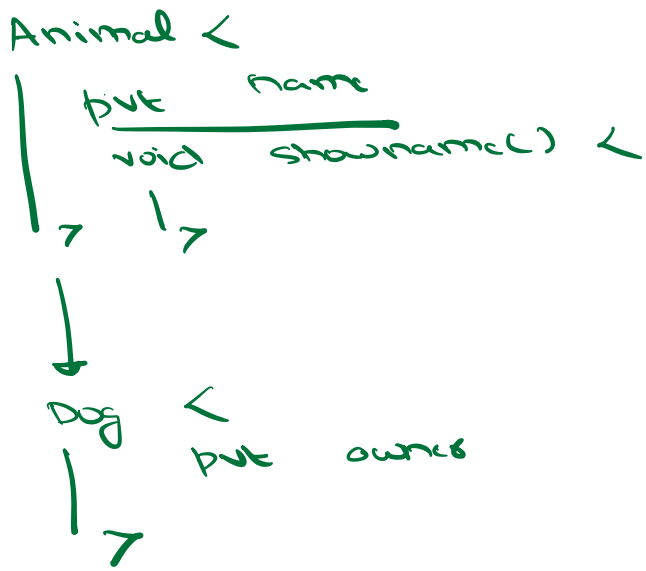


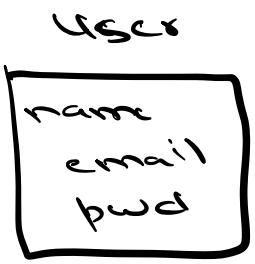
Which class has highest level of abstraction?

general idea

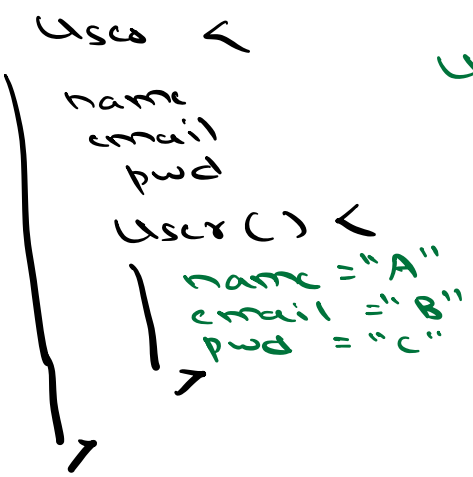
A

Animal → Dog





User u = new User()



User u = new User()

```
Instructor extends User {
    string batch
}
```

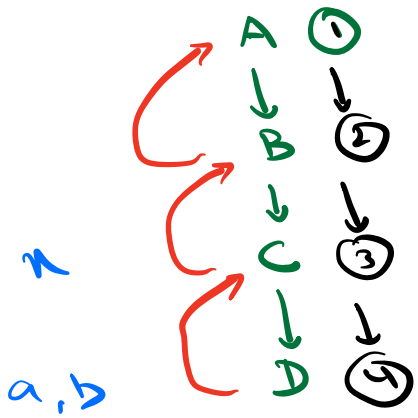
When we create instructor object, someone has to initialize attributes that came from the parent

①

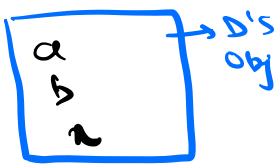
```
Instructor extends User {
    string batch
    Instructor() {
        batch = "B7"
        name = 'A'
        email = "B"
        pwd = "C"
    }
}
```

② In child's constructor, call parent's constructor

10:52



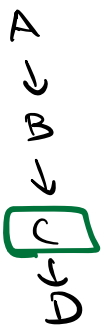
D d = new D()



1. Constructor of D will be called
2. D is a child of someone, so before its execution, constructor of C is called
3. C will call B's constructor
4. B's constructor calls A's constructor before its execution

When we create object of child class, before constructing itself, it will call default constructor of parent.

```java
public class C extends B {
    C() {
        System.out.println("Constructor of C");
    }
    C(String a) {
        System.out.println("Constructor of C with params");
    }
}
```



D d = new D() Constructor of C

public class D extends C {

D() {
super("Chaitra")
print(" Constructor of D")
}

D d = new D()

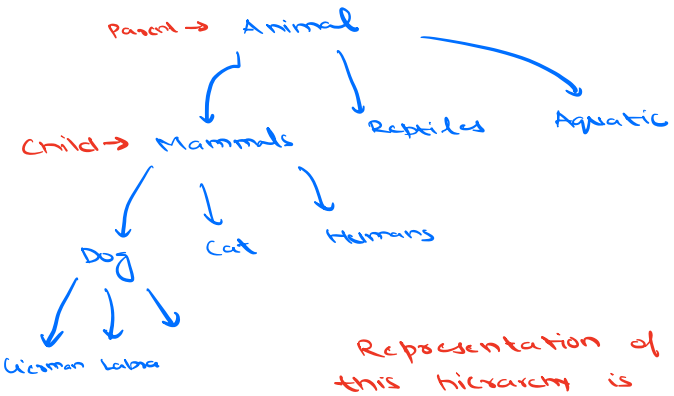
Constructa of C with params
Constructor of D

class A {
int id
A(int num
id = num

B extend A {

Poly morphism

Many forms



Representation of this hierarchy is

```
Animal a = new Dog();   ✓
```

all dogs are animal

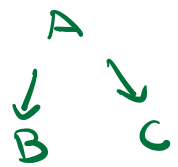
```
Dog d = new Animal();   X  // Error
```

all animals are dog

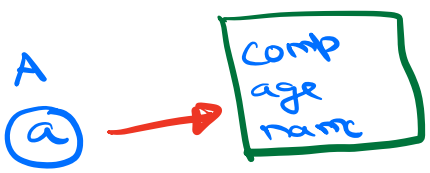
We can put obj of child class in reference variable of parent class and not vice versa

```
class A {
    int age
    String name
}

class B extends A {
    String univ
}

class C extends A {
    String comp
}
```

A a = new C() ✓



a.age
a.name



a.comp // throw compile time error

A a

↓

Compiler only allows you to access members of data type of variable

data type of 'a' → A

2 types of polymorphism

↓ Compile time

↓ Run time

Method Overloading → Compile time polymorphism

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

A → a method named hello()

```
A a = new A()
a.hello()          → "Hello"
a.hello("Vena")    → "Hello Vena"
```

① ✓ No. of args

```
void printHello()
void printHello(String s)
```

② ✓

```
void printHello(String s)
void printHello(Integer s)
```

printHello("ab")
printHello(3)
→ Type of args

③

```
void printHello(String s)
String printHello(String s)
```

// compile time error

↓ are these 2 same? NO

↓ no. and type of arg are exactly same

Method Overloading

↓

Method signature should be diff

void print (String n, int age)

Methods are overloaded when they've same name but diff signature → no. of args

→ type of args

Method Overriding

```
Class A {
    void doSomething (String A) {
        |_ ...
    }
}

Class B extends A {
    String doSomething (String C) {
        |_ ...
    }
    void doSomething (String A) {
        |_
    }
}
```

int a = 30

a = 40

a → 40 (30)

A

↓

B

// Compile time error

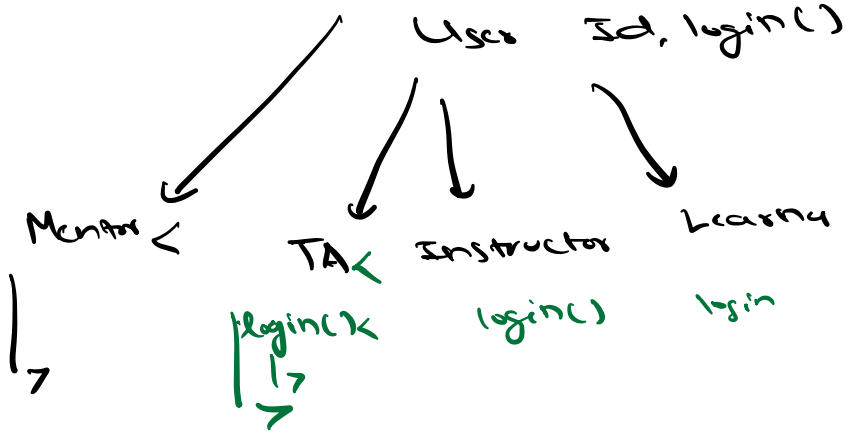
```
Class A {
    void doSomething (String A) {        — (1)
        ...
    }
}
```

a. dosomething

A
↓
B

```
Class B extends A {
    void doSomething (String C) {        — (2)
        ...
    }
}
```

Parent & child have same method with same name, same return type, same signature, is called method overriding.



```
class A {
    void fun() {
        print("Hello")
    }
}
```

```
class B extends A {
    void fun() {
        print("Bye")
    }
}
```

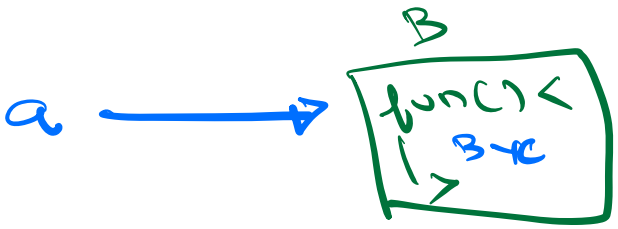
A
↓
B

```
A a = new A()
a.fun()  → Hello

B b = new B()
b.fun()  → Bye
```

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

all B's are A

```
a.fun()  → Bye
```



Doubts

7154~~7~~

Min digits to be removed to be divisible by 25

28

x

if $x \,\%\, .25 = 0$

$\Rightarrow x \,\%\, .5 = 0$

100/s

20/s

5

1s

5 10 15 20 25 30