Agenda

- Constructor
- Types of constructor
- Deep copy and shallow copy
- Inheritance
- Polymorphism
- Method Overloading and Overriding

Constructors

class → blueprint of an entity
object → an instance of a class

```
class Student {
    String name;  ┐
    int age;      ├ instance variables
    double psp;   ┘
}
```

To create an object of class Student :-

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

int x = 3;

new Student() → constructor
- → construct the object
- → initializes the instance variables
- → no return type
- → same name as the class name.

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

    Student(){                 → default constructor
        name = null;  ┐
        age = 0;      ├→ default values of those data types.
        psp = 0.0;    ┘
    }

}
```

If we don't create our own constructor in a class, then a default constructor gets automatically created.

default constructor
- → takes no parameter
- → sets every attribute to its default values
- → created only if we don't call our own constructor
- → it is public (can be accessed from anywhere)

Manual Constructor

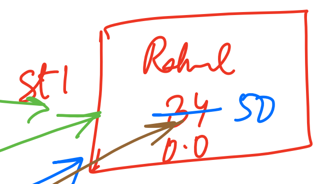
```
class Student {
        String name;
        int age;
        double psp;

        public Student( String name, int age){    } Parameterized
              this.name = name;                   } constructor.
              this.age = age;                     }
              psp = 0.0;                          }
        }

}
public class Client ( ) {
        p s v main (String args [ ]){
              Student st = new Student ("Akash", 30);

              Student x = new Student(); // Error

                    :

        }
}
```

```
name = Akash
age = 30
psp = 0.0
```

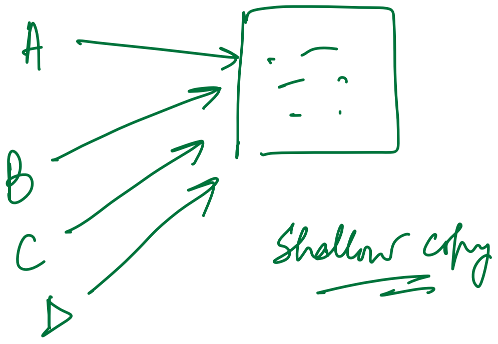
```
class Student {
    String name;
    int age;
    double psp;
    public Student() {
        this.name = null;
        this.age = 0;
        this.psp = 0.0;
    }
    public Student( Student st){
        this.name = st.name;      or    name = st.name;
        this.age = st.age;               age = st.age;
        this.psp = st.psp;              psp = st.psp;
    }
}

public class Client() {
    p s v main(String args[]){
        Student st = new Student();
        st.name = "Rahul";
        st.age = 34;
        Student st_copy = new Student(st);     [Deep copy of st]
            :
    }
}
```

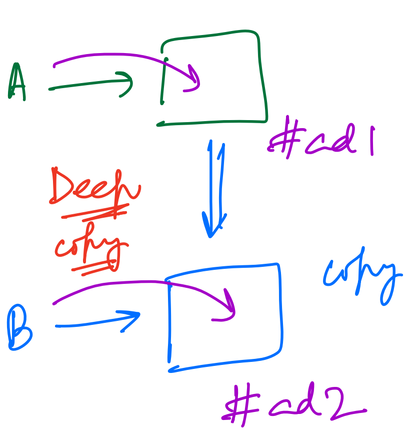
Shallow copy

```
Student st2 = st1;

st2.age = 50;
print(st1.age) ——→ 50.
```



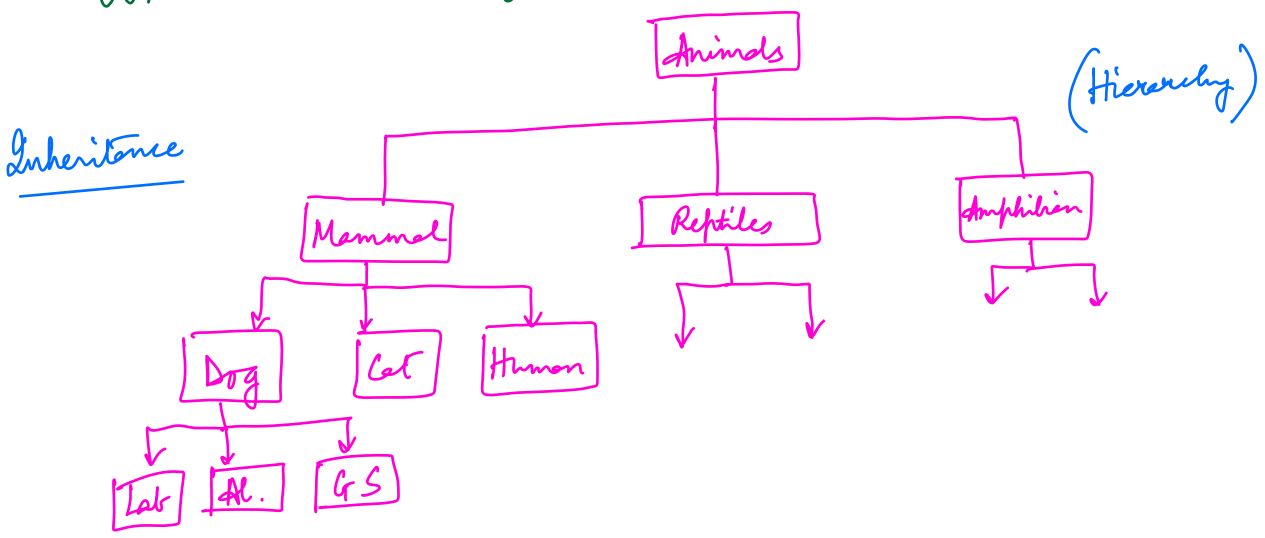
- New object is not created
- Changing one variable will impact the other(s).

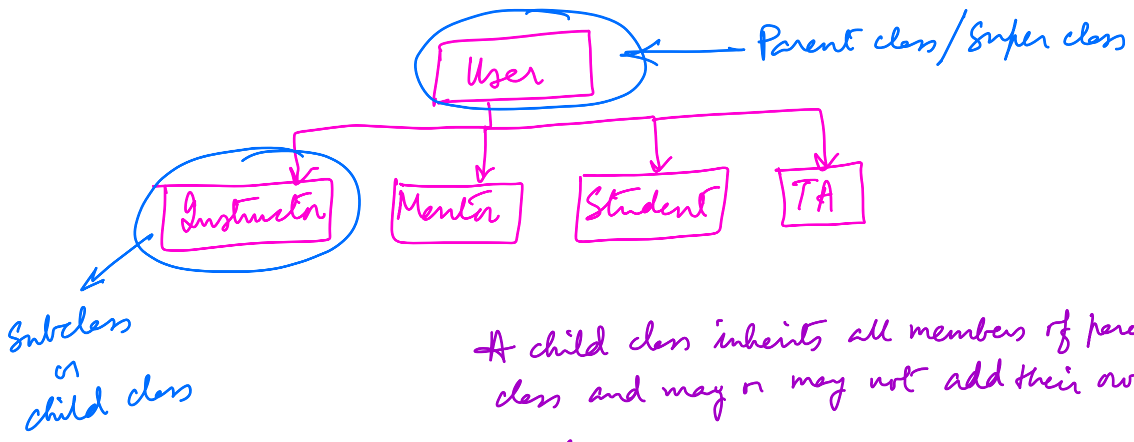

A
B
C
D
Shallow copy


A
#cd1
Deep copy
B
copy
#cd2

Inheritance

OOP $\longrightarrow$ Someone doing something





A child class inherits all members of parent class and may or may not add their own members.

[Break till 10:37 PM]

```
class User {
    String username;
    void login(){
        ...
    }
}

class Instructor extends User {
    String module;
    int num_sessions;
    void schedule_class(){
        ...
    }
}
```

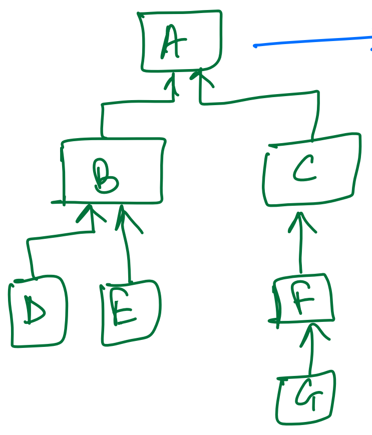
Does the Instructor class need a username property?
No, it already has from parent.

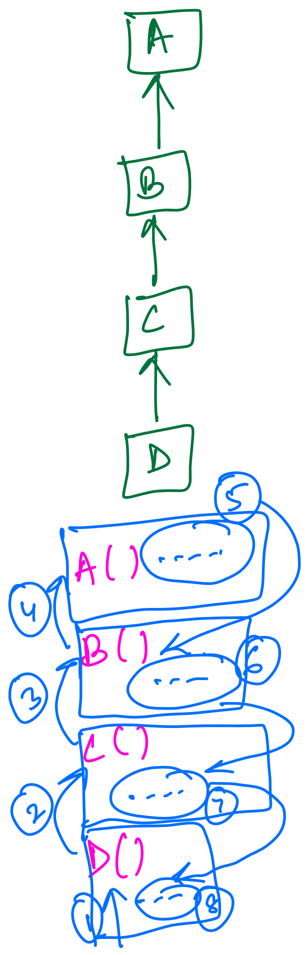
Python :- class Subclass(Superclass):
C++ :- class Subclass : public Superclass { }
C# :- class Subclass : Superclass { }

```
Instructor i = new Instructor();
i.username = "Rahul";
i.login();
i.module = "...";
```

→ Default constructor

Superclass → generalization
Sub class → specialization





D d = new D();

1. D() constructor gets called
2. Before its own execution, D() calls its parent's const. C()
3. Before its own execution, C() calls its parent consts. B()

A finishes first
↓
B finishes
↓
C finishes
↓
D finishes last

} constructors.

Polymorphism

① Inheritance-driven

Poly → many
morphism → forms

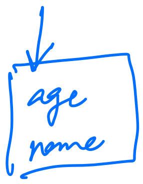
User class → can be Student / Instructor / TA etc.

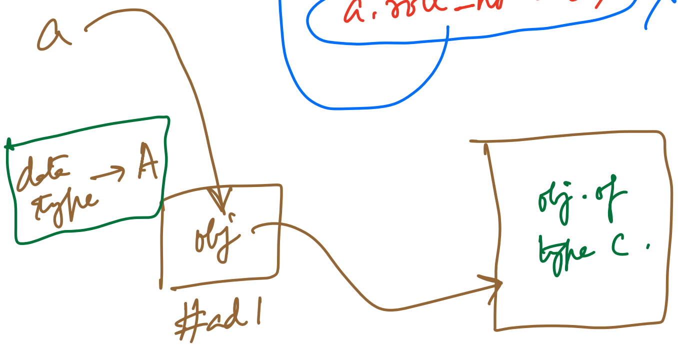
```
User x = new Instructor(); ✓✓

Instructor y = new User(); ✗
```
Not every user is an instructor.

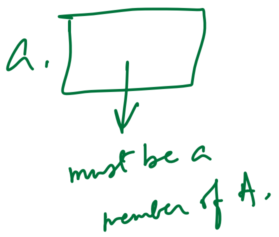
```
A {
    int age
    String name
}

B extends A {
    String address;
}

C extends A {
    int roll_no;
}
```

```
A a = new C();
a.roll_no = 5; ✗
```
Compiler throws error.

a → a variable of type A.







User x = new Instructor()
↓
compiler treats this as User type.

② Method overloading

```
void hello( ){
        Sopln("Hello World");
}
void hello(String name){
        Sopln("Hello "+ name);
}
```

Same name, diff. param.
[Overloaded methods]

hello() ⟶ calls the 1st fn. ⟶ Hello World
hello("Saptarshi") ⟶ calls the 2nd fn ⟶ Hello Saptarshi

```
void hello(String name){
        Sopln("Hello "+ name);
}
int hello(String name){
        Sopln("Hello "+ name);
        return 1;
}
```

✗ compiler throws error

... hello("Harish");

compiler looks at client code and decides which of the overloaded functions need to be called.

⟶ looks at $\boxed{\text{methodName(<Parameter list>)}}$

↓

Method signature.

③ Method overriding

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

Is this allowed?

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

Compile time error

```
class A {
    void doSomething(String a){
        Sopln("Hello");
    }
}
class B extends A {
        void doSomething(String c){
            Sopln("Bye");
        }
}
```

→ same signature

→ same return type.

Method Overriding.

→ Parent class methods get hidden.

different content for same fn. call.

```
A a = new A();
a.doSomething(); → "Hello"
a = new B();
a.doSomething(); → "Bye".
```