4 pillars of OOP

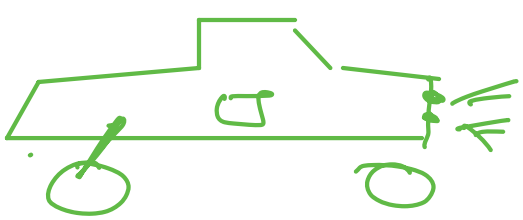
1. Encapsulation: Hiding Data
2. Polymorphism: One entity multiple behaviors
3. Abstraction: Hiding complexities of System
4. Inheritance

Abstraction

```
class BA
==
deg withdraw
==
```

1. Focus on What? instead of How?
2. Inheritance + Abstract Class

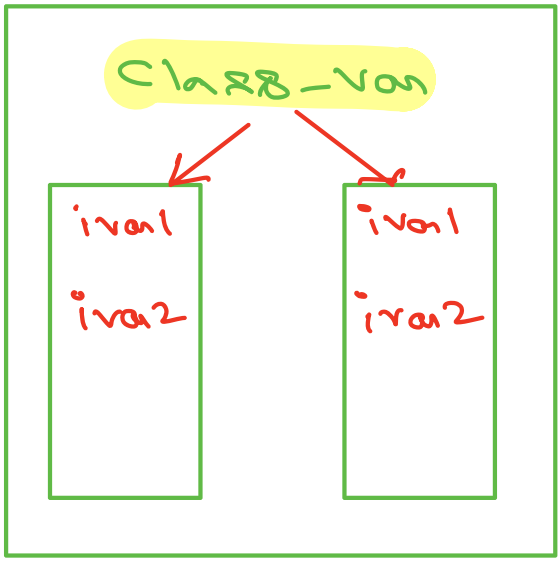
```
bal = BA()
balwithdraw(10)
```



"Hiding all the complexity of the system"

* Instance Variable : Variable that belongs to an instance

* Class Variable : A variable define in the class common to all the instances



Class_var can be accessed via instance as well as Class itself.

Modification of Class Var

① When modifying class_var using instance, it ends up creating a new instance Variable.
Class_var stay as it is.

② Class_var can be modified using only class itself.

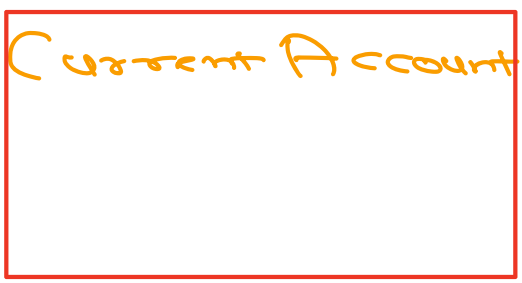
Inheritance

- Saving Account
- Current Account
- Demat Account

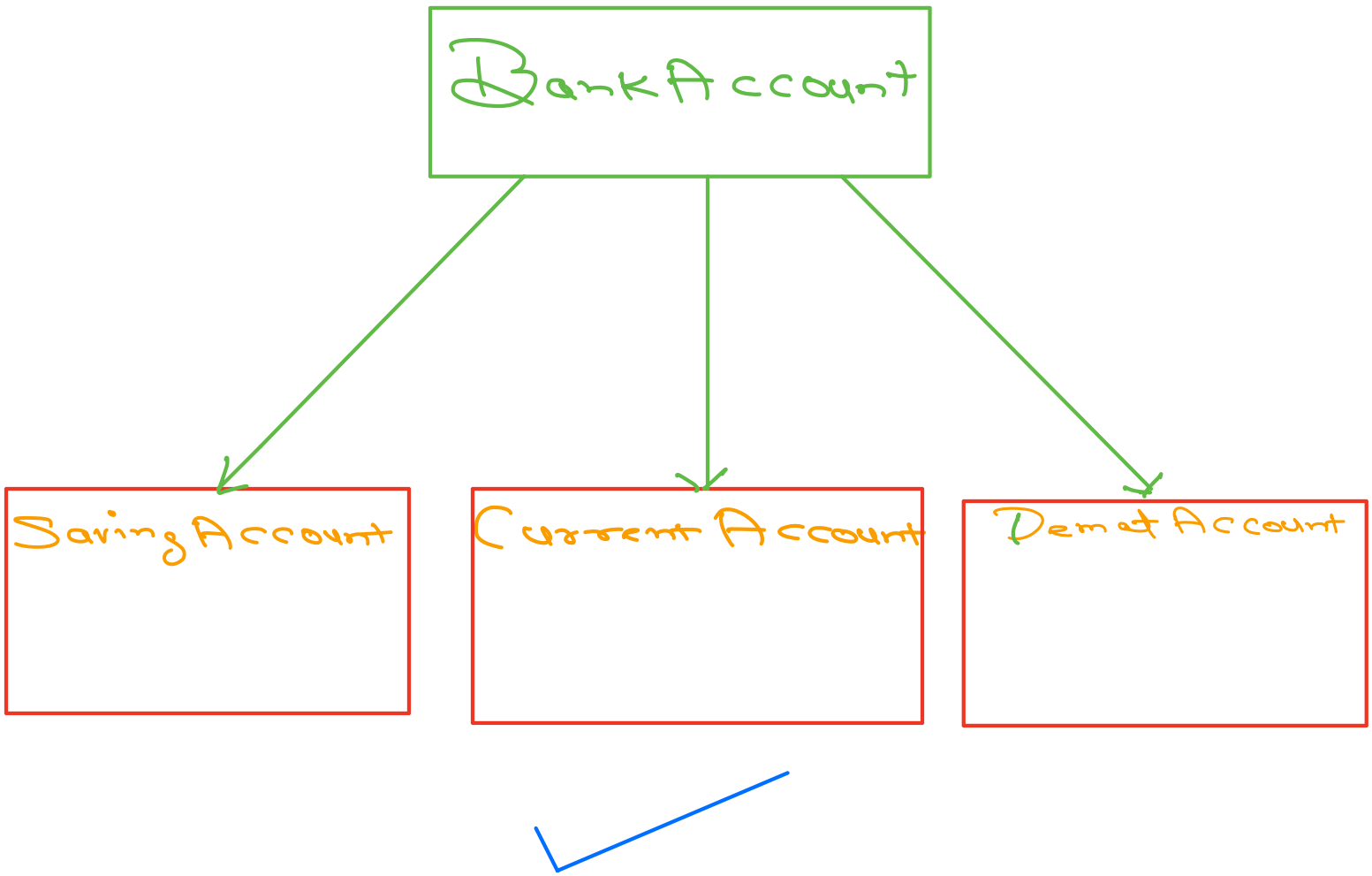
idea1 : Put all code into a single class

X

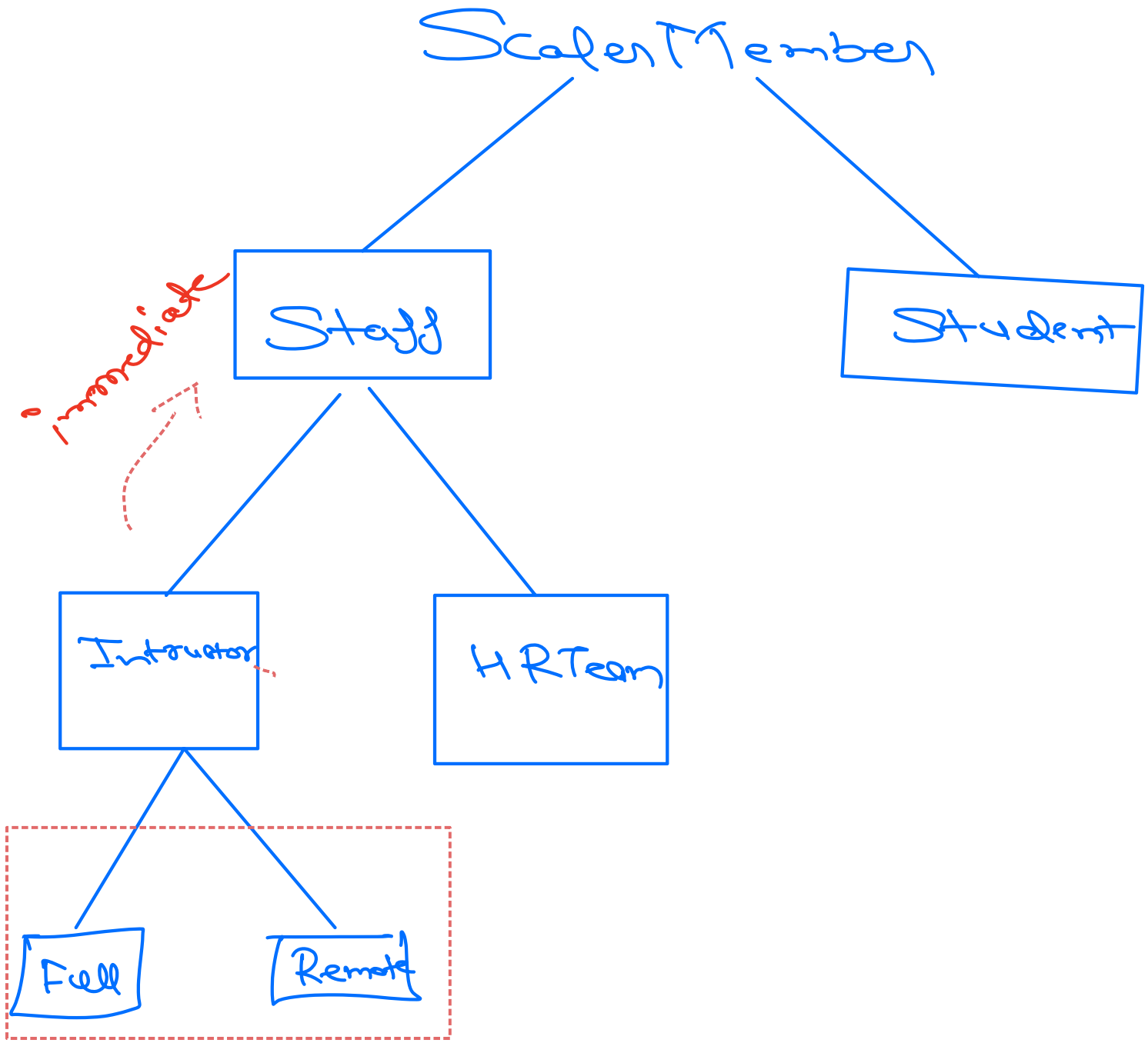
idea2: Create a seperate class for each type



* idea 3: Keep common data and methods into a parent class and create child classes which will contain their unique data/methods while inheriting the common properties and Methods



Object
(automatically inherited)
by all the classes



f1 = FullTimeInstructor()

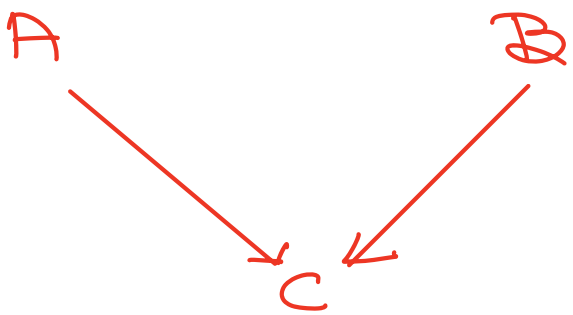
f1.method1()

Full → Instructor → Staff → Scaler

↓

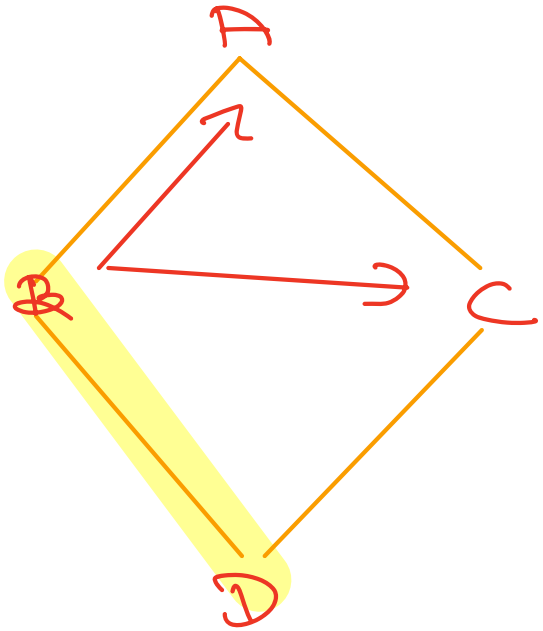
Object

Ex Multiple Inheritance



MRO ⇒ C(A,B)

Left to Right ⇒ C → A → B → Object

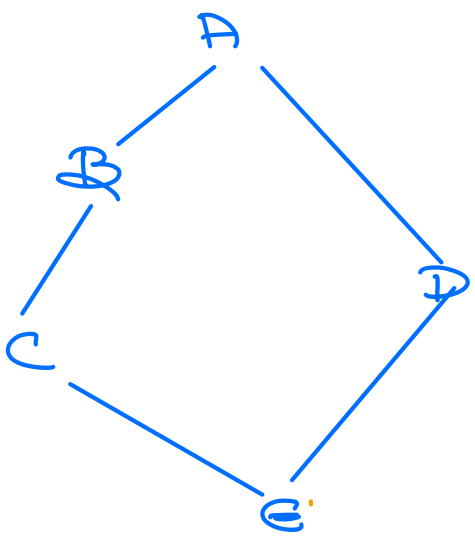


D → B → ~~A~~
C → A

MRO(D) ⇒ D B C A

① Left to Right

② Only goto a Parent when all the child have been visited



MRO(E) ⇒ E → C → B → ~~A~~ D → A

```
class A(Abstract Class)
    @abc.Method
    def Func1( )
        Pass
```

```
class B(A):
    def Func2( )
        ____
        ____
```

$B_1 = B()$ ✗ Until all abstract Methods are Overridden

To create instance B you will have to Override all Abstract Methods