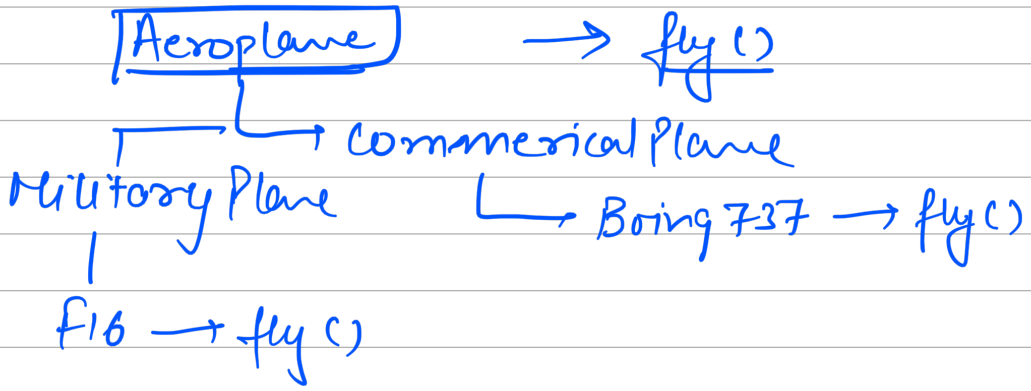
Agenda

Start @ 9.05

① Interfaces
② Abstract Classes

What is Class

↳ Blue print of entity
↳ Attributes and behaviors
↳ Definition ⇒ Implementation



⇒ A Concept that is Not a Real Entity
(Not really having attributes & def^{n} of behavior)

Such entities are called Interfaces

eg Animal ⇒ Anyone in the world who can walk, eat, run is an Animal

Interface: Blueprint of Behaviors

⇒ Declare a set of behaviors/Methods that must be implemented by a class in order to belong to the category of interface

Ex If Cat wants to be animal

```
Interface Animal{
        void run();   ⇒ Abstract Functions or Method { Method Signature only (No Body) }
        void walk();
        void eat();
        void hunt(String Prey);
}
```

```
Class Cat implements Animal {
{
    Void eat(){
        Sout(Cat is eating);
    }
    Void walk() {
        Sout (Cat is walking);
    }
    Void Run(){
        Sout(Cat is running);
    }
    Void Meow(){
        Sout ( Meow );
    }
}
```

→ Compiler is forcing to imp. the Methods

```
Dog implements Animal {
    Void walk() {
        Sout( Dog is walking);
    }
    Void Run() {
        sout (Dog is running );
    }
    Void eat() {
        sout (Dog is eating);
    }
    Void Bow (){
        Sout ( Bow Bow);
    }
}
```

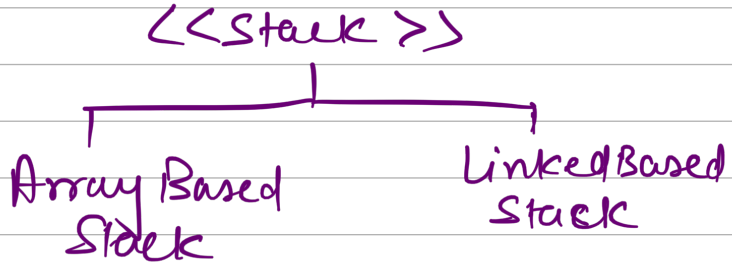
eg

Stack → Array Based Stack
→ Linked List Based stack

Stack → Push()
→ Pop()
→ size()
→ IsEmpty()

Stack<> s = new ArrayBasedStack<>();
LinkedListBasedStack<>();

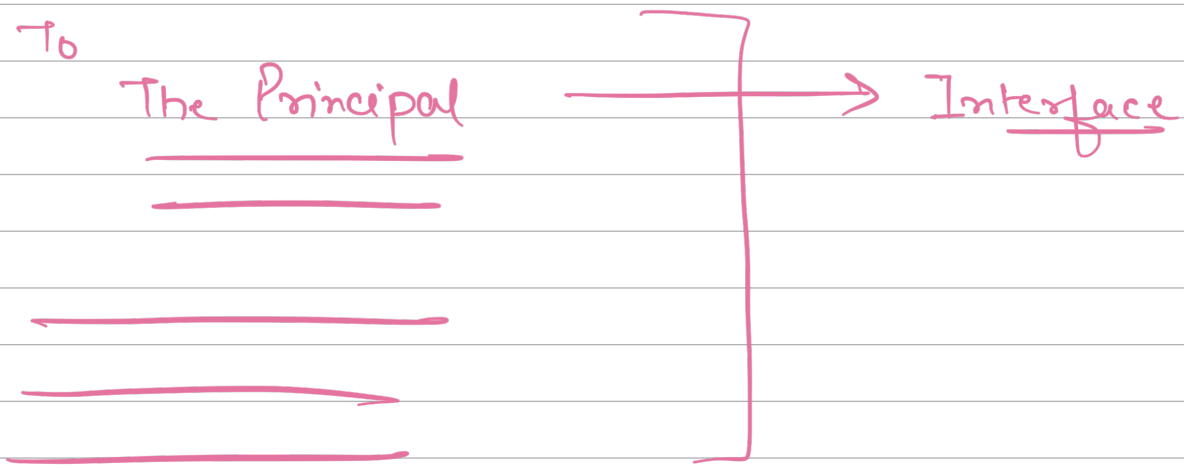
```
Class ArrayStack implements Stack
{
        Bool isEmpty()
        {
          ==========
        }
        Void push(){
          ==========
        }
}

Class LinkedListStack implement Stack
{
        BoolIsEmpty()
        {
          ==========
        }
        Void Push()
        {
          ==========
        }
}
```




```
Client {
        Stack s = new ArrayBasedStack();
                  LinkedListStack();
```

N is large { s.push()
s.pop()
!
!
! } { Code is more Maintainable

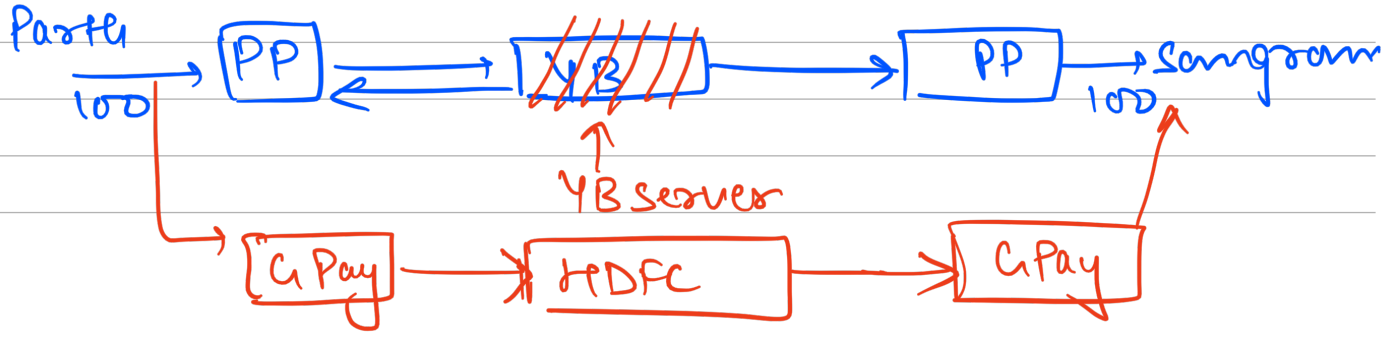
Principle: Program to an interface not to a implementation



Never attach to company
Always get attach skills

eg Phonepe → YES BANK

Phonepe was down 1 day



Q Phonpe wants to migrate from YES Bank to ZeZeI Bank

↳ 1 day

Fintech
30 days release

⇒ Interfaces

Tight Coupled

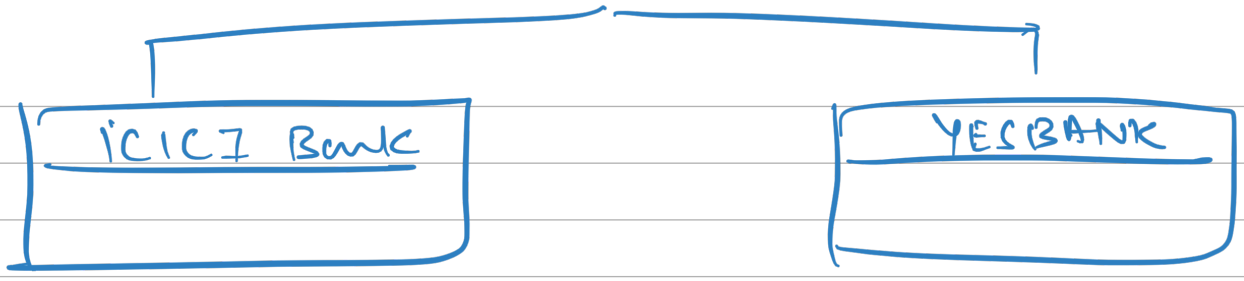
```
Class Phonpe {

    YBApi yb = new YBApi();
    ZeZeIApi i = new ZCICIApi();
        {yb.checkBal()
         i.getBal
         yb.SendMoney(ft,amt)
         i.transferAmt( ⟶ )
    }

Class YBApi {

    void SendMoney(from, to, Amount)

    int checkBal(Acc) {

    bool register(Acc)
    }
```

```
ICICI Api {

    boolean transferMoney(amount, from, to)

    int getBal( bank acctNumber)

}
```

Ideal Solution → Single pattern

```
<< BANK API >>

int checkBal( ⟶ );
bool transferAmt( );
```

Mandate By RBI

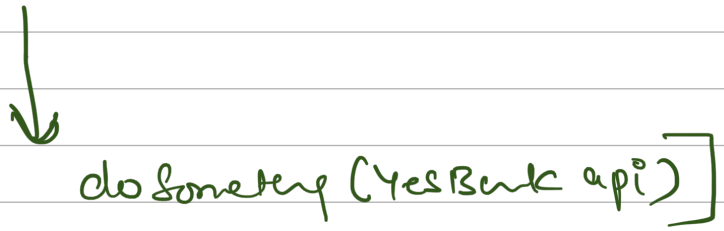


Class Phonepe {

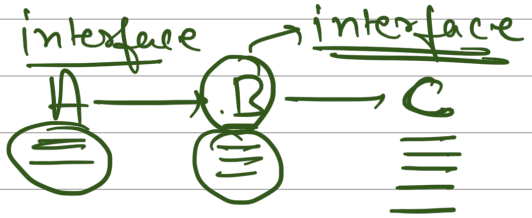
BankApi ba = ~~new YESBANK()~~ new ICICIBANK()

YESBank api = new YESBANK

ICICIBank api = ————



PaymentMethodApi PMA = CC() DC()



Assg4 ① Implement Stack using interface

① ArrayBasedStack()

② LinkedBasedStack()

Abstract Classes

A Entity with attrs and Behaviors

But don't 100% clarity on all the behaviors

```
Animal {

    String Name;
    int age;

    Void walk() {
        Sout("Dummy");
    }
}
```

instead

```
Abstract Animal {

    String Name;
    int age

    Abstract Void walk();
    abstract void eat();
    int numofeyes()
    {
        return 2
    }
}
```

If class has abstract Methods then class also needs to be abstract

Any child class of Animal will be forced by compiler to implement Abstract Methods

```
Tiger extends ~~Animal~~ Cat {
    Void walk() {
    sout("tiger walking");
    }
    Void eat() {
        Sout("tiger eating");
    }
}
```

```
Abstract Cat extends Animal {

    int numOfLegs()
    {
        return 4;
    }
}
```

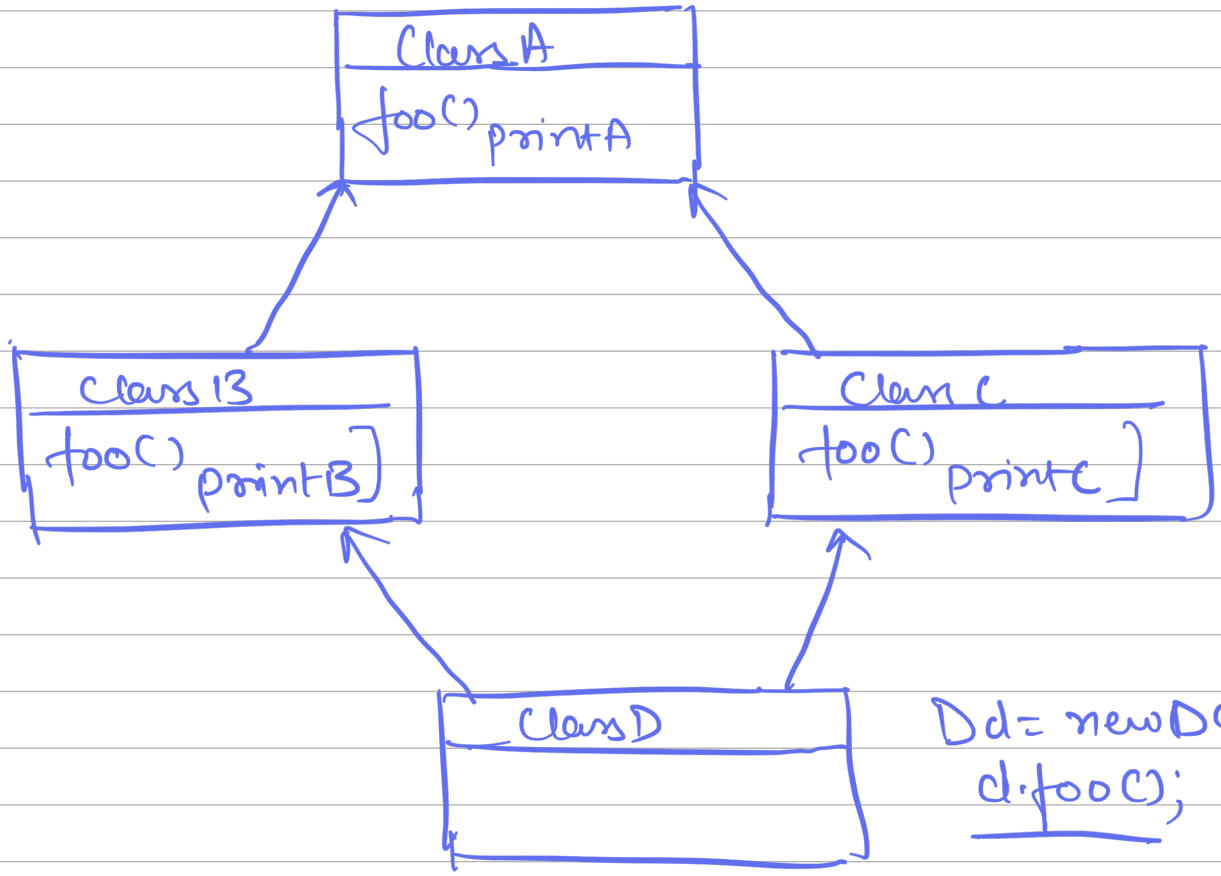
How to walk is not clear at CAT

If Child class can not implement all methods of parent then it also needs to be abstract

2) You can't create an object of abstract class.

2) An abstract class without any attribute ⇒ Interface

Q Why in JAVA multiple inheritance is not allowed



D d = new D()
d.foo();

Class D extends B, C

[Prefrence]

When to use abstract Classes

↳ NEVER use abstract classes

Use abstract class when we have a parent class whose object we never want to create

ANIMAL ⇐ Abstract

eg Tiger extends Animal

Lion extends Animal

X

Multithreading
Thread
class

Carnivores

[Tiger extends animal implements carnivores]

animal → foo

carnivores → boo

{ fooC)
booC)
kooC) }

Static Keyword

psvm()

↳ Public static void Main()

[accessModifier] [ReturnType]
[MethodName] ()

Q Do we call a method on a object ✓
or a class

Static → Variable
↳ Method

Static allows us to call method (without) w/o creating its object of corresponding class

⇒ Because static method doesn't require any object to be created, it can access only static attributes

[Static Attribute or Variable]

[public static final MAX = 256;]

Website for own University

↳ NIT Jaipur

Student → UnivName → String [72 x 10^5]

Static variable [UNIV NAME] → will be shared accross all objects

↳ 72

Constants

Role

static final String TA = "TA";

→ CONST

User u = new User()

u.setRole(Roles.TA); ①

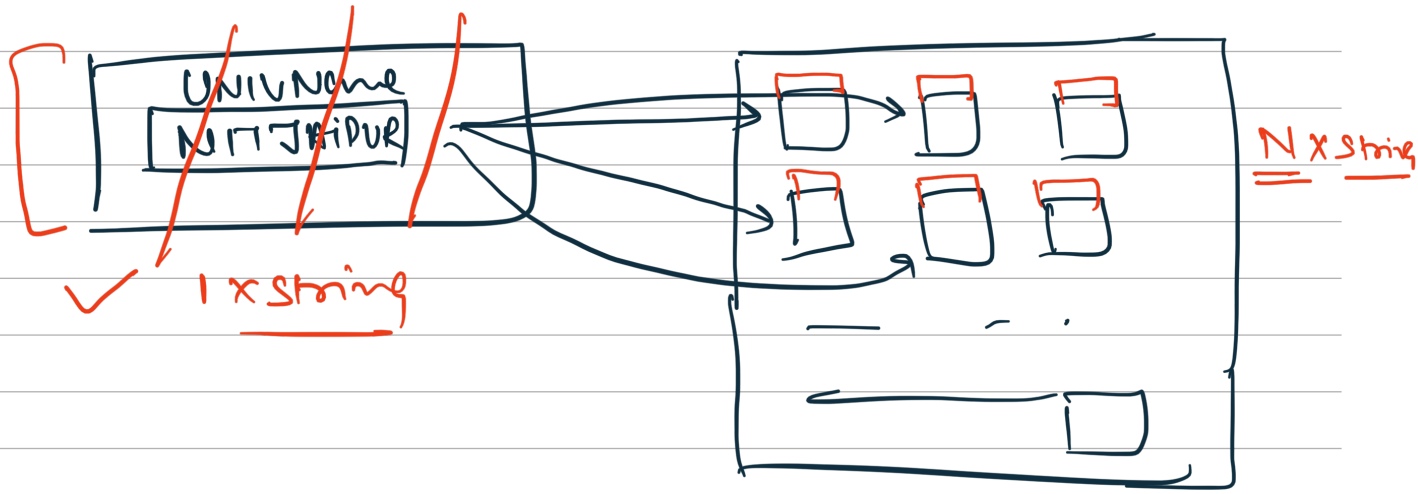
u.setRole("TA"); ②

↳ Teaching Assitant

100 times

u.setRole("TAA") TYPOS

Static keyword allows us to access the corresponding thing W/O creating an object of the class



Student {
int age=
Univ Nbr = XYZ
}

Static

XYZ

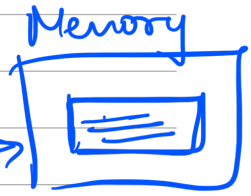
S1 age=20

S2 age 30

S3 age 40

User {
Public Static UnivName="P2"
}

Student ~~extends User~~
{
Utility.Helper();
UnivName = User.UnivName;



Utility Class{

Static Helper() {

POI Lib → Bulk upload / download