AGENDA

= 1.) Interfaces

= 2.) Abstract classes

start by 9:05 PM IST

*) INTERFACES:

```
Eat()
walk()      →  dog   }
runs()
```

Interfaces in OOPs.

B.P. of behaviour

```
Eat() →            Eat()
  {
     .....
}
```

→ set of methods w/o implementations

```
interface Animal {
    Eat();        •
    walk();       •    }
    sleep();      •
}
    class dog implements Animal {
        Public void Eat() {
        }

        Public void sleep() {


        }
        Public void walk() {

        }
  }

   class Fish implements Animal {
        .

        .

        .
   }
```

Real life usecases:

stack:
```
void  Push(x)
int   pop()
bool  isEmpty()
```

```
interface stack {

    void  Push(x)
    int   pop()
    bool  isEmpty()

}
```

class Array implements stacks

- int top
- int arr[]
- int Maxsize

```
Push(int x) {
    arr(top) = x
    top ++;

}
```

class Linked List implements stack

case 2·)

PhonePe

− yesbank

× yesbank....

Phonepe — YesBank ——×

(1 day)

HDFC

```
class YesBank {

    double checkBalance()          .
    bool   registerAccount()       .
    bool   transferMoney(A, B, x) .

}
```

```
class PhonePe {
 ×  Yesbank api = new B yesBank()

    addMoneyToWallet() {
        api. checkBalance() .
        api. transfer(A, B, 500) .
```

↑

}   ✓

```
class HDFC {                                  X X X

    int addMoney(....)  —
  double getBalance(.... )
           ⋮
}
```

Solution: → (N.P.C.I)

```
     interface BankAPI {

  =  double  FetchBalance(....)        ✓    }
  =  bool    transferMoney(... )       ✓

     }

                     HDFC, ICICI, ...

     class PhonePe {

         BankAPI api = new ICICI();

               . . . . .
                . . .
```

```
addMoneyToWallet() {
    api. checkBalance() •      }  ...
    api. transfer(A,B,500)•    }


}
```

N.P.C.I

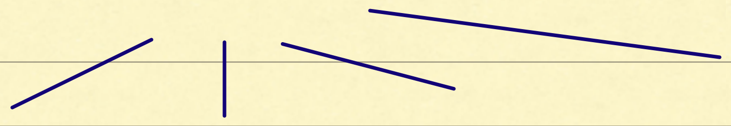
```
class x implement A, B {

        | A        =

        | B        =

}
```

When to create interface:

Multiple ways to do thing (x)

$\downarrow$

create interface x



10:17 PM IST

*) ABSTRACT CLASSES:

Entity + attributes.
Methods.

↓

SOME Method(S)
don't know implementation....

Abstract class.

atleast 1 Method - where you don't have
defination for that

```
class abstract Animal {
        string name;     } = ] =
        int age;

(E1) abstract void eat();     // this.  ]

     Public string getName() {   } ] =
          .....
     }
}
```

Requirement •) atleast (1) Method abstract
→ so class is also defined as abstract

Abstract class:

1·) atleast 1 abstract Method

2·) You cannot create Objects of Abstract class.

classes that extend Abstract class — should define the func. of abstract Methods

```
class Dog extends Animal {

    Public void eat() {
        . . . . . .
    }

}
```

→ Abstract classes v/s interfaces:

→ Multiple Inheritance

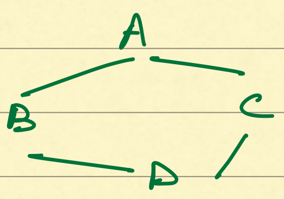
```
class A {
    greet(){
    }
}

class B extends A {          class C extends A {
    greet() { •                   greet() { •
    }
}                                 }
                              }

class D extends B, C {

}

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

✗ PROBLEM...

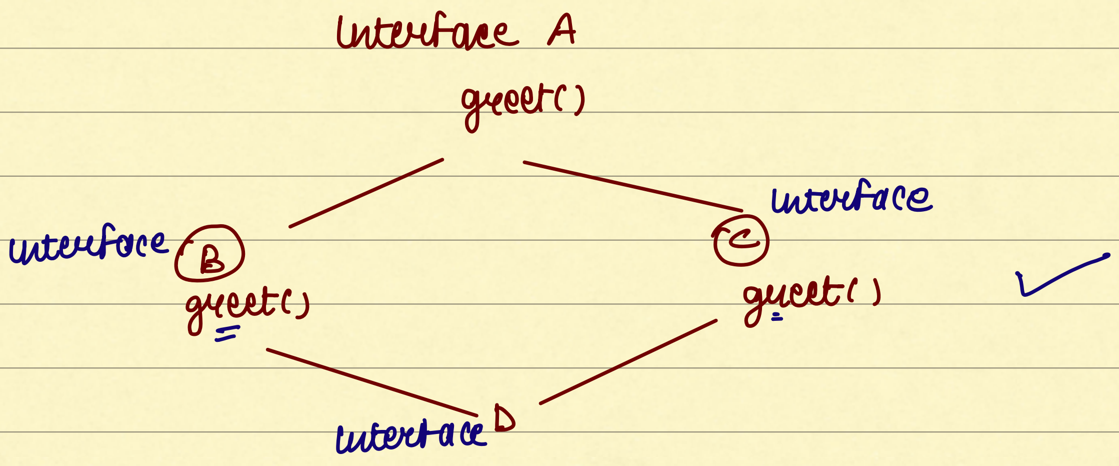
Multiple Inheritance is NOT allowed in

Many OOPs.

DIAMOND PROBLEM



Now, with interfaces:



class X implements D {

greet() {

.....

}

}

Abstract classes:

```
interface User {

        login();
        logout();

}

class User implements User {

    // login() {

    }

     logout() {
}    }

User   st = new User();

class abstract User {


     abstract login();
              logout();
```

✓

×

}

class abstract User {

↓ XXXX

} student extends User {

}

•) variable initializations

List< >

↓

interface in java