Agenda : ① Polymorphism - @ Method overloading
② Method overriding
③ Interfaces
④ Abstract classes
⑤ Static

class starts at 9:05 PM

① Method overloading

```
class A {
    ① public void doSomething() { }

    ② public void doSomething(String name) { }
}
```

⇒ doSomething is overloaded with diff. type of args. ⇒ method overloading

```
class Client {
    psvm() {

        A a = new A();
        a.doSomething();      —(1)
        a.doSomething("ABC"); —(2)
```

method overloading ⇒ compile time polymorphism

eg:

```
convertToString(int a)
convertToString(double d)
```

client → convertToString(10)
(10.0)

Ques Is this method overloading?

(1)

```
void print()
void print(String s)
```

② void print (String s)
void print (int s)

③ void print (String s, int a)
void print (int a, String s)

print ("ABC", 10)
print (10, "ABC")

④ void print (String s)
int print (String s)

print ("ABC")

function signature ⇒ name of function + datatype of args.

method overloading when name of function is same but overall function signature is diff.

② Method overriding

```
class A {

    void doSomething() {
        SOP("Hi")
    }

}

class B extends A {

    void doSomething() {
        SOP("Bye")
    }

}
```

between 5-6()

Method overriding ⇒ overriding / re-defining the function of a parent, but function signature + return type has to be same.

```
class A {
    void doSomething() {
```

SOP ("Hi");

}}

class B extend A{

int doSomething(){
SOP ("Hi");
}

}}

class B extends A {
@Override
void doSomething(){
SOP ("Bye");

}}

A a = new A();
A b = new B();
a.doSomething() ⟹ "Hi"
a.doSomething() ⟹ "Bye"

executed

⇒ actual method called is the one whose object is there at the run time.

```
class A {
    doSomething()
        → Hello
}
```

```
class B extend A {
    doSomething()
        → Hi
}
```

```
class C extend A {
    doSomething
        → Bye
}
```

List<A> = { new A(), new B(), new C(), new A() }

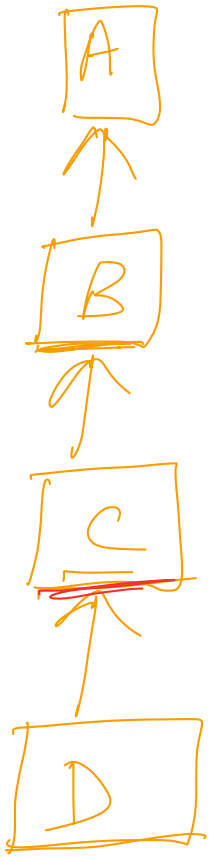
```
for obj in A :
    obj: doSomething()
```

{ Hello, Hi, Bye, Hello }

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

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

A c = new C();



doSomething() → Hi

doSomething() → Bye

① D d = new D();
d.doSomething();
→ bye.

② C c = new D();
→ c.doSomething();
↳ Bye

③ B b = new C()
b.doSomething();
↳ Bye

(4) A a = new B();
a.doSomething()
↳ Hi

(7) Interfaces ⇒ break till 10:25

Q. ⇒ How did we think about an entity

concept ⇒ class ⇒ attributes
⇒ functions

Q ⇒ did it contain the function definitions ⇒ yes

⇒ I want to define concept ⇒
no attrs / function definition.
concept that is categorized by the type of behaviour it represents.

⇒ walk(), run(), eat()
↳ Animal

⇒ any entity (class) which wants to be categorized as an Animal must be implementing their behaviours

⇒ we can do this by something called as interfaces

⇒ Class ⇒ Blueprint of an entity
Interface ⇒ Blueprint of behaviours

Interface ⇒ represents a set of behaviours that must be implemented by a class to be categorized of that particular type.

```
interface Animal {
    void run();
    void eat();
    void walk();
}

class Cat implements Animal {
```

```
@override
void run() {
    SOP("cat is running");
}

@override
void eat() { ==== }
void walk() { ==== }
}
```

Eg. 1 Stack ⇒ Linked list
⇒ Array

LIFO ⇒ Last in first Out

- push();
- pop();
- isEmpty();
- peek();

```
interface Stack {
    push();
    pop();
    isEmpty();
    peek();
}

class ArrayStack implements Stack {
    int counter

    void push() {
    }

    void pop() {
    }
}

class ListStack implements Stack {
}
```

LIFO

Assignment 1

Assignment 2


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

```
Stack st1 = new ArrayStack();
Stack st2 = new LinkStack();
```

Always code to an interface, and not to specific classes (implementation classes)

~~ArrayStack st = new ArrayStack();~~

Eg: 2 ⇒

To,
The principal

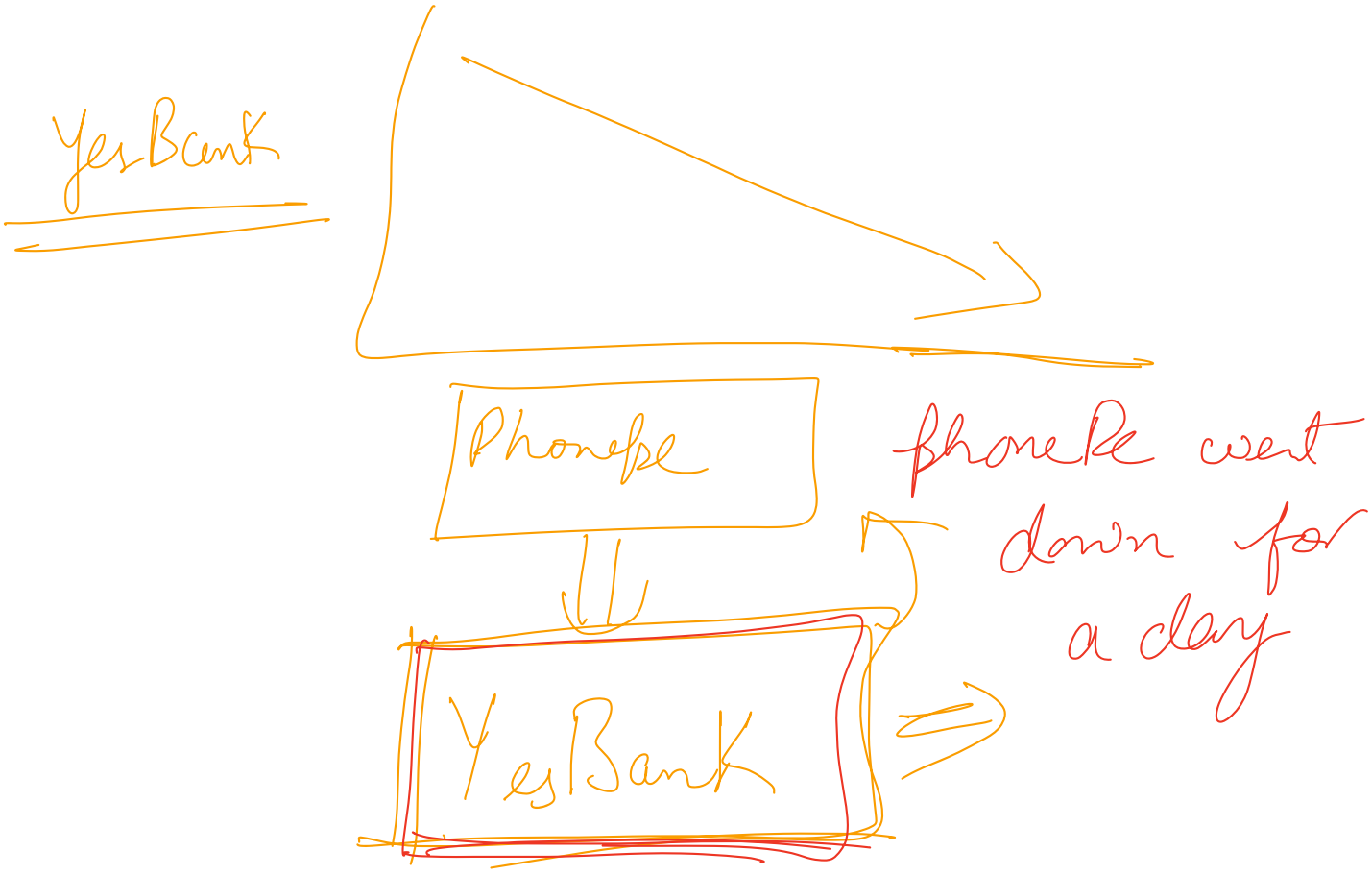
Regards
Abhinav

~~To
Mrs. Vineeta~~

Anyone who has the authority to grant you leaves, special request

↳ Principal

Eg 3 ⇒ PhonePe



Q ⇒ How long ⇒ 1 day they switched to ICICI

```
class PhonePe {
    YesBankApi yesBankApi;, ICICI
    yesBankAPI.checkBalance(String uid);
    yesBankAPI.transfer(String accountFrom, String accountTo, );
}

class YesBankAPI {
    checkBalance( )
    transfer(from, to, amt)
}
```

}

int amount

class ICICI {

checkBalanceBetweenCustomer( )

transferAmt(from, to, amount, remarks)

}

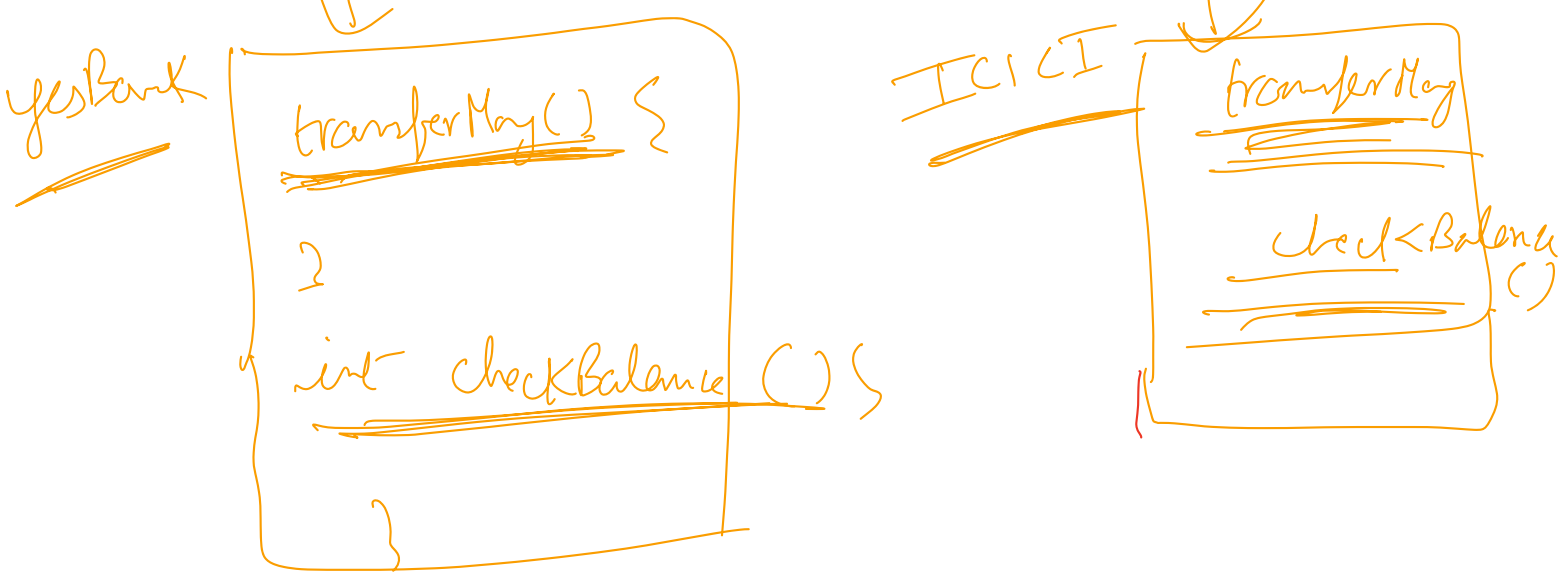
RBI mandated

BankAPI {

void transferMoney( )

int checkBalance( )

}



class PhonePe {

YesBankAPI yesBankAPI;

BankAPI bankAPI = new YesBank()

ICICI()

bankAPI.checkBalance( )

BankAPI.transferMoney();

}

Eg: 4 ⇒ In Java ⇒ List

List

Array ArrayList Dynamic Array

List<Integer> list = new ArrayList<>();

= new DynamicArray();

<<interface>>

doSomething()

class implement interface

doSomething()

class implement interface

doSomething()

Way 3

(2) Abstract classes ⇒ Let us say

we have an entity ⟹ it has few attrs. & few functions ⟹ we are not really clear about the function defⁿ.

```
abstract class Animal {
        String name;
    abstract void walk();
}

class Cat extends Animal {
    Void walk() {
        SOP("walking slowly");
    }
}

abstract class Dog extends Animal {
    abstract void walk();
}
```

Animal a = new Cat(); ✓

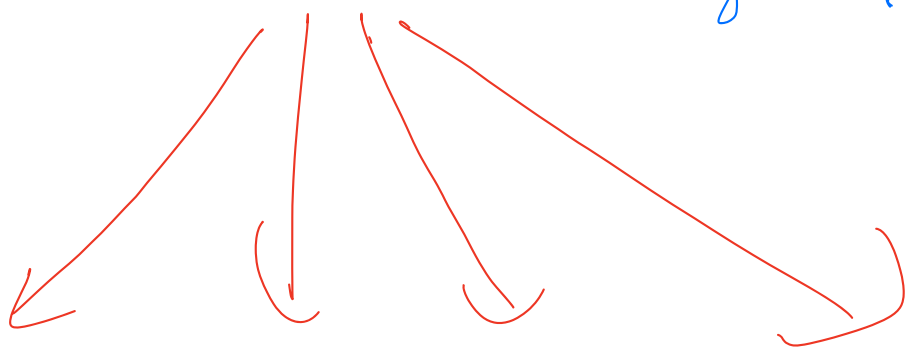
= new Dog(); ✗

⇒ we cannot create object of abstract classes as they are not complete

class C extends Dog {
    void walk() { }
}

Dog d = new C();

abstract doSomething() ;



doSomethig() doSothy()

Can I say interfaces are somewhat equivalent to abstract classes without

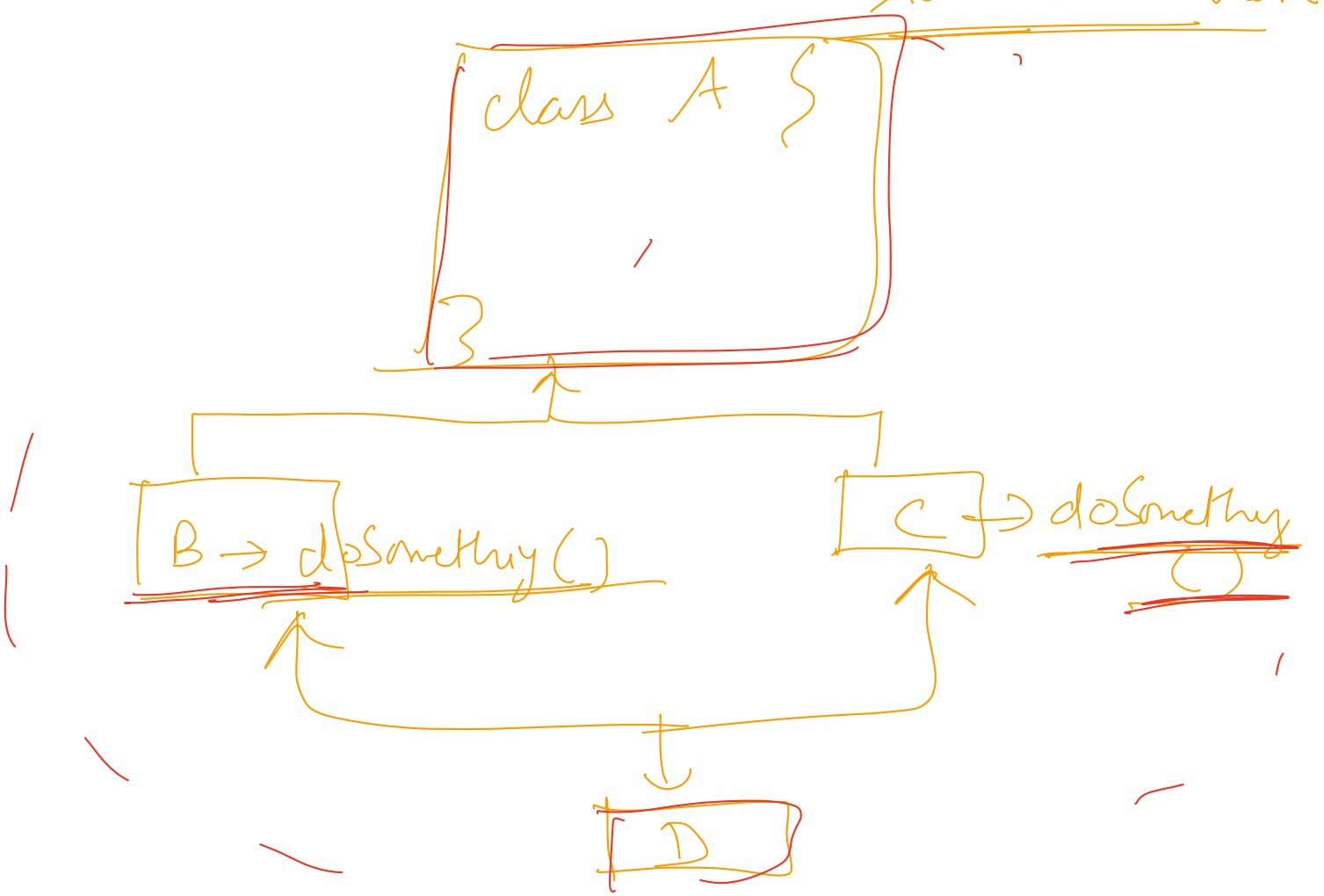
attributes ? ⇒ Yes

Ques When to use abstract classes & when to use Interfaces ??

Q1⇒ does Java allow multiple inheritance ? ⇒ NO

⇓

Diamond Problem



D d = new D();

d.doSomething(); ✗

interface A {
doSomethy();
}

interface B {
doSomethy();
}

class C implements A, B {
doSomethy() {
}
}

Client ⟹ C c = new C();
c.doSomethy();

Next days
(1) When to choose interface Vs abstract class
(2) static
(3) Destructors

class C extend A, B