oops

★ Object Oriented programming :-

↳ Entity (class)

↳ data (attributes)

↳ behaviour (methods)

★ Pillar of OOP

i) Inheritance

ii) Polymorphism

iii) Encapsulation

Abstraction :-

↳ Hiding the complex impl.

Python

Java

C++

C

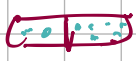
Assembly

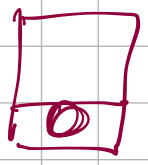
machine code

STL :-

↳ Queue (Vector, List, Array)

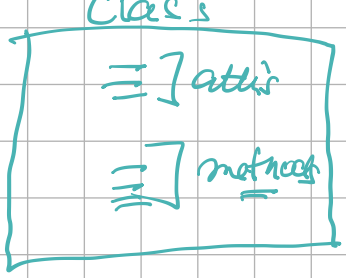
i) Encapsulation

↳  → Protection layer



Mobile

Samsung, IPhone ...



Blue print of an object

⇒ Access modifiers :- (Visibility)

⇒ Public, protected, default, private

Strictness increasing →

(i) Public :-
Same class, Same package, Other package.

(ii) Protected
Same class, Same package, Child class
↓ Same package → Other package

(iii) Default
Same class, Same package.

(iv) Private
Same class

	Same Class	Same pkg	Child in Same pkg	Child in other pkg	Other pkg.
Private	✓	✗	✗	✗	✗
Default	✓	✓	✓	✗	✗
Protected	✓	✓	✓	✓	✗
Public	✓	✓	✓	✓	✓

★ Constructor :-
↳ Used for a creation of an Object

↳ Default constructor will be present if I don't specify constructor on my own

```
class Student {
    int age;
    String name;
    public Student (int age, String name) {
        this.age = age;
        this.name = name;
    }
}
```

parameterized constructor

Student s = new Student(); ✗

Student s = new Student (20, "Alec");

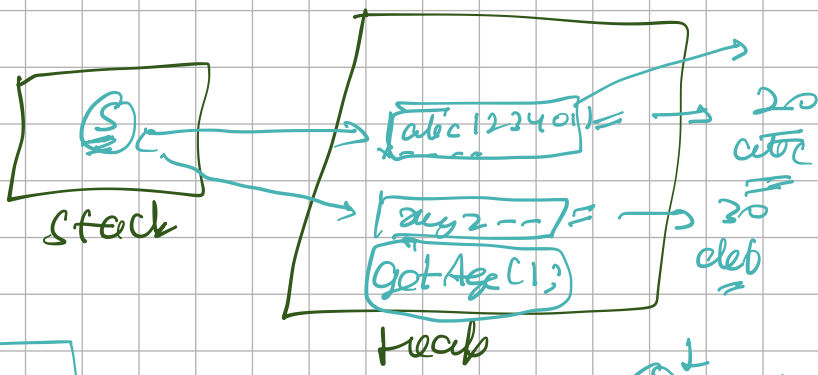
Default constructor won't be there:

Student s = new Student (20, "Alec"); → Runtime

Stack memory

Ref in heap memory.

addrs → 8 Bytes



s.age;

s.name;

s → age;

s.getAge();

set age;

S_1 = { age = 20, getAge(); }

S_1.prototype.getAge();

i) Constructor doesn't have set type.

ii) Same as class Name.

☆ Copy constructor :-

Student s_1 = new Student (20, "alec");

Student s_2 = new Student (s_1);

class Student {

private age, name;

Student (int age, String name) {

}

```
student (student s) {
    this.name = s.name;
    this.age = s.age;
}
```

↳ This is allowed even when name and age both are private.

☆ Shallow copy vs deep copy

↳ Is Java pass by ref, pointer or value? → Java

C++ `void swap(int &a, int &b)`

```
a = a^b;
b = a^b;
a = a^b;
```
→ Swap

a → a^b
b → a^b^b
0

```
3
a=5, b=6 ] → a=6, b=5
swap(a, b)
```

Java → Only pass by value.

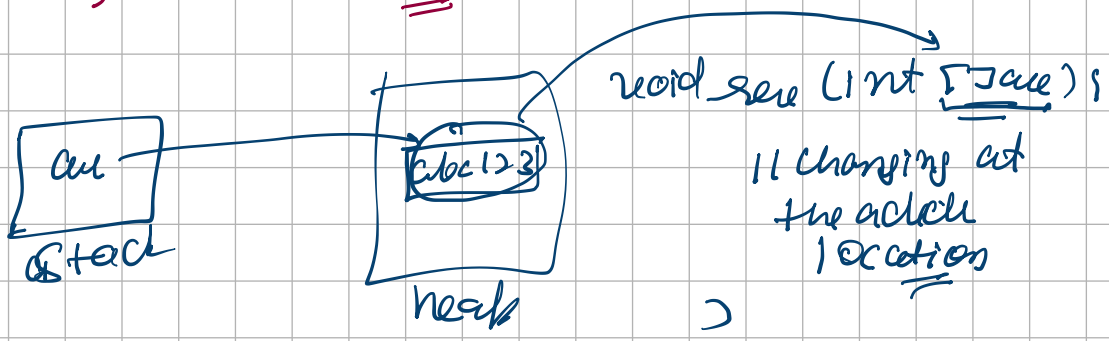
```
void see(int [] arr) {
    ≡ code to reverse
}
```
→ 123ab

```
see(new int[]{1,2,3})
```
→ 123ab

arr (Stack) → 123ab (Heap) ← ref

It will see the array

```
int [] arr = new int [] {1, 2, 3};

rev(arr); // To reverse
```




```
void swap (Integer a, Integer b) {
    a = a ^ b ]
}
```

Box it before move after wrapper.

↳ We will see ↳ final and immutable

Shallow copy

```
class Student {
    int age;
    List<String> subjects
    Student(age, subjects) {
        this.age = age;
        this.subjects = subjects;
    }
}
```

→ shallow copy

```
List<Subjects> list = Arrays.as
    List("Maths", "Eng")
Student s1 = new Student(20, list);
Student s2 = new Student(22, list);
```

⇒ S_2 wants to opt for PT

```
s2.getSubjects.add("PT");
```

S_1.getSubjects(): → ["Maths", "Eng", "PT"];

✱ Deep copy :-

```
class Student {
    int age;
    List<String> subjects
    Student(age, subjects) {
        this.age = age;
        this.subject = new ArrayList<>(subjects);   // copy
    }
}
```

→ Deep copy

```
List<Subjects> list = Arrays.as
                      List("Maths", "Eng");
Student s1 = new Student(20, list);
Student s2 = new Student(22, list);
```

Student s_2 opts for subject Physical Edu.

```
s2.getSubjects().add("PT");
```

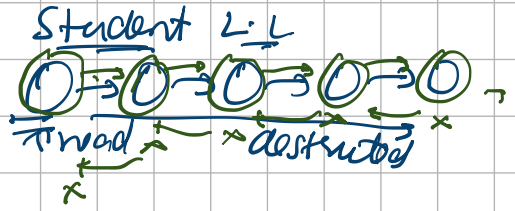
```
s1.getSubjects(); → ("Maths", "Eng")
```

✱ Destructor :- (Java closes it none it)

↳ When scope of object is finished destructor is invoked

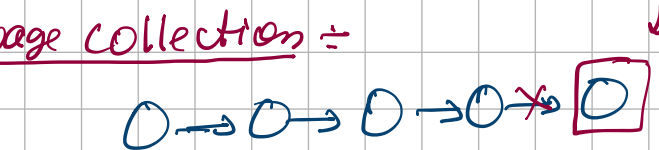
```
~Student() {
    // → deletion task
    delete this->next;
}
```

After fully execution object is gone.



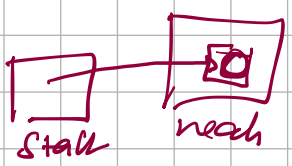

```
delete head;
```

✱ Garbage collection :-



head

Background thread



Spring boot

↳ Singleton ⟹ @Autowired ⟹ 100 req

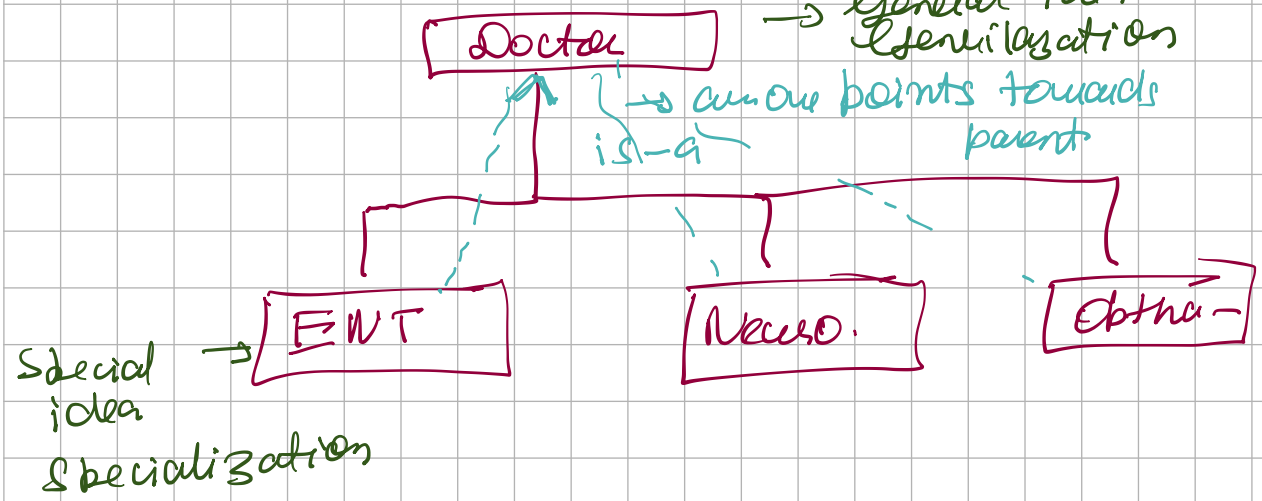
↓ request comes

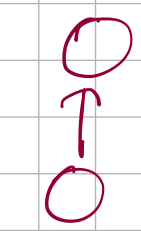
new RestTemplate<>();

restTemplate.doSomething();

Heap memory Overflow.

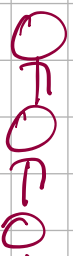
☆ Inheritance (is-a relation)





Single

i)

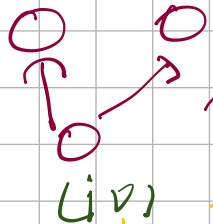


Multilevel

(ii)

Hierarchical

(iii)



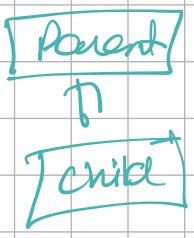
Multible

(iv)

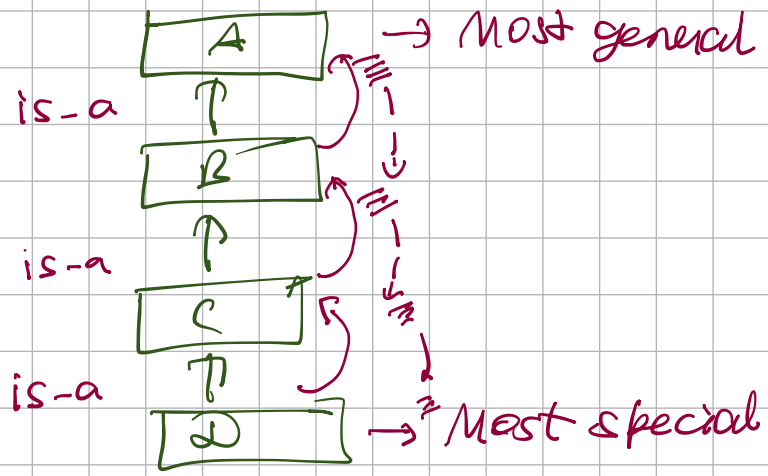
(Java doesn't support multiple inh. directly)

✯ Constructor chaining

```
public class Parent {
    private int age;
    private String name;
    public Parent(age, name) {
    }
}
```




```
public class Child extends Parent {
    private int grade;

    i) public Child(age, name) {
           super(age, name);
       }

    ii) public Child(age, name, grade) {
           super(age, name);
           this.grade = grade;
       }
}
```




```
Dd = new D(...);
```

★ Polymorphism ÷ Polygon + Polynomial

many forms — many sides — many power.

EMT is a doctor
Otho is a doctor

Doctor d = new EMT();

check Bar();

is-a

Emt e = new Doctor(); ✗ wrong

d.checkBar(); ✗

((EMT) d).checkBar();

type casting procedure.

⇒ Combile time and runtime.

⇒ Combile Time
- ↳ Method Overloading
- ↳ Operator overloading

⇒ Runtime
- ↳ Method overriding.

Method overloading

run(String s);
run(String s, int w);

add(long a, long b)
add(int a, int b)
↳ Casting

⇒ PK

add(1, 2);

add(1L, 2L);

↳ int

this is invoked

⇒ S_1 (Student) + S_2 (Student)

↳ add two age's

$S_3 = S_1 + S_2$

⇒ Eg:

```
Doctor {
public doCheckup() {
}
}
```

```
Ent {
publiccheckEar();
@Override
doCheckup() {
}
}
```

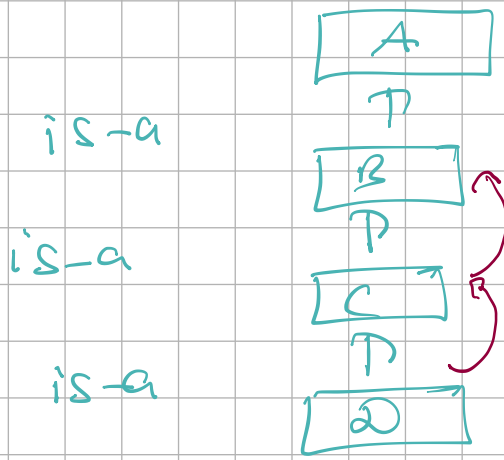
not important to write

Doctor d = new Ent();

d.checkEar(); ✗ not allowed

→ d.doCheckup();

↳ in compile time it will check whether this method is present or not



A a = new D();

a.doPlay();

B's doPlay is called

PK

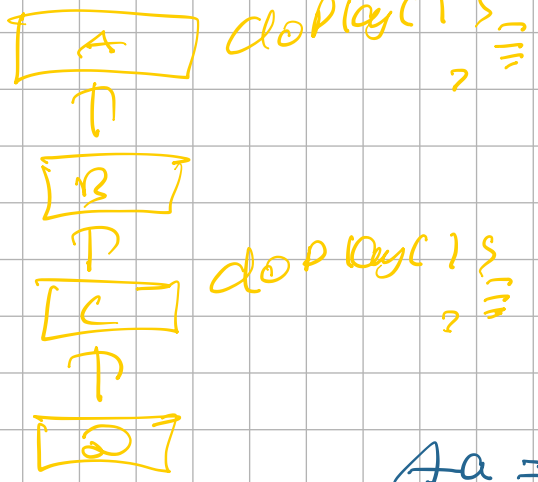
B

Cupil

A

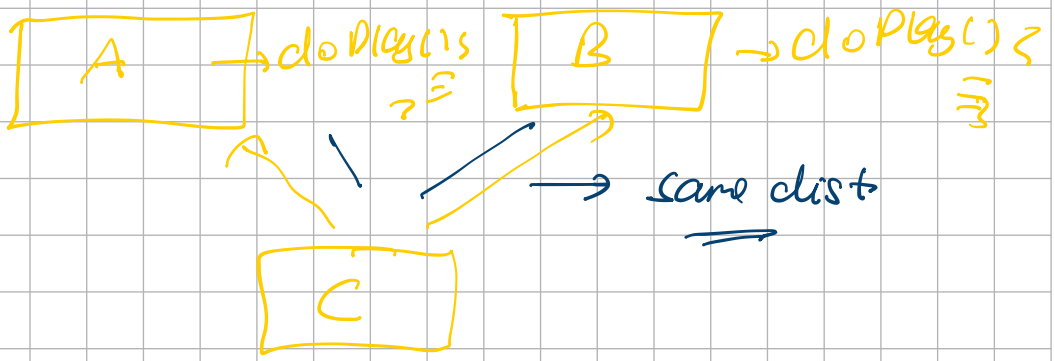
```
C c = new C();
c.doPlay();  → B's doPlay
```

```
A a = new A();
a.doPlay() → A's
```




```
A a = new D();
a.doPlay() → C will be called
```

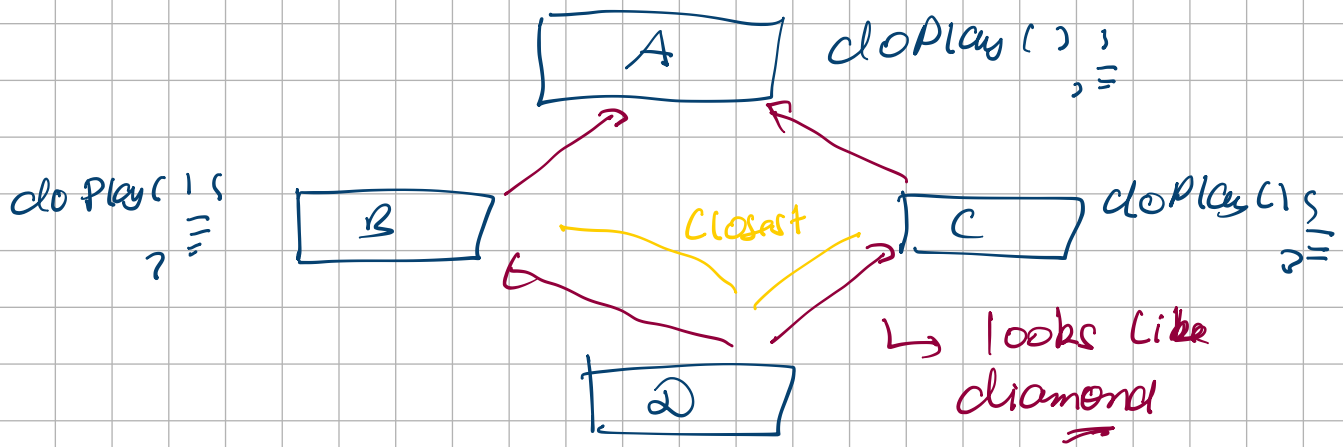
```
B b = new B();
b.doPlay(); → A will be called
```



↳ This is not allowed dereferencing.

```
A a = new C();
a.doPlay(); (ambiguity)
```

☆ Diamond problem



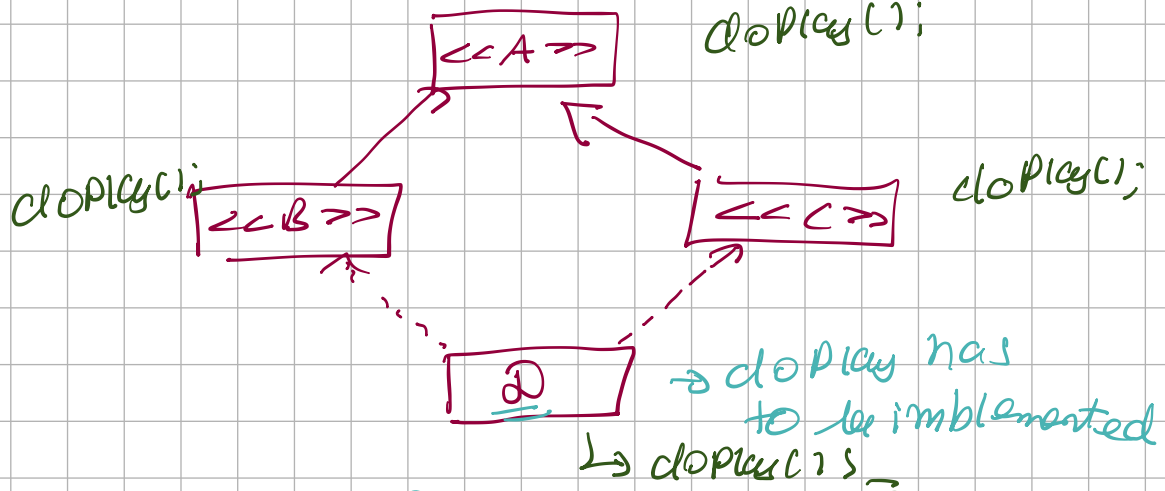

```
A a = new D();
a.doPlay();   [not able to decide]
```
Ambiguity

⇒ Interfaces can help solve diamond problem.

↳ Blue print of a behaviour

```
public interface A {
void doPlay();  → public abstract
}

public interface B extends A {

}

public interface C
    extends A {
}

public class D implements B, C {
    void doPlay() {
        ] some code
    }
}
```

3

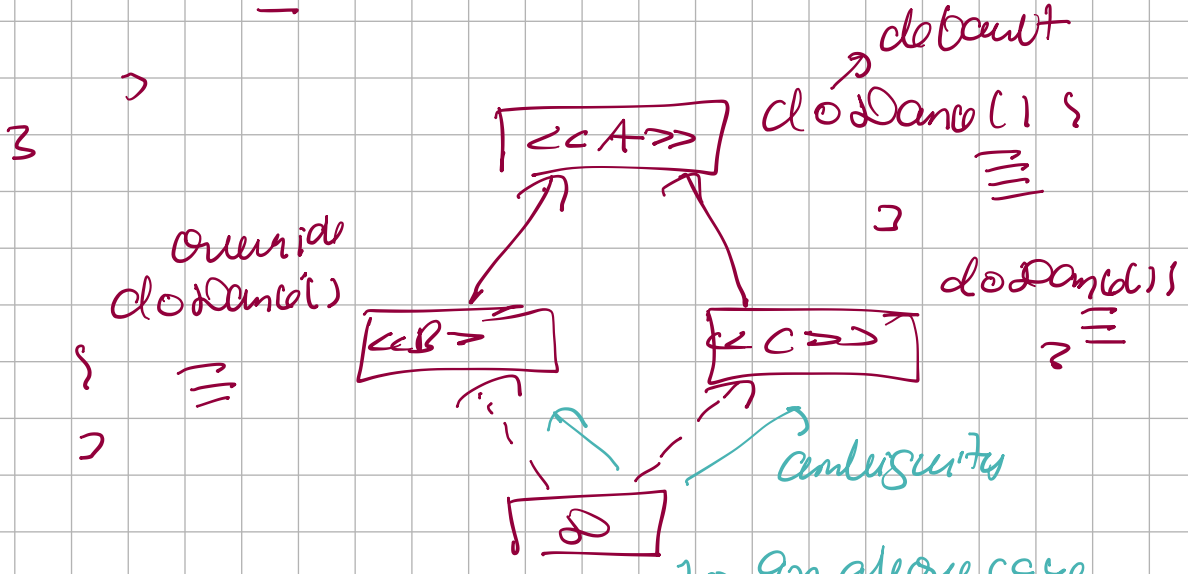


A a = new D();
a.doPlay();

A a = new A(); ✗ Not allowed

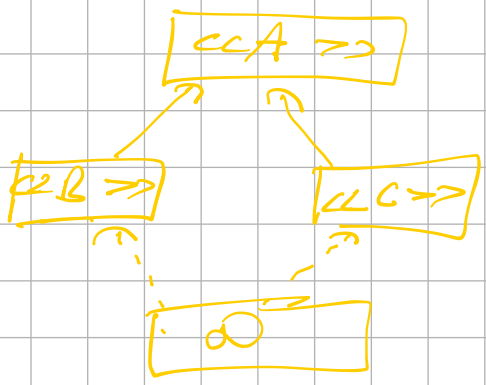
Java 8+ → added default methods to interfaces

```
public interface A {
    void doPlay();
    default doDance() {
    }
}
```



A a = new D();
a.doDance(); →

☆ In case



doDance() {
}

doDance() {
}

In above case B has not overridden doDance() ; so it's not important to override at D level.

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

→ We will generally see this syntax

```
void
public Pabc ( List<> ---- ) {

}
```

List<Integer> list = new ArrayList<>();
ArrayList<Int> arr = new ArrayList<>();
↓ → xyz();

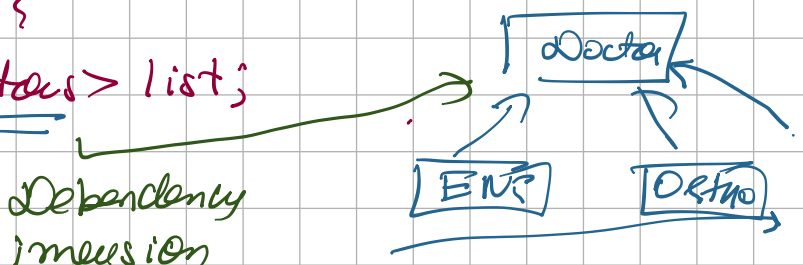
arr.xyz();

list.xyz(); ✗ not inside the list

class Hospital {

List<Doctors> list;

}



Dependency Inversion

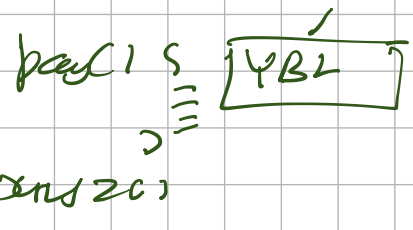
☆ Depend on interface rather than concrete impl.

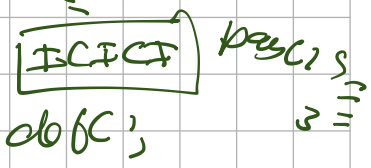
☆ Phonepe :

↳ @ybl (yesbank)

@Override

@icici





YBL ybl = new YBL();

ybl.xyz();

RbiInterface rbi = new YBL(); ICICI();

rbi.pay();

☆ Abstract class :

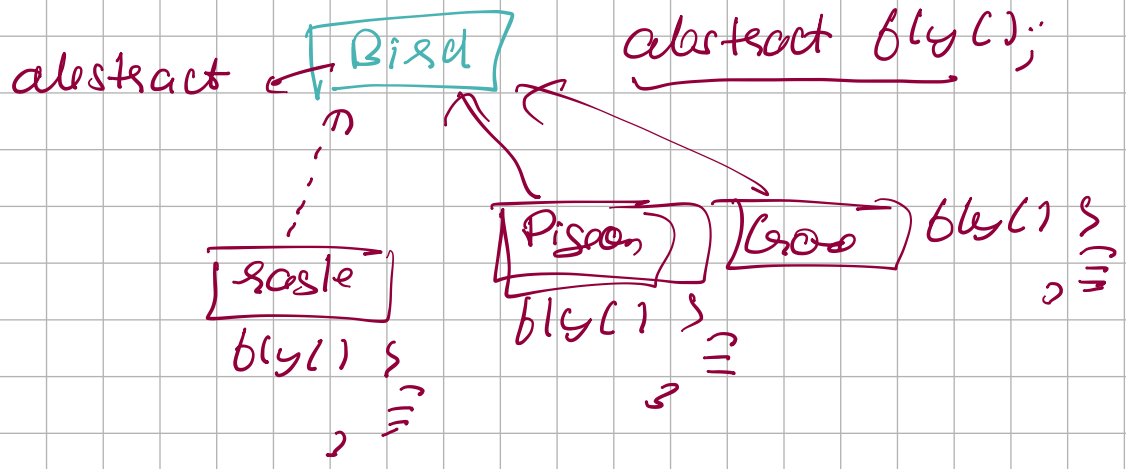
↳ Bird

↳ colour
↳ beak
↳ eyes
⋮

attr's

fly();
sleep();
eat();

Assuming every bird can fly();



⇒ Can we make Object of abstract class?

Ans

```
Bird b = new Bird();
b.fly();
```

you cannot create Object of abstract classes and also interface

☆ Final keyword :

```
const int a = 10;
```

```
class Bird {
    final int noOfFeathers;
    void setNoOfFeathers(int f) {
        this.noOfFeathers = f;
    }
}
```

It will give error

Above code will give error

☆ How to assign value to final.

1) Direct

```
class Bird {
    final int feathers = 2;
}
```

2) Constructor

```
class Bird {
    final int feathers;
    public Bird(int feathers) {
        this.feathers = feathers;
    }
}
```

// Instance block

```
class Bird {
    final int feathers;
    {
        feathers = 2;
    }
}
```

Instance block

⇒ class and methods could be made final

Final class could not be extended

↳ Can a abstract class as final?

↳ No (Abstract class cannot be final)

✱ Static keyword :-

↳ Statics are members of class & other than object.

⇒ Count no. of objects of a class

```
public static int ct;
Student() {
    ct++;
}
```

In case of destruction / garbage collection this ct may become invalid

How to access static ?

Student.ct--; ✓

=

Student st = new Student();

Object access → st.ct++;

st works but not recommended

```
class Student {
    private int totalCount;
    static void getTotalStudents() {
        sout(this); → Not allowed
    }
}
```

this doesn't make sense here

Non static members cannot be accessed static context.

Static and final ✓

↳ Static attr gets initialized during bootup

```
public class Test {
    private static final int abc = 2;
}
```

// Static block

```
public class Test {
    private static final int abc;
    static {
        abc = 2;
        sout ...
    }
}
```

Constant

→ Static block

```
        public static void main (String ... args) {
            System.out.println("Hello World");
        }
    }

class Constants {
    public static final String CAT_HOURS = "CAT_HOURS";
}
```