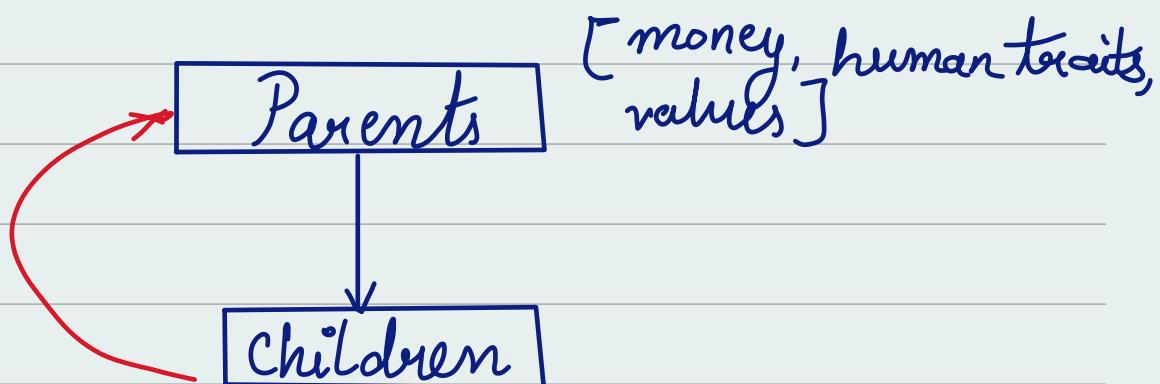


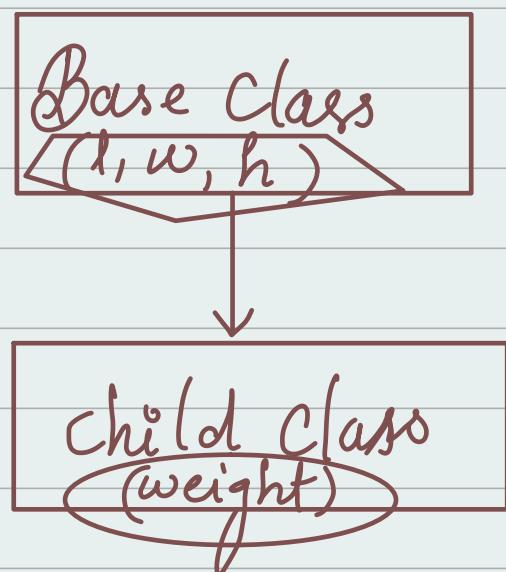
\* Use Command + N (to create a constructor)

- Static can run without an object
- We cannot access non static stuff without referencing their instances in a static context.
- Static is not dependent on<sup>the</sup> objects

i) Inheritance :-

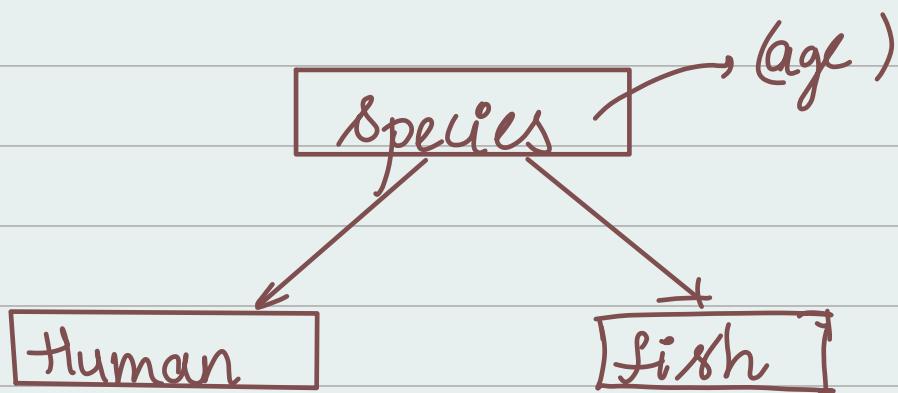


In OOP



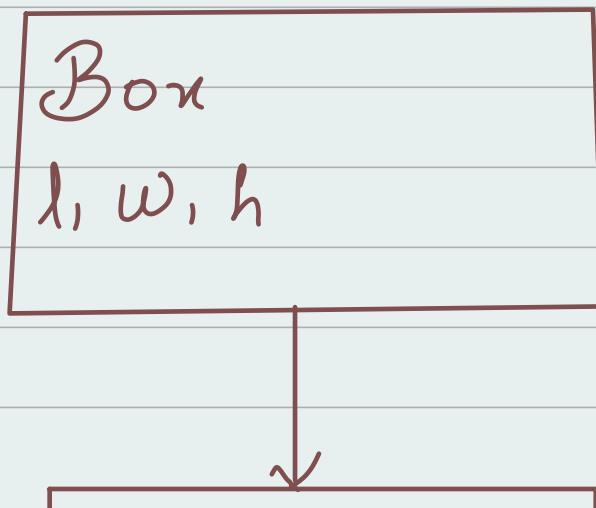
Child class is inheriting properties from Base class.

class Child extends Base {  
 int weight;  
 }  
 Child child =  
 new Child(),  
 child.l();  
 child.w(),  
 child.h();



Human Rohit = new Human();  
 Rohit.age

Initialize parent class variable  
 also



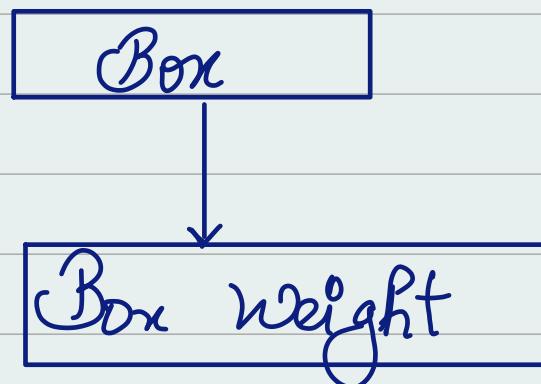
Box weight  
weight

Super class won't have any knowledge of the child class and what it contains.

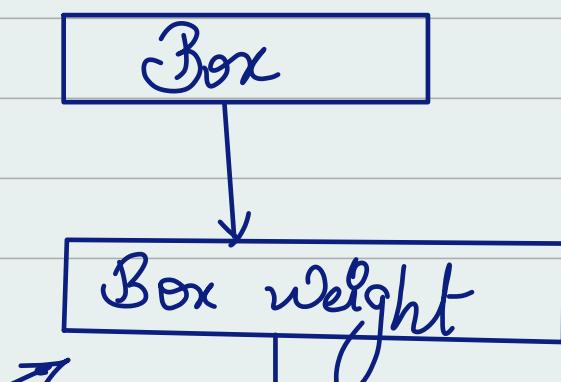
→ But child <sup>class</sup> cares about what parent class contains.

## Types of Inheritance

- 1) Single inheritance
- \* One class extends another class



- 2) Multilevel Inheritance :-



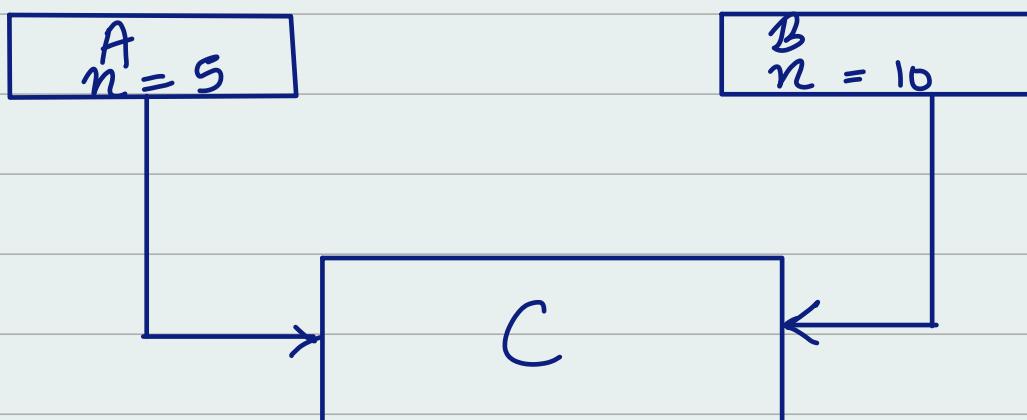
## Box Price

By using Super() it will call

the above function ex: Super() of Box Price is Box weight and Super() of Box Weight is Box.

3) Multiple Inheritance : We will do this in interfaces

One class extending more than one classes



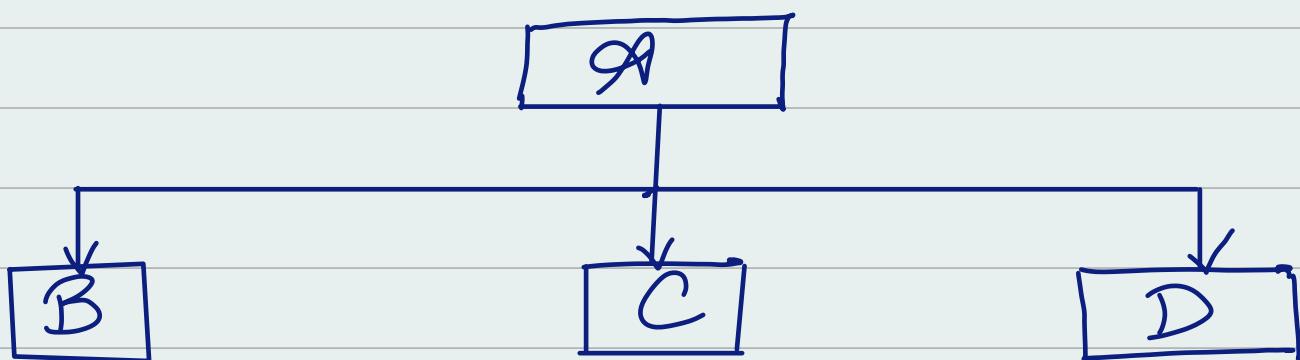
C object = new C();

C.n // what is it going to print?

\* Because of Ambiguity, multiple inheritance is not supported in Java. Because two or more classes have same m values

#### 4) Hierarchical Inheritance :-

One class is inherited by many classes.

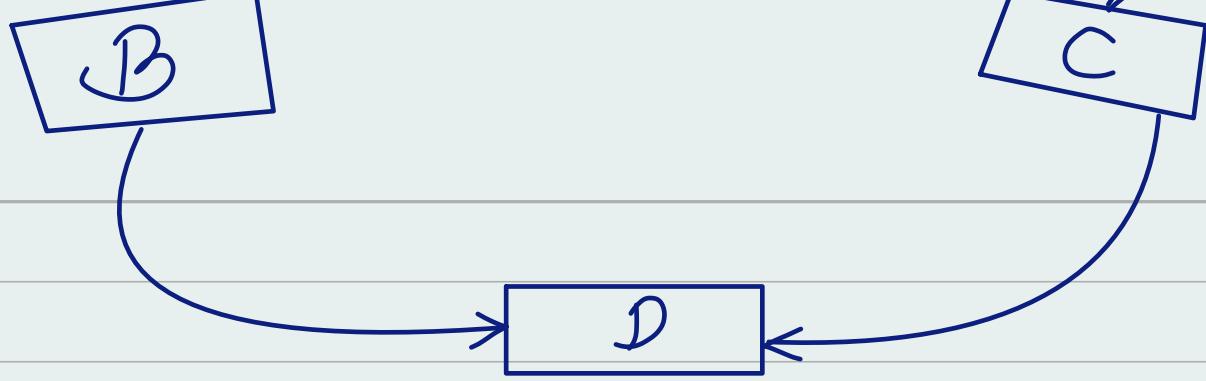


#### 5) Hybrid Inheritance :-

Combination of single and multiple inheritance

(Not in Java check in Interface lecture)

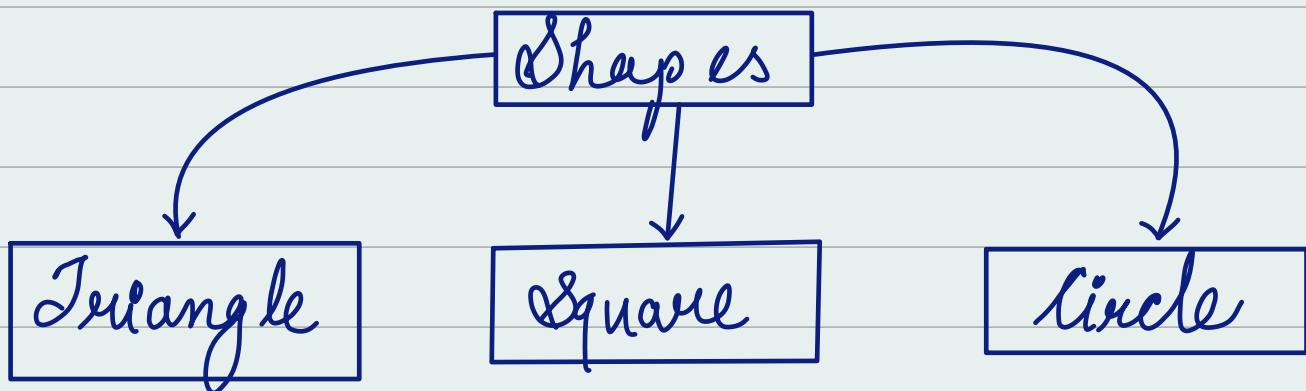




→ A class cannot be its own class.

\* Polymorphism : Poly means many  
morphism are ways to represent-

→ Shapes :-



Types of Polymorphism ?

1) Compile time / static polymorphism :-

Achieved via method overloading

Same name but types, argument, return

types, ordering can be different

Ex: multiple constructors:

A a = new A();

A a2 = new A(3, 4);

2) Run time / Dynamic Polymorphism

Achieved by method overriding

Parent obj = new Child();

Here which method will be called depends on child(). This is known as Upcasting. This is how overriding works.

→ Which particular area to call?

→ It happens by Object and how Java determine this → Dynamic Method Dispatch  
It determining

- So the name give Dynamic Polymorphism
- You cannot override a final method
  - ① Override // final X

Ex: X final void area() { cout("")  
 }

- Late Binding & Early Binding.

Check notes in Pdf

- Sometimes you would want to prevent the class from being inherited for that you can just put it as final.

- Can we override static methods?
- Overriding depends on objects and static method does not depend on objects. Hence, we cannot override static methods.

# Encapsulation :-

\* Wrapping up the implementation of the data members and methods in a class. It's for solving implementation level issue. It's focuses on the internal working.

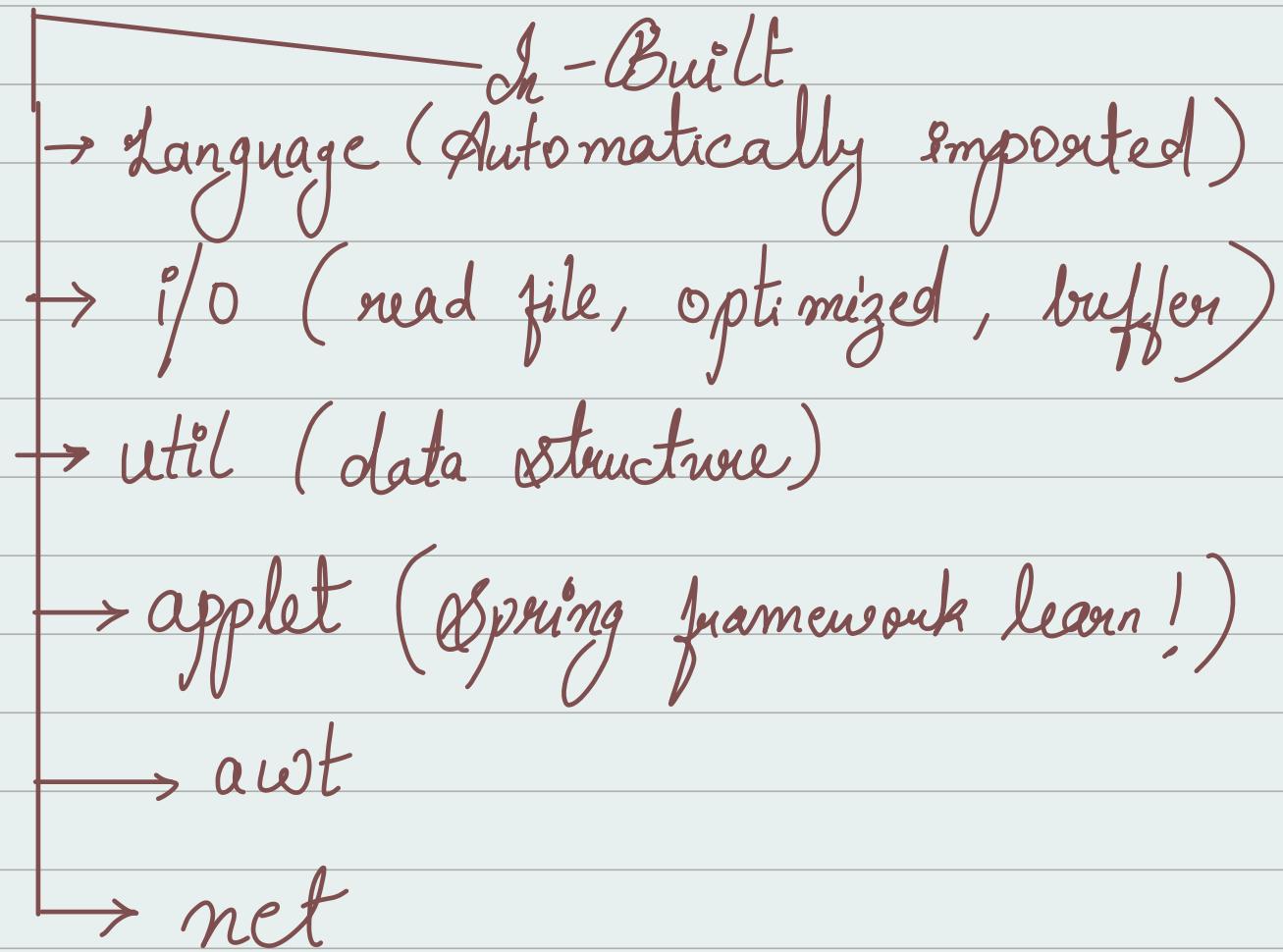
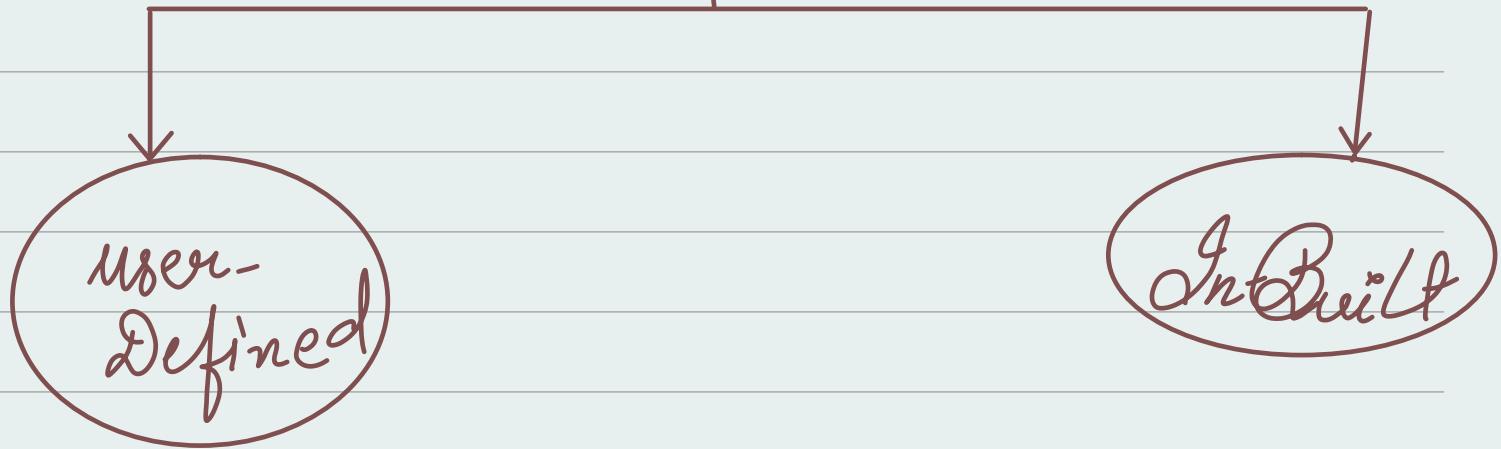
## Abstraction :-

\* Hiding unnecessary details and showing valuable information. It's for solving design level issue. It's focuses on the external working.

## Data hiding and Encapsulation :-

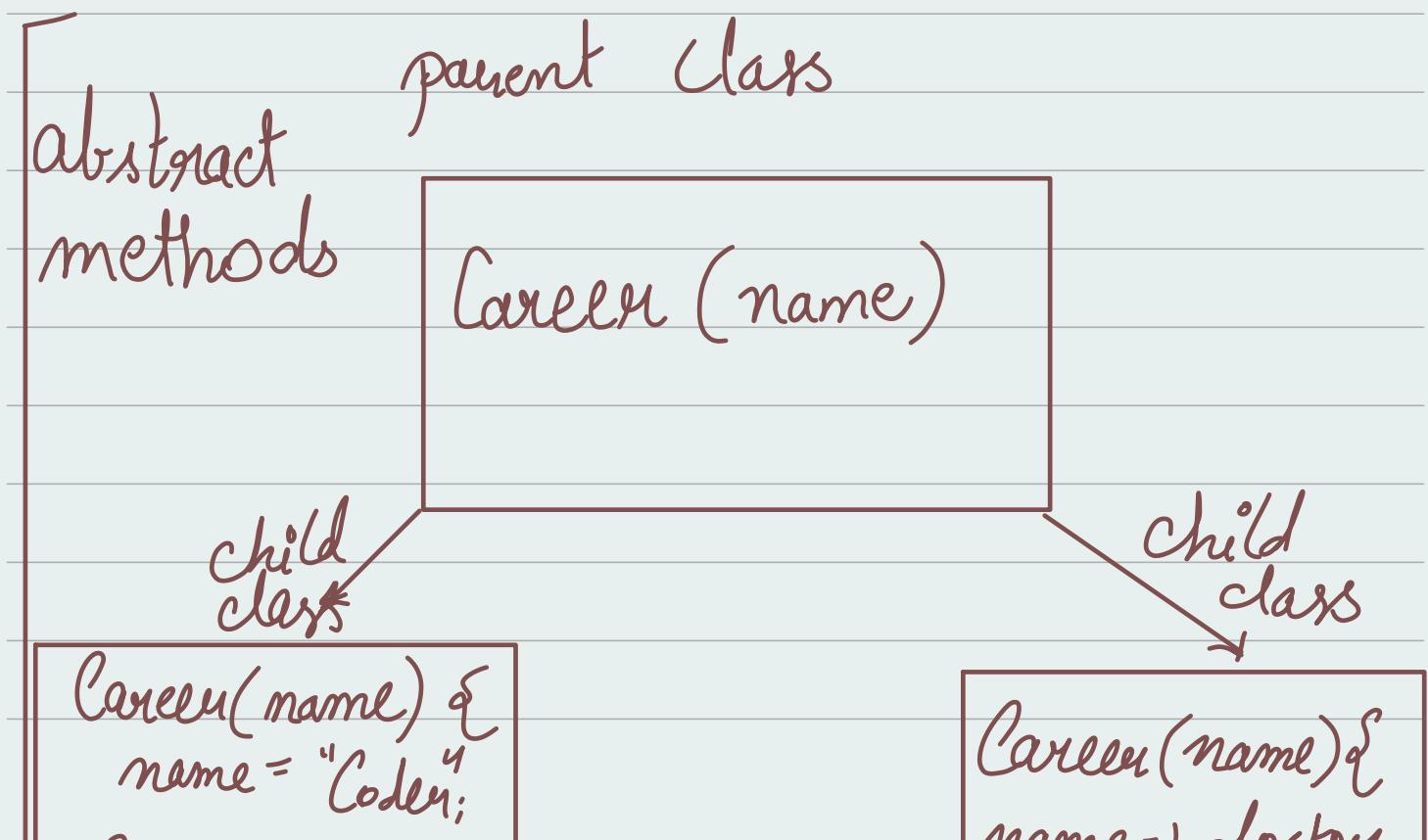
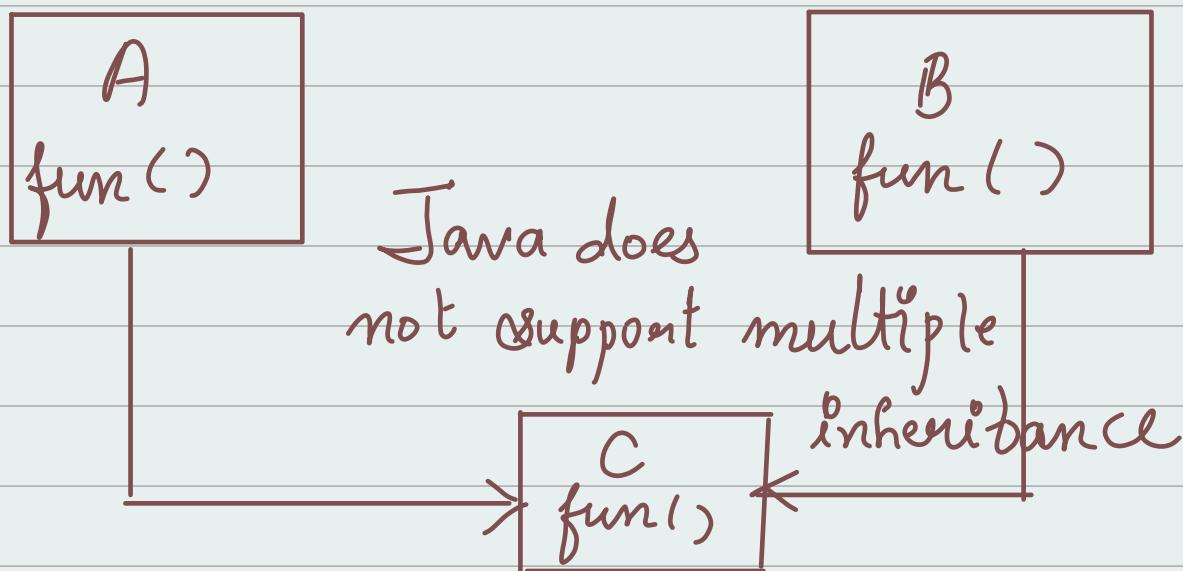
Data hiding is focused on data security and encapsulation focuses on hiding the complexity of the system.

# packages



# Abstract Classes.

## 9) Interfaces, Annotation



It will define the body for itself to support the multiple inheritance. So child class will override all the necessary methods.

Parent Class will provide empty methods and child class will override the methods.

abstract void career (String name);  
If any class that contains one or more abstract methods, must also be declared as abstract.

\* Interfaces have abstract method contains:-  
(non abstract as well as abstract method) It can also have default and static method.



# Lambda function :-

```
import java.util.ArrayList;
```

```
public class LambdaFunctions {
```

```
    public static void main(String[] args) {
```

```
        ArrayList<Integer> arr = new
```

```
        ArrayList<>();
```

```
        for (int i = 0, i < 5; i++) {  
            arr.add(i + 1);  
        }
```

```
        arr.forEach((item) -> System.out.  
                    println(item * 2));
```

3

operation sum = (a, b) → a + b;

operation prod = (a, b) → a \* b,

operation sub = (a, b) → a - b ;

Lambda Functions myCalculator = new

## Lambda Functions () ;

cout ( myCalculator operate ( a : 5 , b : 3 , sum ) ),

cout ( myCalculator . operate ( a : 5 , b : 3 , prod ) );

cout ( myCalculator operate ( a : 5 , b : 3 , sub ) );

private int operate ( int a , int b , operation op )

return op operations ( a , b ) ;

}

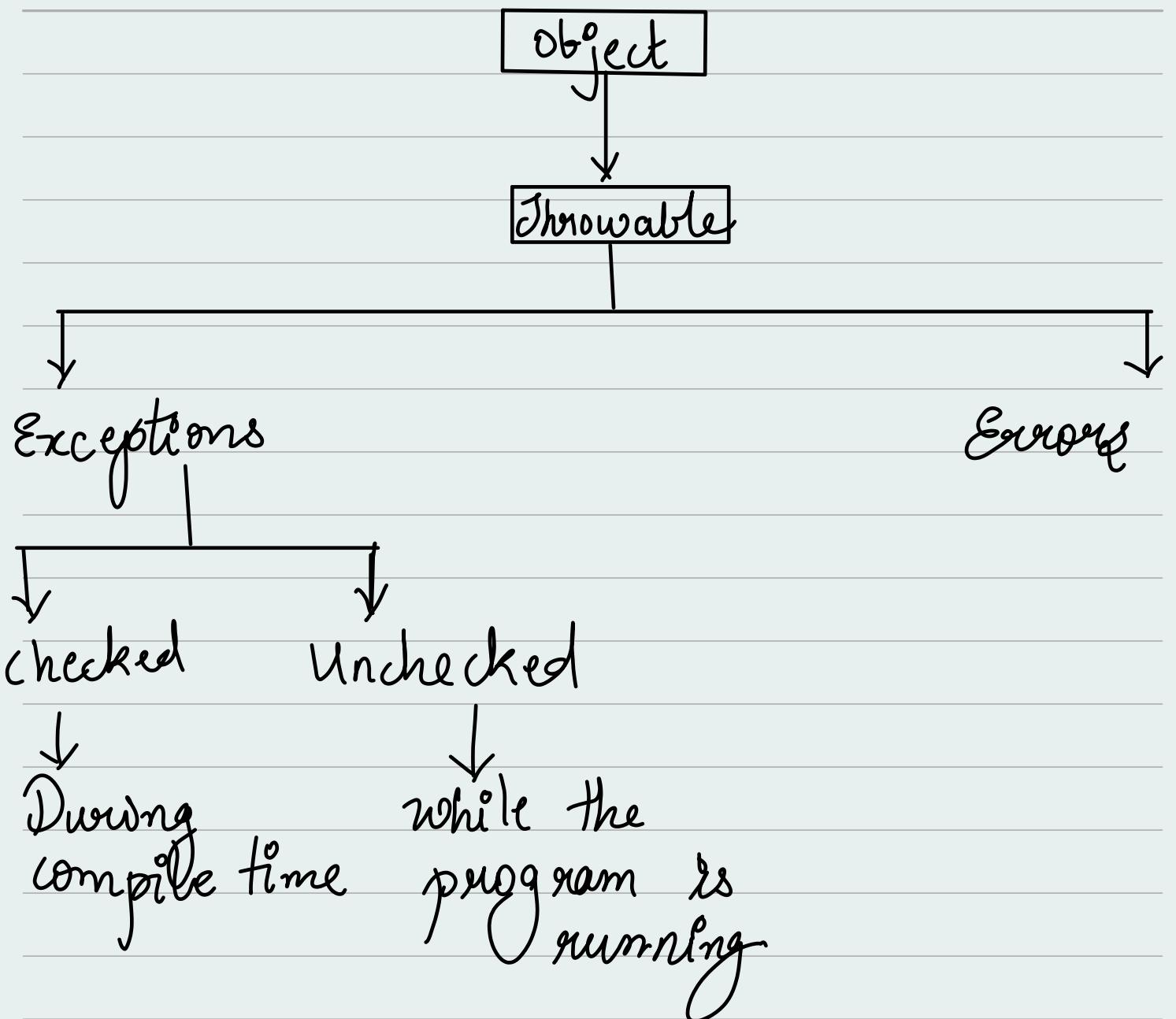
interface operation {

int operation ( int a , int b ) ;

}

Error : lack of resources , stack overflow ,  
non recoverable .

Exception : prevents normal flow of  
the program . Like dividing something  
by 0 . Null poi - cep - on . etc



Exception handling :-

Public class main {

public static void main (String [ ] args)

int a = 5 ;

int b = 0 ,

try {

    int c = a / b;

} catch (ArithmaticException e) {

    System.out.println(e.getMessage());

} finally {

    System.out.println("this will always  
execute");

}

} try {

    divide(a, b);

} static int divide (int a, int b) {

    if (b == 0) {

        throw new ArithmaticException();

}

    return a / b;

}

$t \Rightarrow$  thread

$t_2$

$t_3$

$t_4$

\* Collection Framework

not synchronized  
in ArrayList

ArrayList

$t_5$

Vector

→ It is little bit different

(Multiple threads can access the same

object in the ArrayList)

(t<sub>1</sub>)

waiting (t<sub>2</sub>)

only one thread can access at one time) at vector. So it is synchronized.

\* Enums in Java

→ Abstract are not allowed:

void display(); X

void display () {  
 // body  
}

