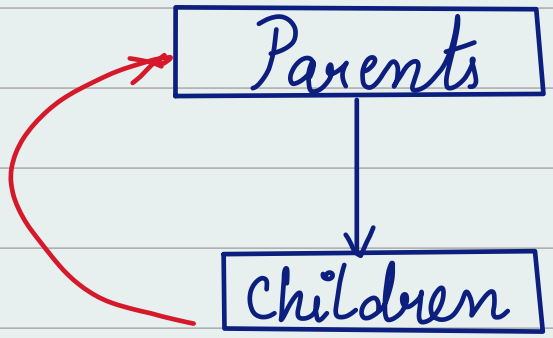
* Use Command + N (to create a constructor)

→ 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 the objects

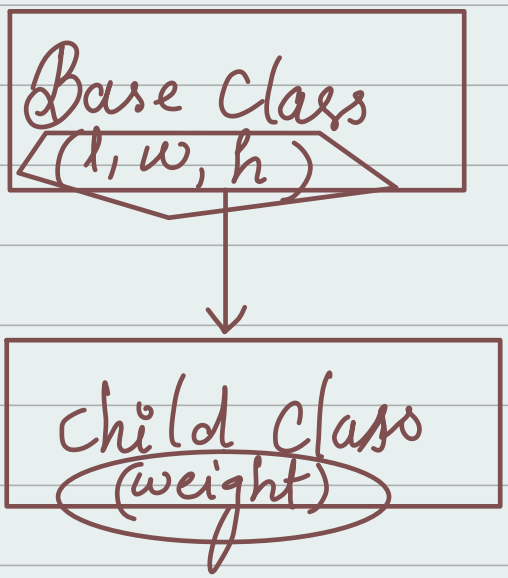
---

i) Inheritance :-



[money, human traits, values]

In OOP



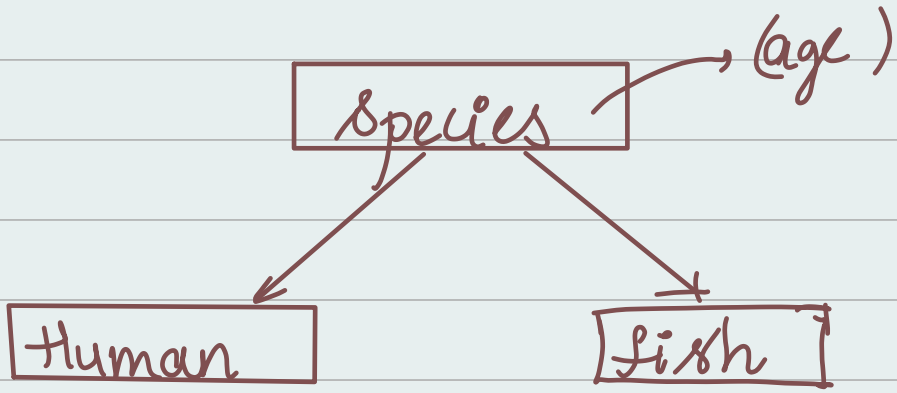
Child class is inheriting properties from base class.

class Child extends Base {

int weight;

}

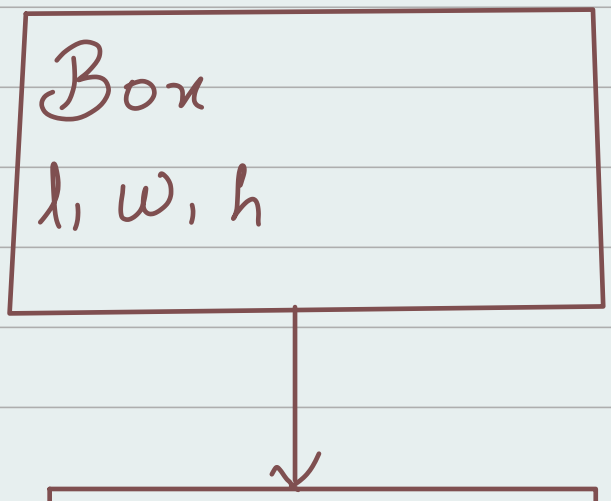
```
Child child =
    new Child(),
child.l();
child.w();
child.h();
```



Human Rohit = new Human();

Rohit.age

Initialize parent class variable also



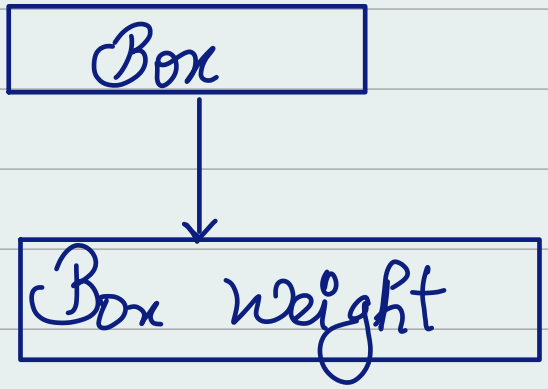
Box weight
weight

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

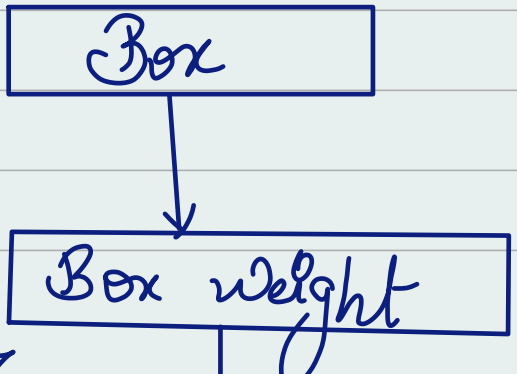
→ But child class cares about what parent class contains.

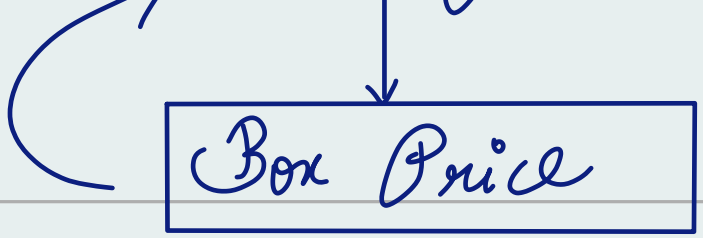
Types of Inheritance

1) Single inheritance
* One class extends another class



2) Multilevel Inheritance :-

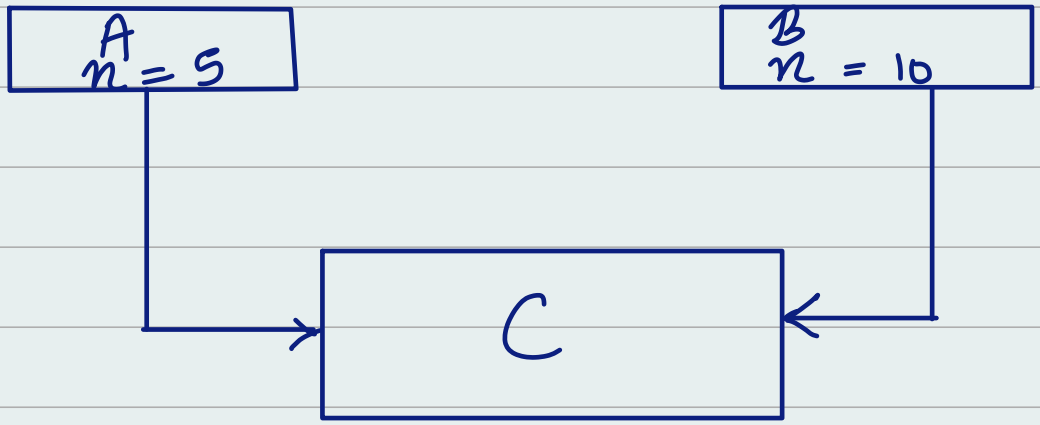




By using super() it will call the above function ex' super() of BoxPrice is BoxWeight and super() of BoxWeight is Box.

3) Multiple Inheritance :- We will do this in interfaces

One class extending more than one classes



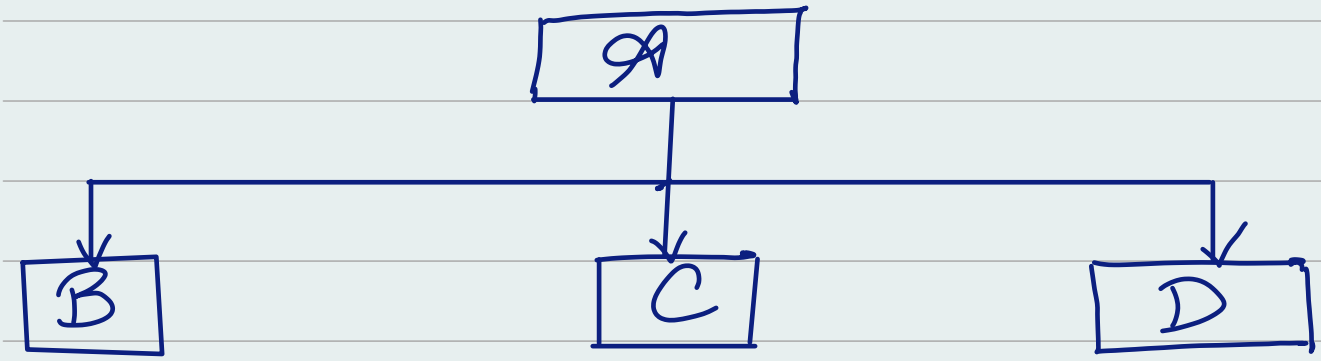
C object = new C();

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 n values

4) Hierarchial Inheritance :-

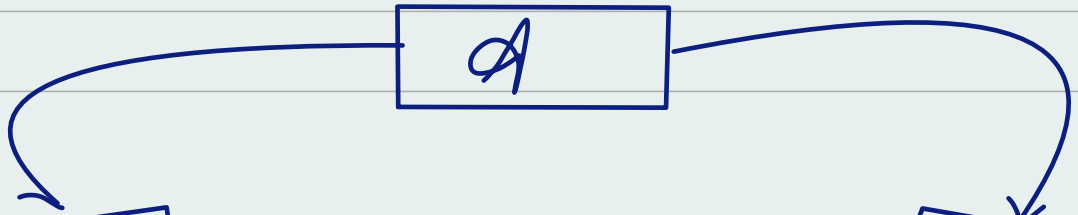
One class is inherited by many classes.

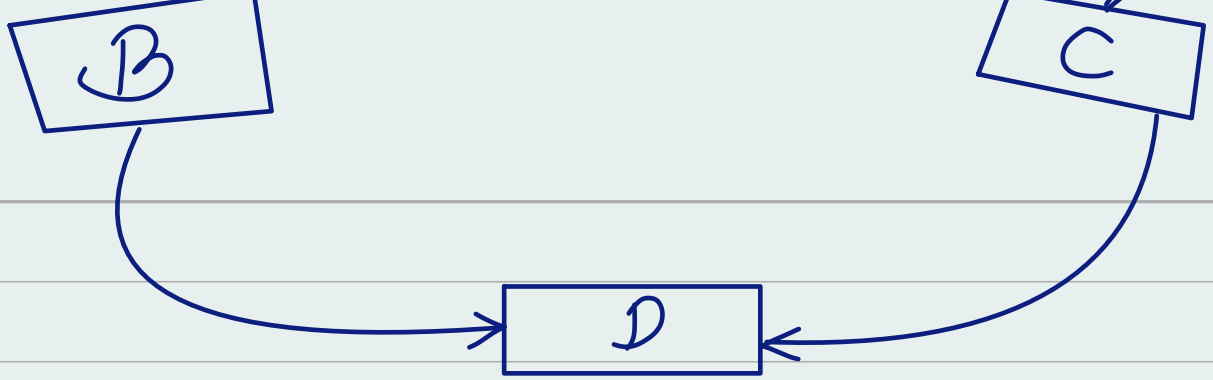


5) Hybrid Inheritance :-

Combination of single and multiple inheritance

(Not in Java check in Interface lecture)

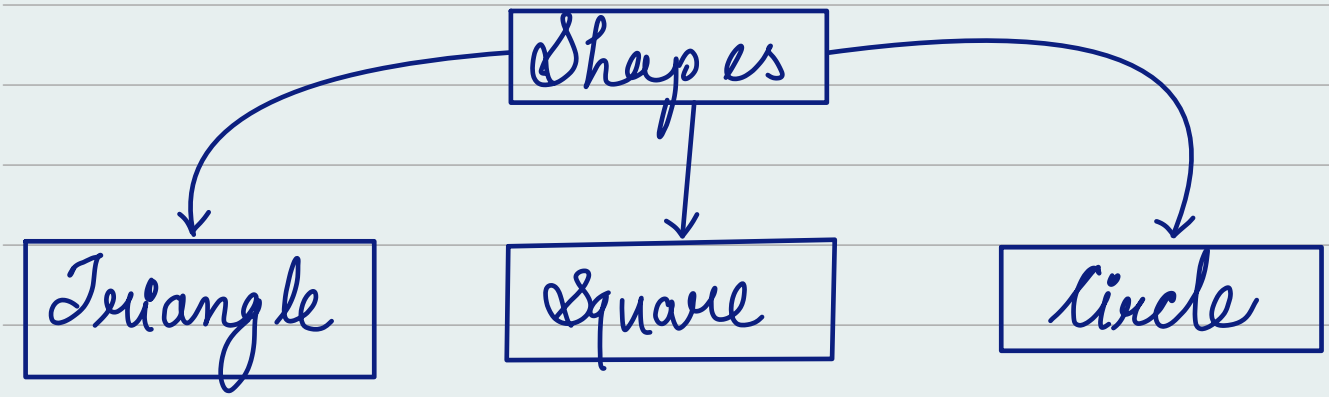




→ A class cannot be it Superclass.

* Polymorphism : Poly means many morphism are ways to represent.

→ Shapes :-



Types of Polymorphism :-

1) Compile time / static polymorphism :-

Acheived 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 Jarea determine this → Dynamic Method Dispatch.
It determines by

→ So the name give Dynamic Polymorphism.

→ You cannot override a final method

@ Override // final X

Ex. X final void area(X sout(" ")
}

→ 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.

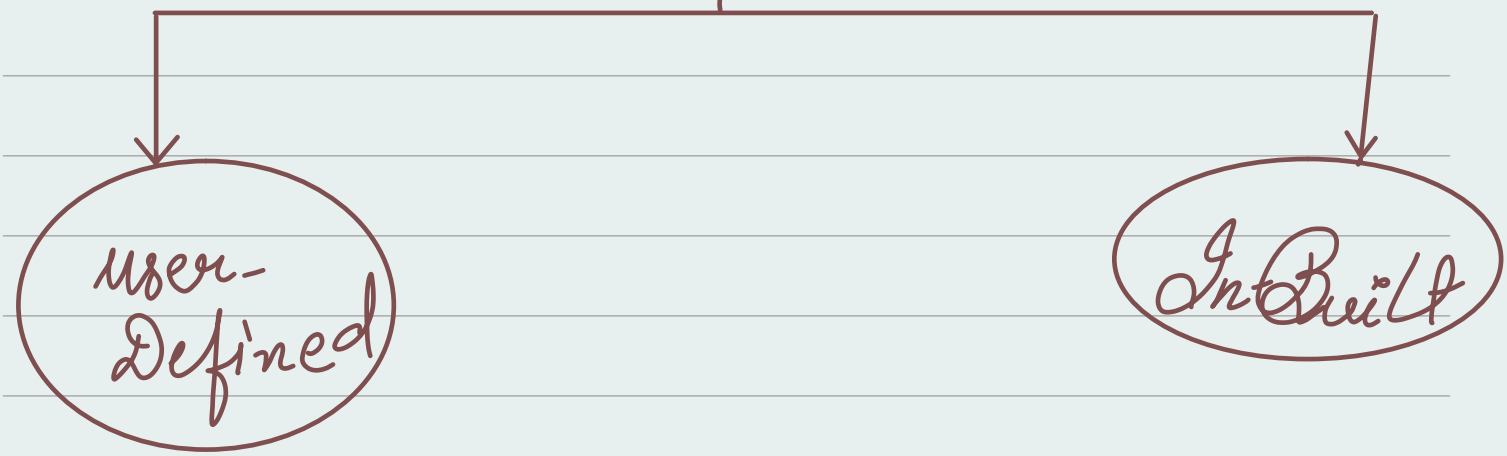
Abstraction :-

* 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

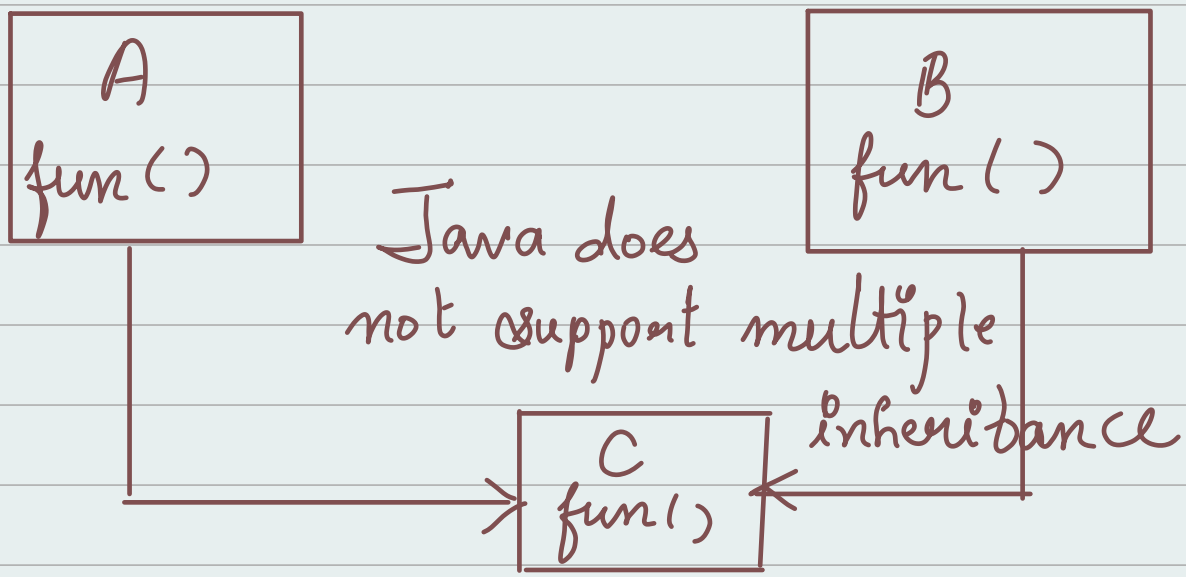
packages



In-Built

- Language (Automatically imported)
- i/o (read file, optimized, buffer)
- util (data structure)
- applet (Spring framework learn!)
- awt
- net

Abstract Classes, Interfaces, Annotation



parent class

abstract methods

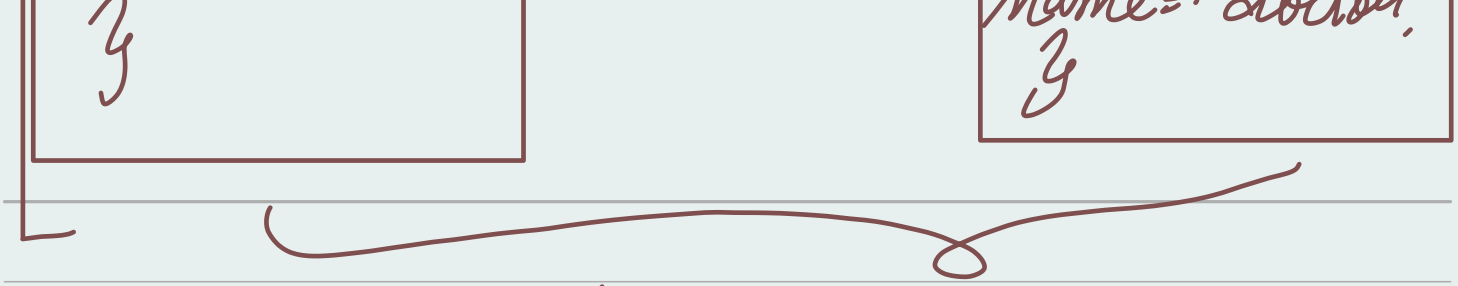
Career (name)

child class

child class

Career(name) {
name = "Coder";

Career(name){



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

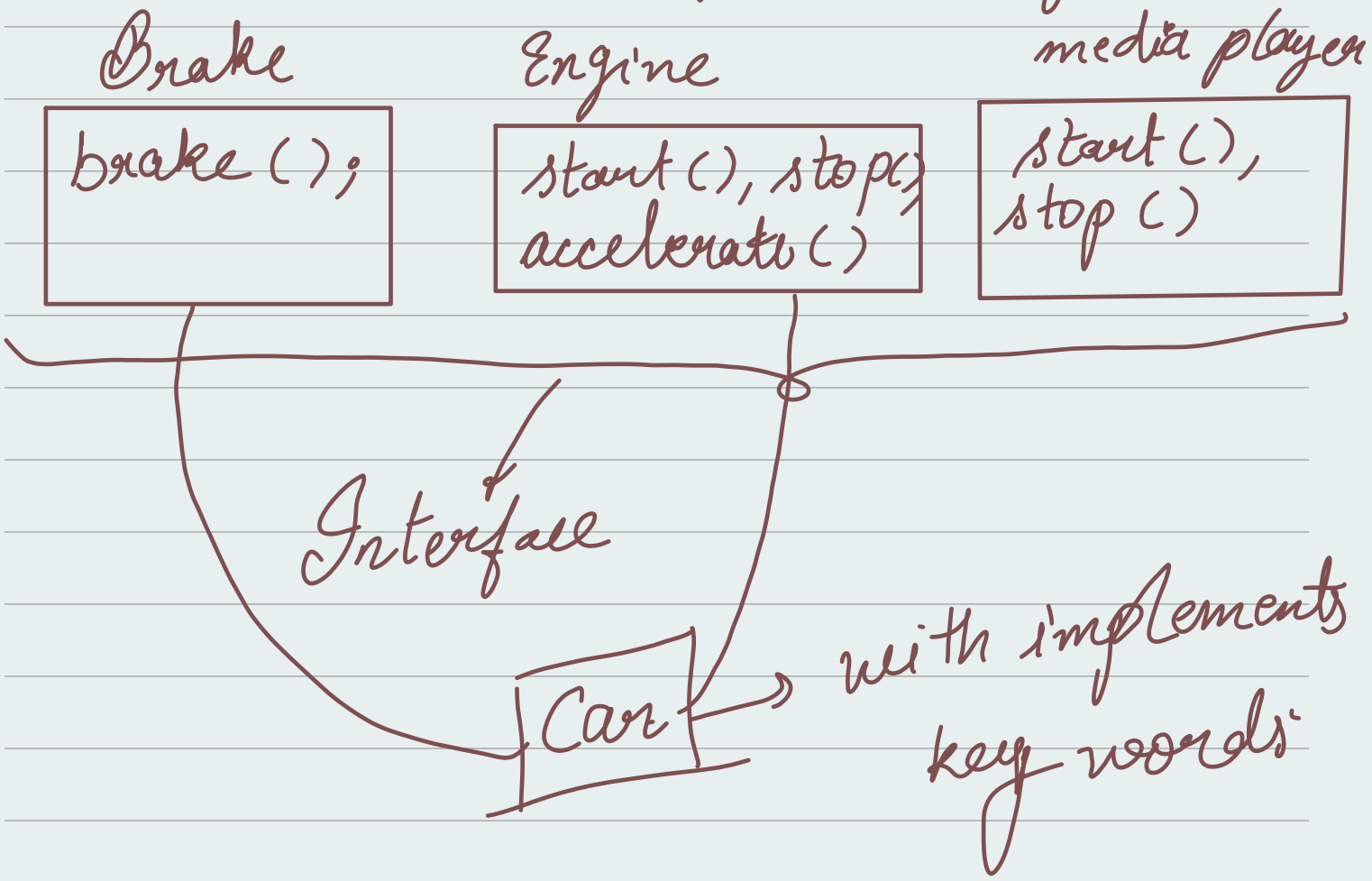
Parent Class will provide empty methods and child class will override the methods.

abstract void career (String name);

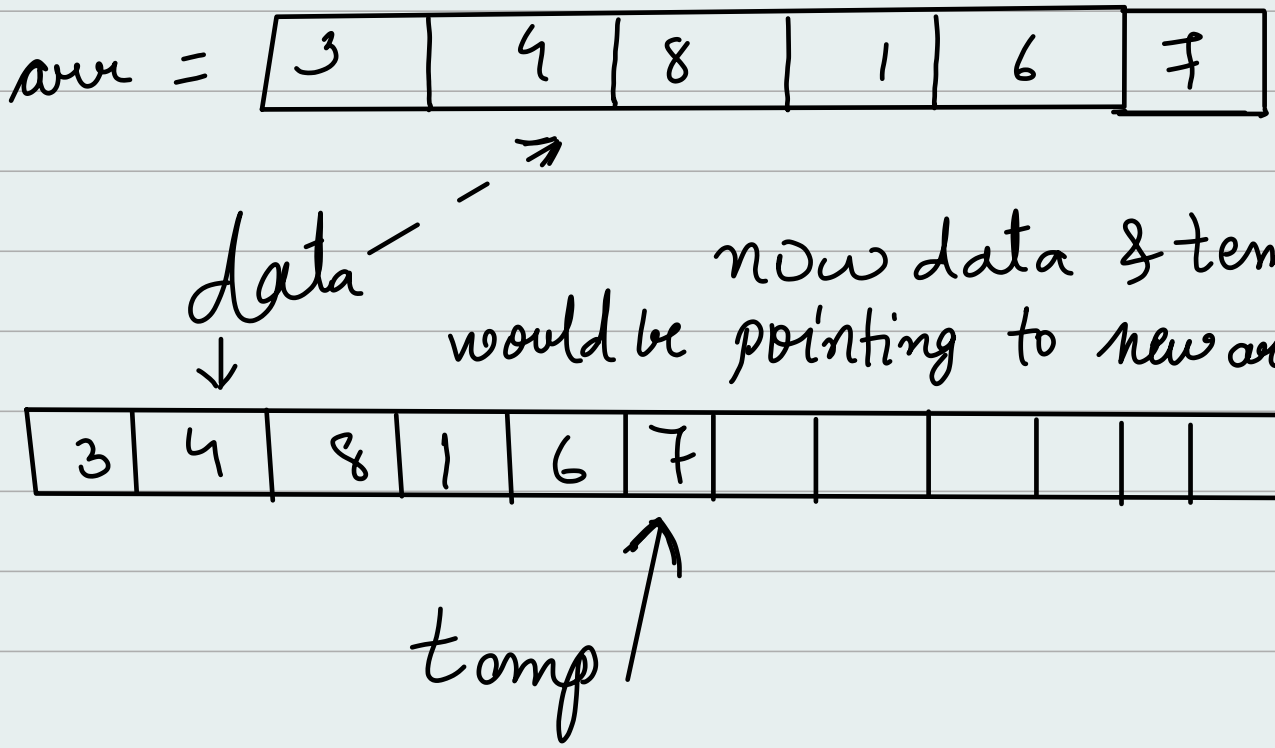
If any class that contains one or more abstract methods, must also be declared as abstract.

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

→ Implement multiple interfaces



* Custom Arraylist



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));
    }
    operation sum = (a, b) -> a+b;
    operation prod = (a, b) -> a*b,
    operation sub = (a, b) -> a-b;
    LambdaFunctions myCalculator = new
```

LambdaFunctions();

&Out(myCalculator.operate(a:5, b:3, &sum));

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

&out(myCalculator.operate(a:5, b:3, &sub));

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

return op.operations(a, b);

}

}

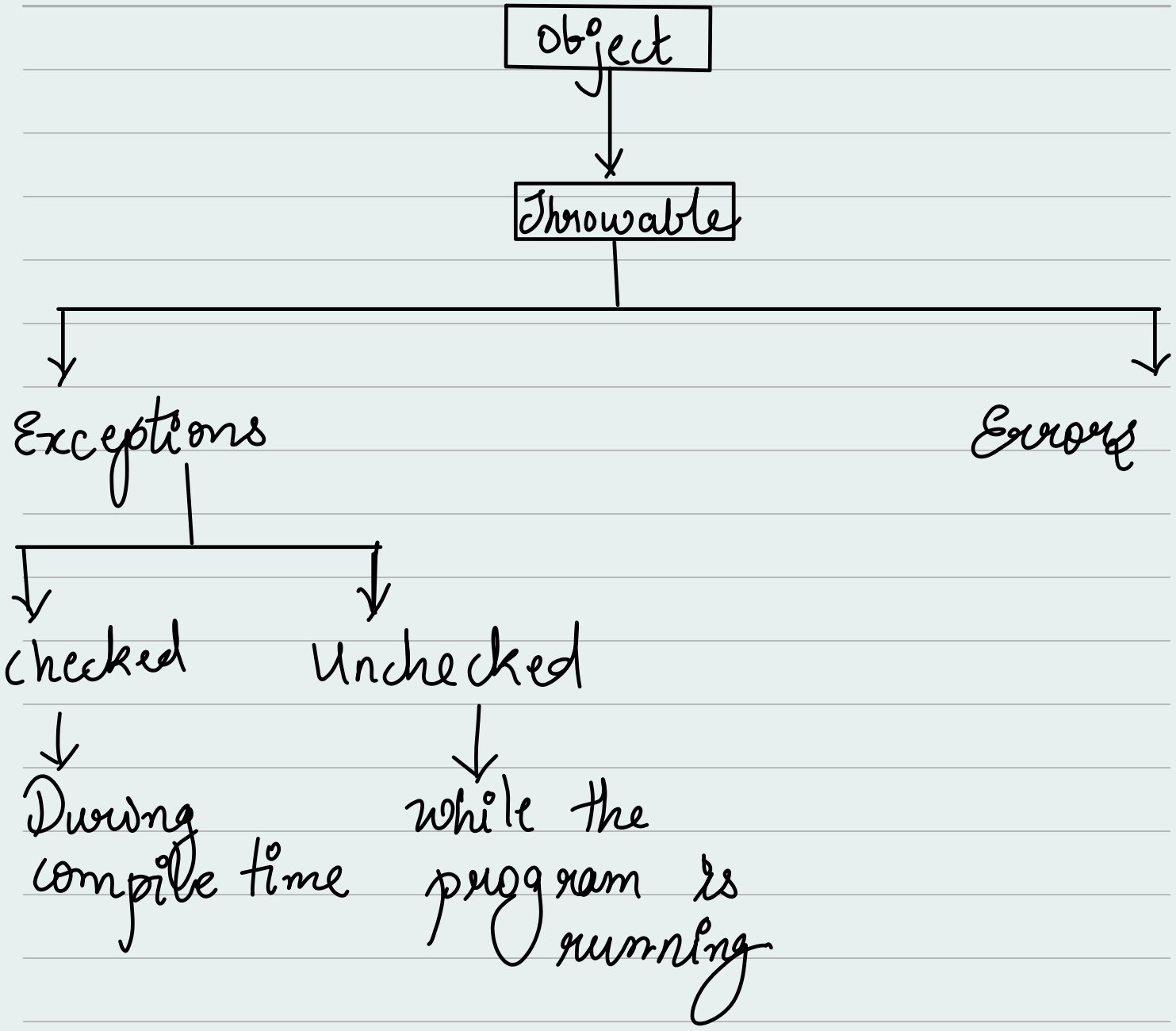
interface operation {

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. NullPointerException etc



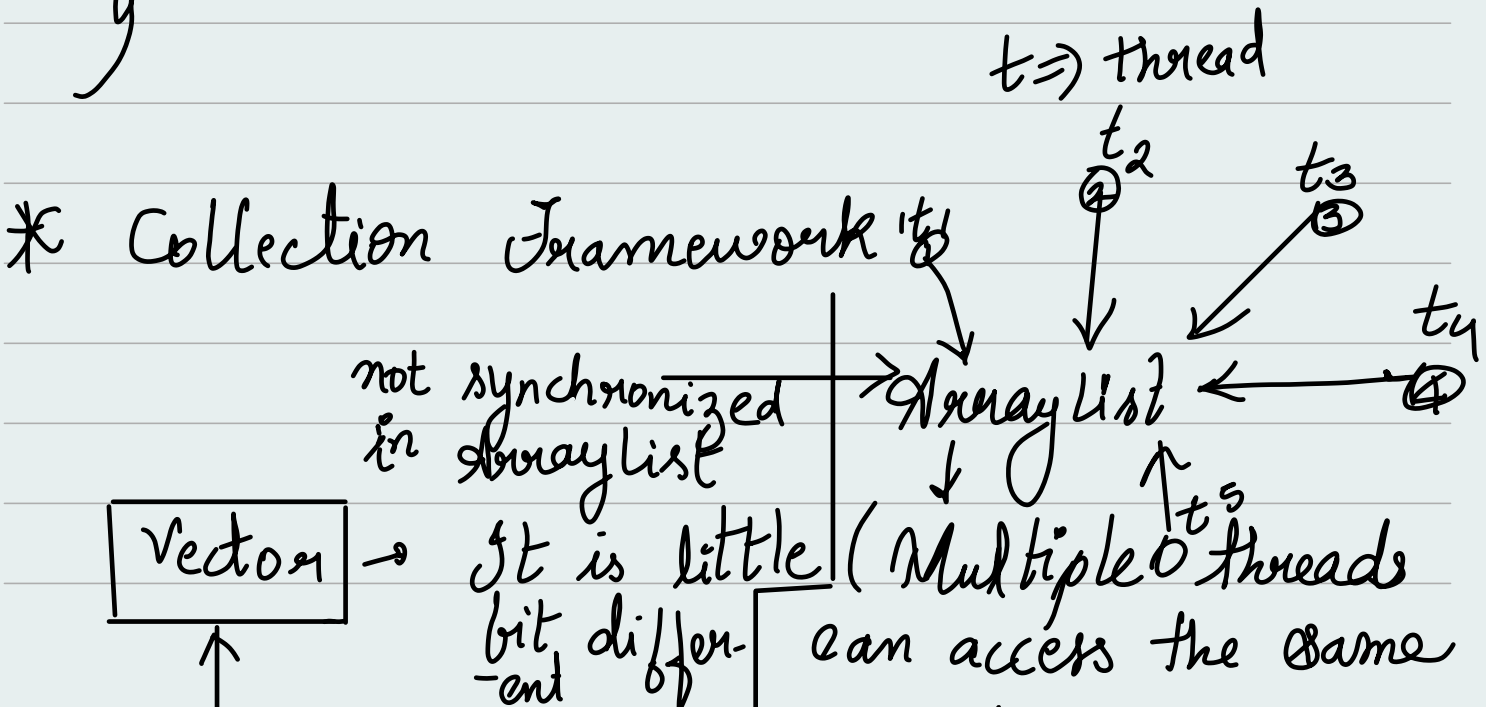
Exception Handling :-

```
public class main {

    public static void main (String[] args){

        int a = 5;
        int b = 0;
```

```
try {
    int c = a/b;
} catch (Arithmatic Exception 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;
}
}
```



object in the ArrayList)

(t1)

waiting (t2)

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
}
```