
Introduction To OOPs
Encapsulation
Abstraction
Polymorphism
OOPs
Inheritance
Object
Class
OBJECT ORIENTED
PROGRAMMING
algorithms365
Performance with purpose

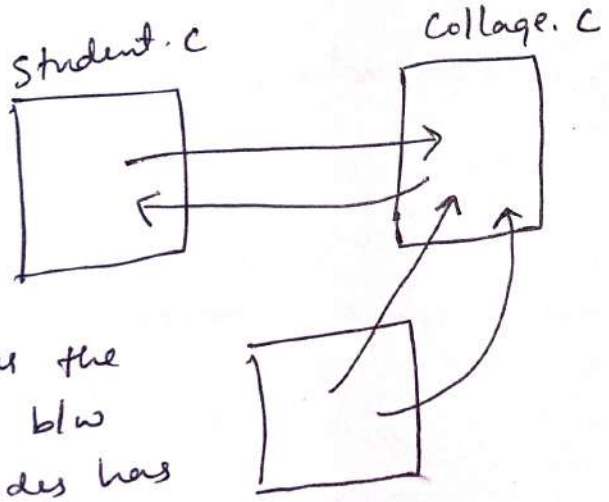
Day 22 10/9/25 wednesday

object oriented programming

1. Why we need OOP?
2. Real world examples
3. Key Concepts
   - class
   - object
   - Encapsulation / Bundling of data and Methods
   - Constructors
   - Access Modifiers
     - public
     - private
     - protected

1. Why we need object oriented program?

⇒



* After years the connection b/w different codes has become more.
* we struggle to understand the code and difficult to refactor (or) change

to solve these problem's their we get the concept of OOP

⇒ Real world objects (or) Entities (2)

Eg: Library management system

classes
- Books
- students
- staff
- Orders

→ Entities / real world objects / Actors

operations / Actions

functions (or) methods
- login , Log out → from student point
- checkout , check in → from student point
- Add books → library person point
- Requests → library person point
- Register → library person point

(3) what is class ?

Student

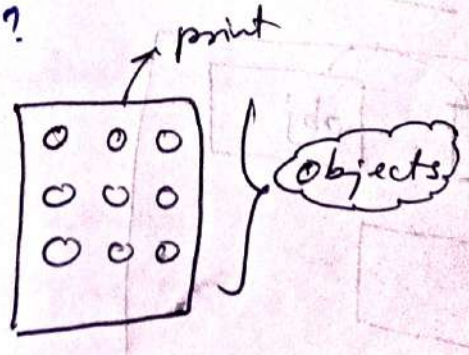
Properties (or) Attributes
- Name
- Id
- Age

* A class is a blue print (or) a template for creating objects
* No memory significance

operations
- Register ()
- Delete ()
- checkout ()
- login ()

→ Methods (or) Functions

what is object ?



Student obj = new Student();

variable or object name

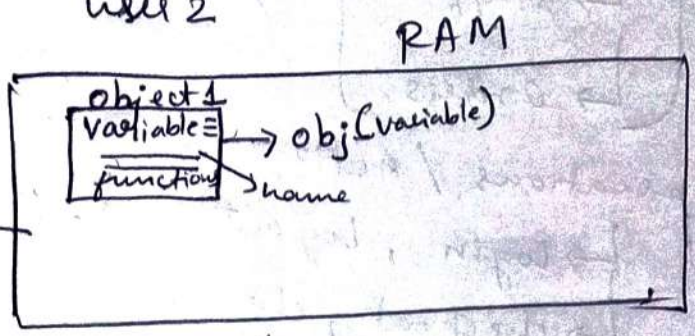
class

type = Student

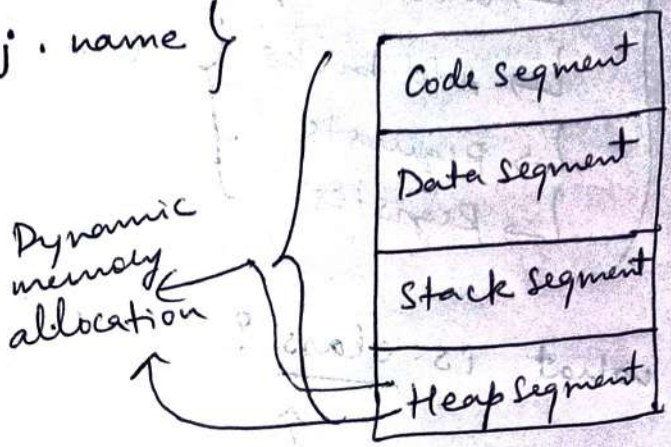
Instance

user 1

user 2



Access → { obj . name }

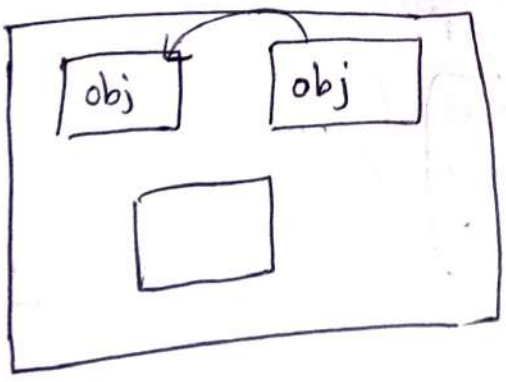


cleaning

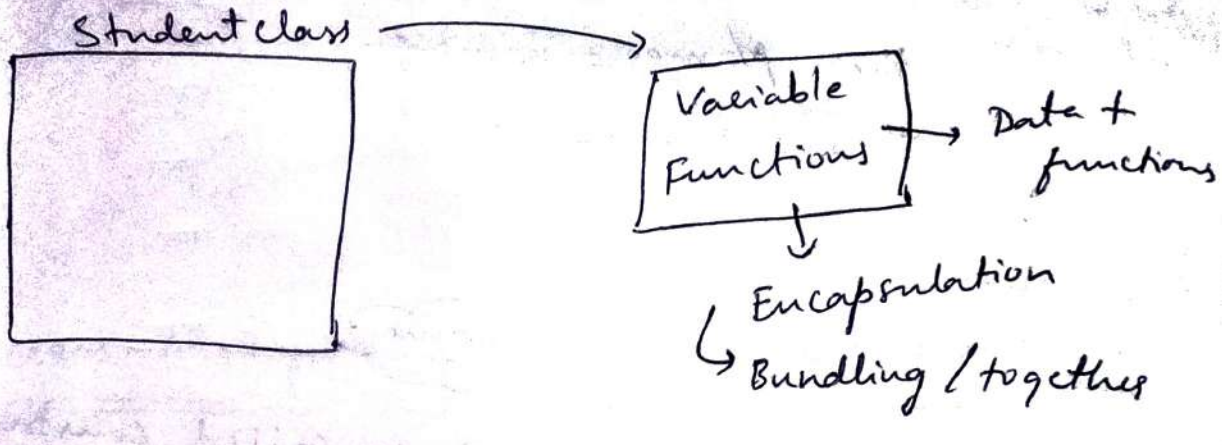
* Garbag collector
* It is a process which runs every 10 sec once
* It will clean the non used objects in the code

class_name obj_name = new class_name();

Multiple class == Module / Library / Component



Encapsulation / Bundling of data & Methods



Constructor

```
class Student
{
    Public string Name ;
    Public int Age ;

    Public string getName ( )
    {
        return Name ;
    }

    Public int getAge ( )
    {
        return Age ;
    }

    Public Student ( )
    {
        this . Name = "" ;
        this . Age = 0 ;
    }
```

this.Name;

this.Age;

variables / attributes / properties

Function or Method

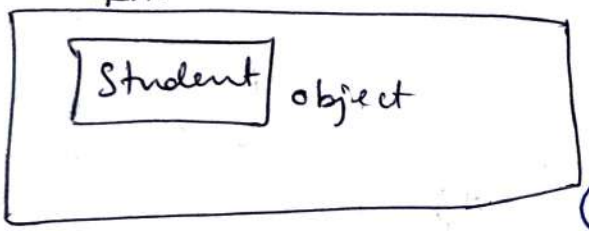
Constructor

```
Public Student (String name, int age)
{
    this.Name = name;
    this.Age = age;
}
```

Constructor

class Name == Constructor name

* The one which construct is called constructor
* this is constructing an object



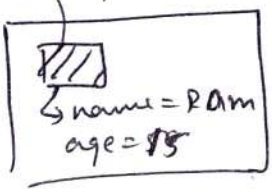
* if you call the Constructor

Constructor
* object create
* Initialize

Calling constructor

⇒ Student new_student = new Student ("Ram", 15);

new ⇒ creating the obj.



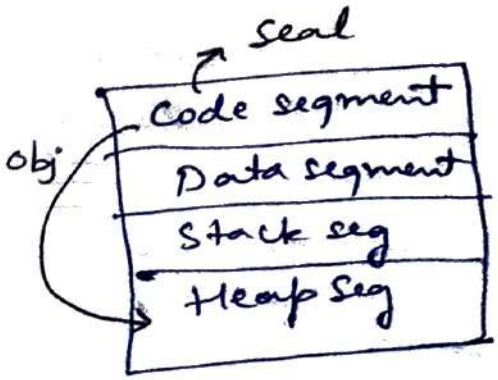
Constructor

- no parameter
- Parameter

* Method (or) function
* One name with different parameters is called Polymorphism ***

High level

Intermidiate level → Binary lang



Access Modifiers
- public
- private
- protected

Private

ISRO campus

* There are some restriction

* can access only in that class

Public

Park, Lake.

* Anybody can access

Protected

* More restriction then Private

* Allow access within the class and its derived class (subclasses)

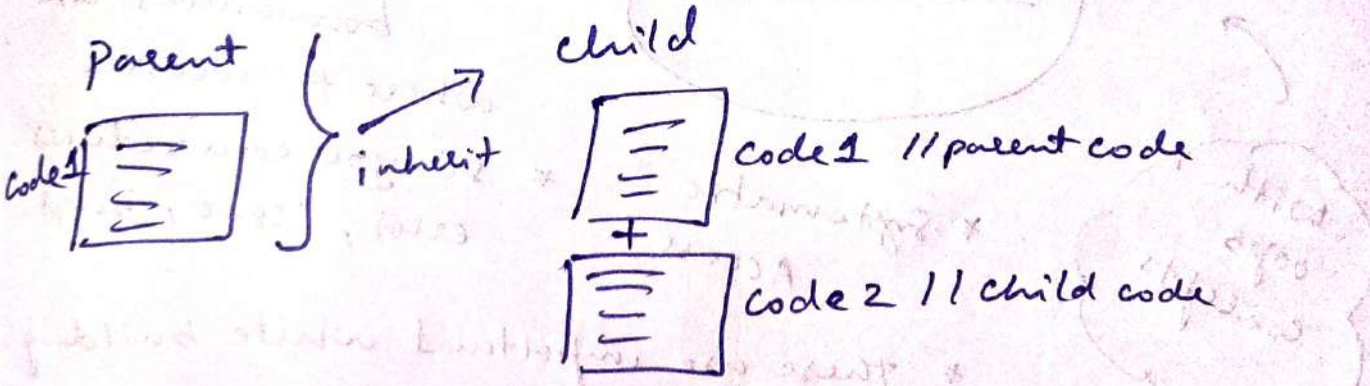
Inheritence

Library

Component/Module

* [C] [class] [class]

* If the class is protected you access in that particular Module or library



Default Modifier

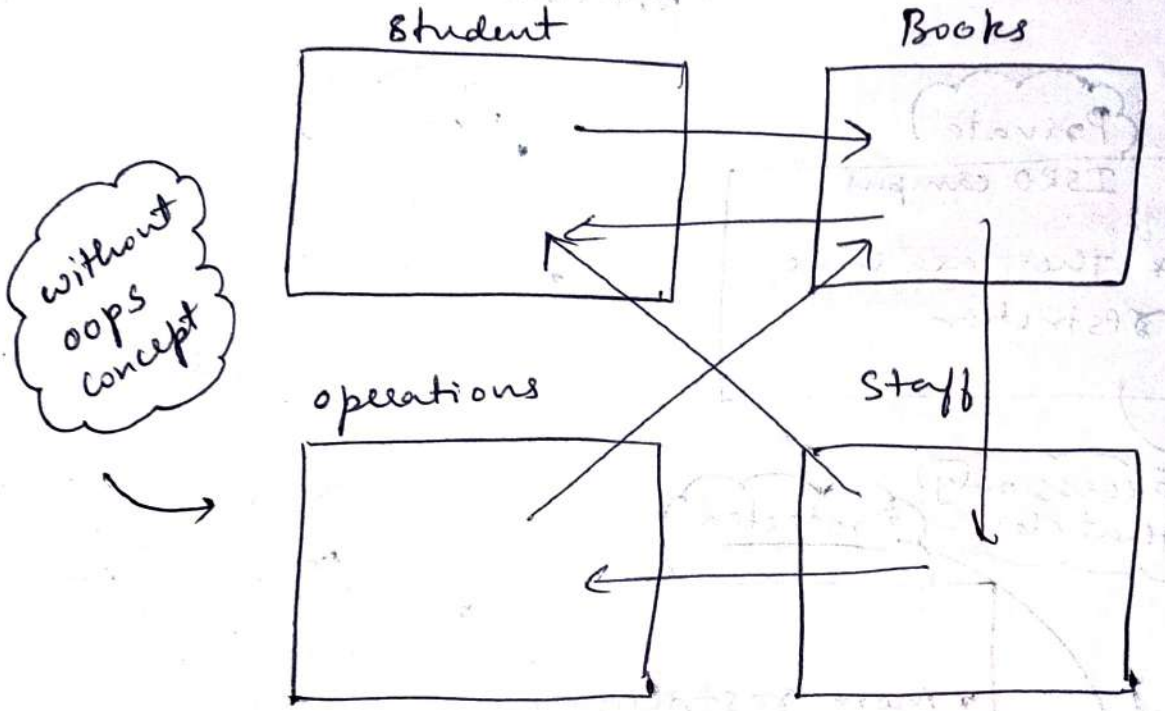
class Student { }

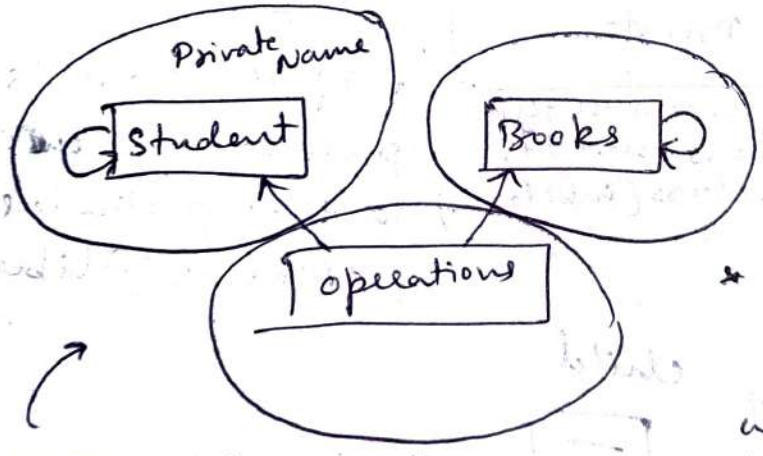
→ C++ (private)

→ Java (Package private)

Package

why these complications





with oops concepts

* Systematic Access

* you can create boundaries like where to accessies

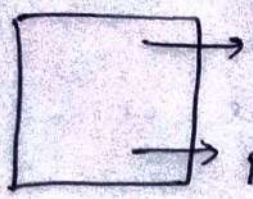
* so you can reduces error, issue, bugs

* These are important while building high security software.

Eg: Bank details

* staff can access only < balance / Name

→ Declaration class
→ Defining. Variables, Methods
→ creating objects



Name (Access)

new_student.name = "Anu";

compile time errors.

* we cannot assign directly to private variable

get → Read.

Day 23 Sept - 11 - 2025 Thursday.

OOPS Workshop in Java

// student → Custume type

// int → Primitive type

Deadcode

Same function Name but different parameters called "polymorphisms"

Day 24 Sept - 12 - 2025 Friday

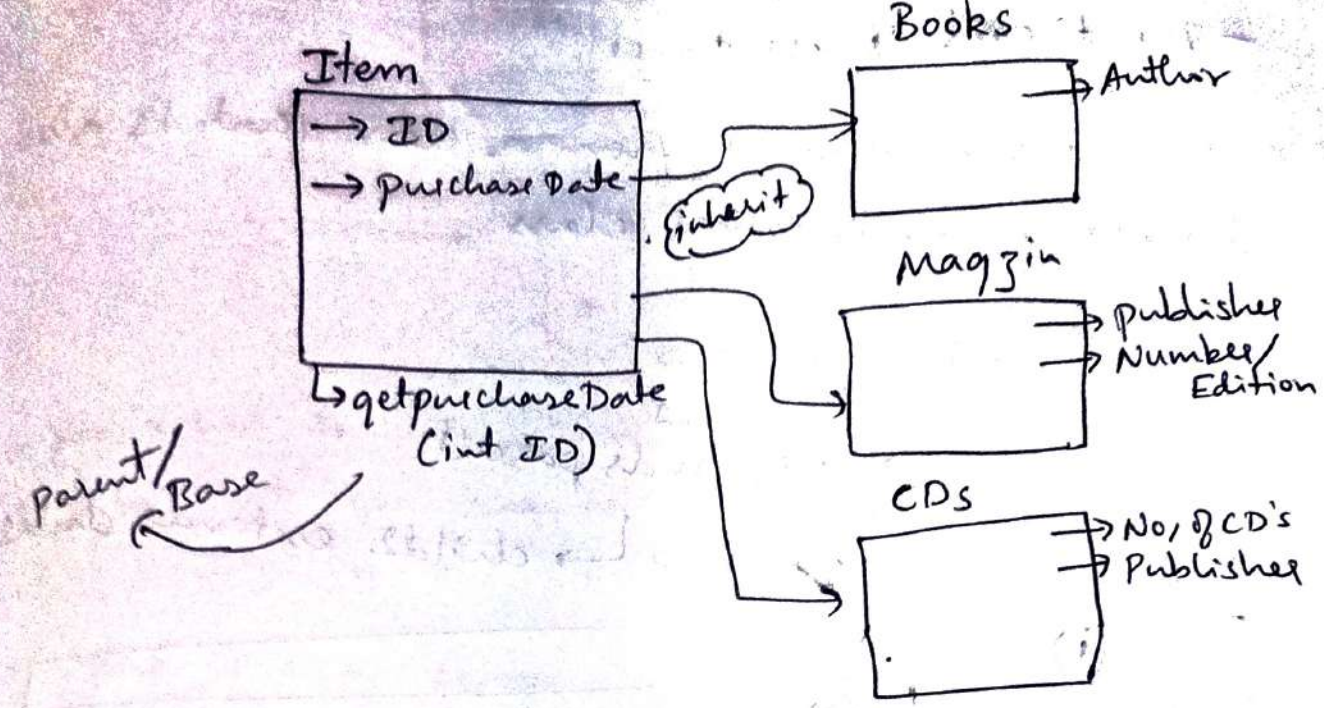
OOPS Advanced Concepts

(uses extends Keyword)

1. Inheritance
   - parent class, child class
   - Access modifiers
     - Public (can be accessed by any class)
     - protected (Accessible within the package and subclass)
     - private (only accessible with in the class)
   - Method overloading
   - Multilevel inheritance
2. Abstract class
3. Interface.

① Inheritance — Parent class child class

It allows one class (child) to inherit the properties and behaviours of another class (parent)



- Access Modifiers
  - Public
  - Private
  - Protected

without inheritance also we can access

Public — can be accessed anywhere in the class (Base class as well as child class)

Private — can be accessed only in the Mentioned class (can not be accessed by any other classes)

Protected — can be accessed only in the derived class

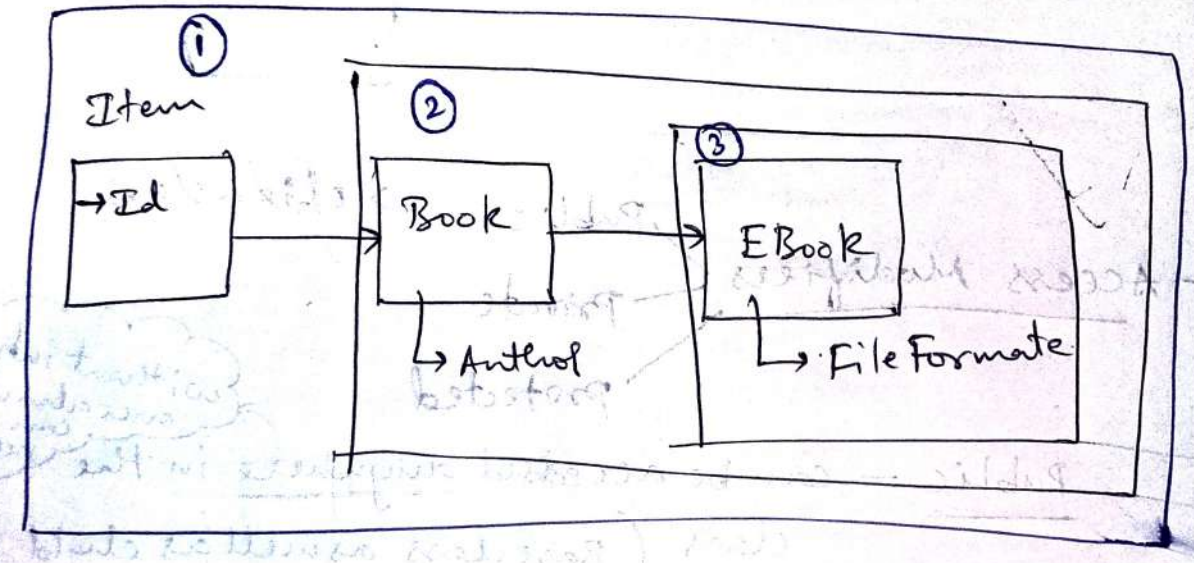
- Method overloading

* It allows multiple methods with the same name with different parameters.

- Multilevel Inheritance.

* A class is derived from a class that is also derived from another class.

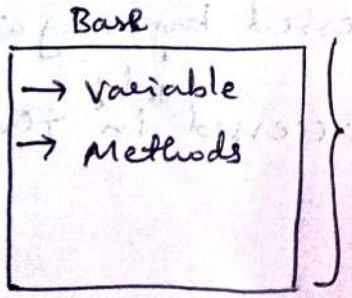
> Parent class
> ↳ child1 extends parent
> ↳ child2 extends child1



(2) Abstract class

Specification

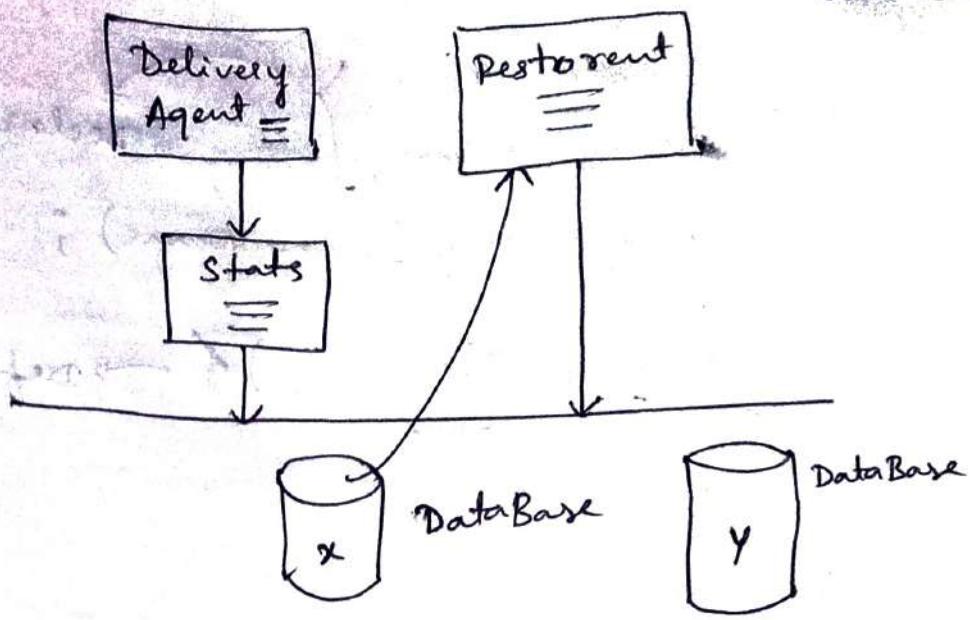
* Their was a need base class



* only Declaration ✓
* no Defination ✗

1 Lakh line

DB interaction Code 20k



* you want to change X DataBase to Y DataBase so that you has to change those 20K line of Code in every classes

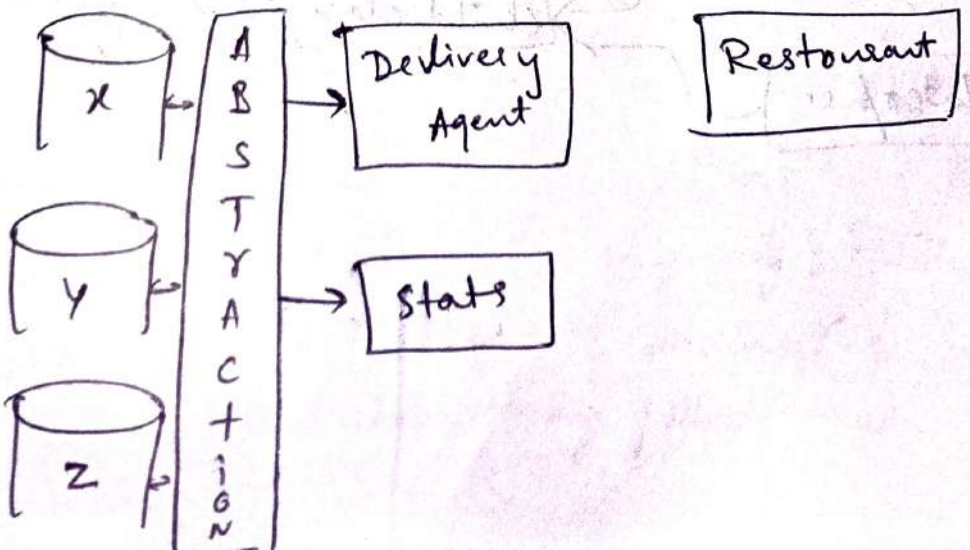
* this is very complex to do

Eg: if you want to change the bike tire with other company tire

you should modify you whole bike which is not possible

→ where the Abstract enters here. to reduce this complexity.

Abstract and interfaces both will help.



⇒ abstract class className

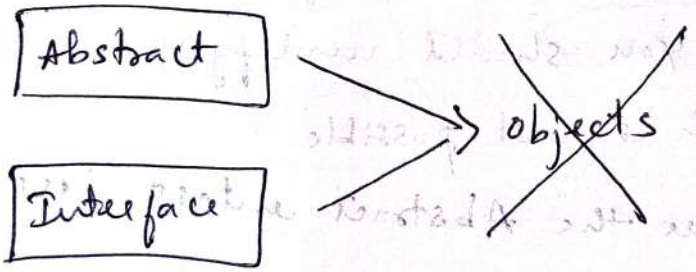
```
{
    Private ID;
    Public Name;        } → Declaration
    Void Search (string Name);
    .
    Void printDetails ( ) { } → Defined
                ═══
    }
}
```

Interface → No Define

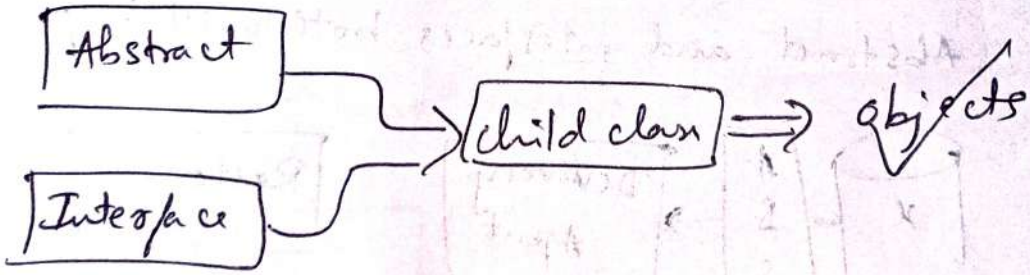
***

* you cannot object for both Interface and abstract class.
* you can only create the object using the children of abstract and Interface class

*



*



⇒ Interface DAL {

Data Access Layer

```
    void connect();
    Void getTable();
    void insert();
}

Sqllite extends DAL
{
    ==
    ==
    <=
}
```

methods. implement here

Abstract class :

It is like a partially build library : it has some concrete things (like walls) but also some incomplete areas (like shelves) for future development

you can't create an instance of an abstract class directly but can extend it to make complete classes.

```
// Abstract method (to be implemented by child classes)
```

use @Override and implement that.

Interfaces :

An Interface is like a library policy : It defines the rules that all types of items (DVDs magazines) must follow; but how they follow these rules is up to them

class can implement multiple interfaces to follow multiple policies.

Day 25 Sept - 13 - 2025 Saturday

OOPs Advanced Workshop in Java

LibraryItem → Base class

- Magzine } child class of LibraryItem
- Book }
  - EBook → child class of Book

Concepts covered.

- Inheritence
  - * parent class, child class
  - * Access Modifiers
  - * Method overloading
  - * Multilevel Inheritence.

I made LibraryItem as a abstract class

① * when you doen't want implement that method in every child class make them abstact (simply write in abstract class itself)

* when output varies for that abstact method then only write the code in derived class

(2)

* when you don't want to create an object of the class then make it as abstract

* you cannot create an object for abstract class (directly we cannot)
= you can create an object for only derived class (but indirectly we can)

Concepts Covered

LibraryItem → Abstract class.

CheckOut → Abstract Method

↳ Implemented in
= Book class
= Magzine class

we cannot create an object

Topic → Abstract

now Interface

* what are the Methods we define in interface class we should implement them in implements class

otherwise WARNING's will arise

> The type ImplementclassName must implement the inherited abstract method interfacename. Method()Name;

CERTIFICATE

OF ACHIEVEMENT

This certificate is proudly presented to

Ananya

Congratulations on successfully completing the course - Java OOPs in Kannada for the period of 17-09-2025 to 21-09-2025 and excelling in the test conducted by Algorithms365.

This achievement reflects your dedication and hard work.

Certificate Number: 11387217238379173356105 65

Verify at: skills.algorithms365.com/learn/certificate/11387217-238379

Mahesh Arali
Chief Executive Officer

*Issued on the basis of an online automated test

algorithms365
Performance with purpose

https://www.algorithms365.com/