OOPS Advanced Concepts:-

1. Inheritance
   - → Parent class, child class
   - → Access Modifiers
   - → Method over loading
   - → Multilevel inheritance
2. Abstract class
3. Interface

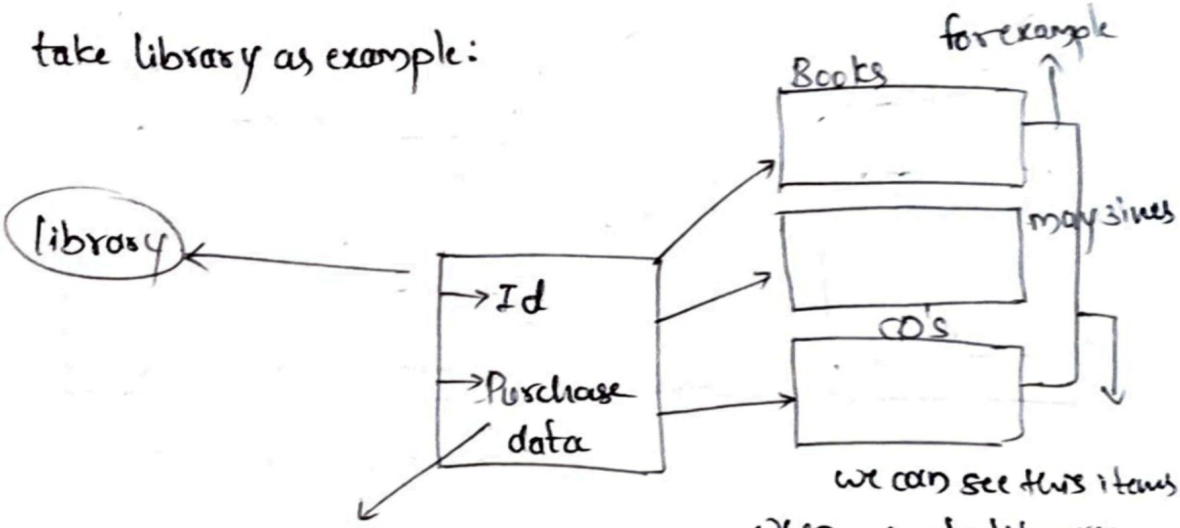
---

1. Inheritance

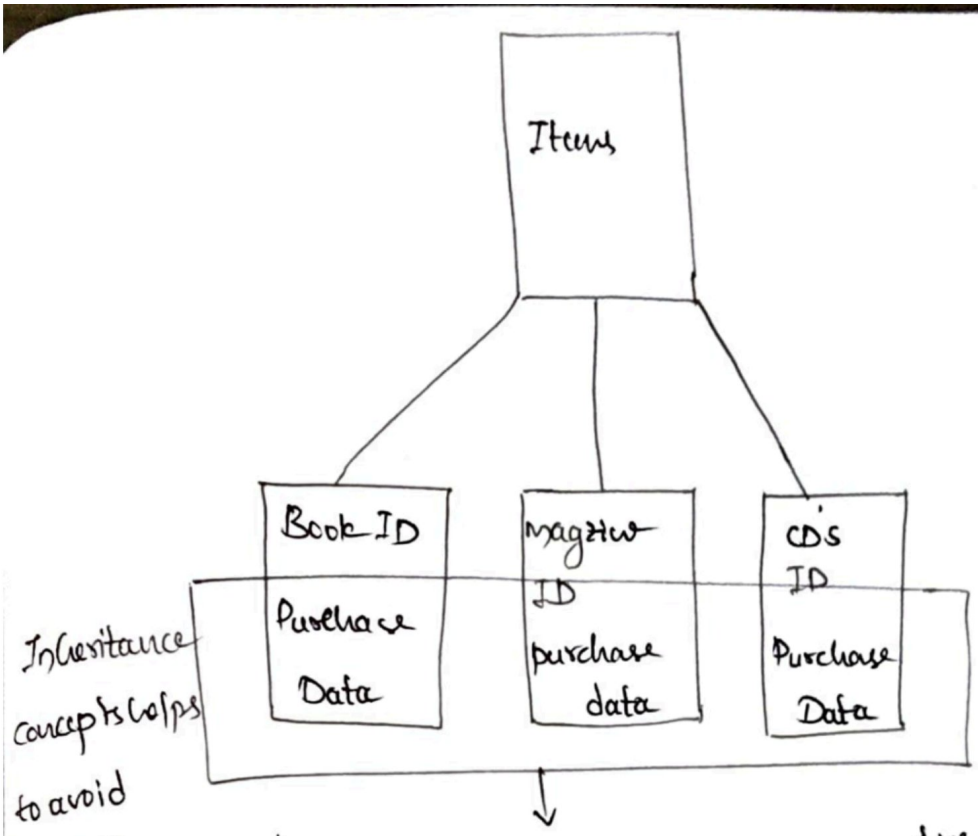
Inheritance allows our class(child) to inherit the properties and Behaviour of another class.

One of the Important advantage of inheritance is code Reusabili-ty.

lets take library as example:



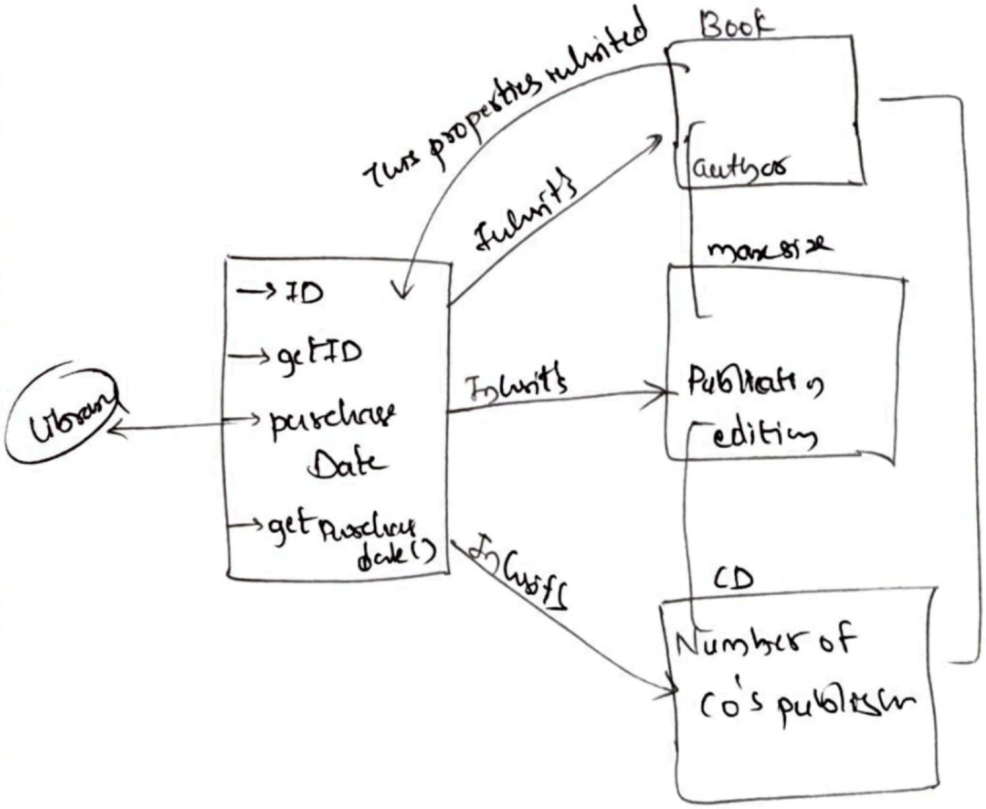
If Inheritance was not introduced then we have to declare the common properties again & again in the different item class like 'Book, magzine, CD'



Inheritance concepts helps to avoid writing duplicate code

So all these common properties are initialized in our parent class / Base class

So Inheritance looks like



All these Item have Inherited common properties from parent class, we can add extra properties to our child classes accordingly

Note: we can also override some methods as properties in child class (methods & properties which are in parent class)

OOPS Advanced Concepts

1. Inheritance: Code Reusability

Inheritance is the mechanism where one class (the **Child Class** or **Subclass**) acquires the properties and methods of another class (the **Parent Class** or **Superclass**).

- **Primary Benefit: Code Reusability**, reducing duplicate code.
- **Method Overriding:** A Child Class can provide its own implementation for a method already defined in its Parent Class.
- **Multilevel Inheritance:** A chain of inheritance (e.g., A → B → C).

Access Modifier	Visibility for Subclasses
public	Fully accessible
protected	Accessible to subclasses (the **most common** choice for methods and fields meant to be extended)
private	**Not accessible** to subclasses or other classes

2. Method Overloading: Compile-time Polymorphism

Method Overloading allows multiple methods within the **same class** to share the **same name**, provided they have a **different signature** (i.e., different number, type, or order of parameters).

- **Rule:** The return type or access modifier **cannot** be the only difference.

Java Code Example :

```
class Calculator {
    // 1. Overloading with different number of arguments
    public int add(int a, int b) {
        return a + b;
    }

    // 2. Overloading with different argument types (same number)
    public double add(double a, double b) {
        return a + b;
    }

    // 3. Overloading with different argument types (again)
    public String add(String s1, String s2) {
        return s1 + s2; // String concatenation
    }
}
```

3. Abstraction: Abstract Classes and Interfaces

Abstraction is the principle of showing only the essential information to the user and hiding the complex internal implementation details. It is achieved using Abstract Classes or Interfaces.

A. Abstract Class

- Cannot be instantiated (you cannot create an object directly).
- Can contain both **abstract methods** (declared without a body) and **concrete methods** (defined with a body).
- A subclass uses the extends keyword and **must** implement all abstract methods.

B. Interface

- Acts as a **contract** or "Rule Book."
- In modern Java, it contains method signatures (declarations) and constants.
- A class uses the implements keyword to adopt the contract and **must** provide a body for all methods.
- A class **can implement multiple interfaces**, solving the "multiple inheritance" problem.

Java Code Example (Inheritance & Abstraction)

This code demonstrates how a class can **extend an Abstract Class** and **implement an Interface** simultaneously.

```
// Abstract Class (Base Class)
abstract class LibraryItem {
    String title;
    String author;

    // Constructor
    LibraryItem(String title, String author) {
        this.title = title;
        this.author = author;
    }

    // Abstract method: MUST be implemented by subclasses
    abstract void displayInfo();

    // Concrete method (optional)
    void getDueDate() {
        System.out.println("Due in 14 days.");
    }
}
```

```java
// Interface (Contract)
interface Downloadable {
    // Interface method: MUST be implemented by classes that use this contract
    void download();
}

// 1. Book Class: Extends the Abstract Class
class Book extends LibraryItem {
    Book(String title, String author) {
        super(title, author);
    }

    // Implementation of the abstract method
    @Override
    void displayInfo() {
        System.out.println("Book: " + title + " by " + author);
    }
}
```

```java
// 2. EBook Class: Extends Abstract Class AND Implements Interface
class EBook extends LibraryItem implements Downloadable {
    EBook(String title, String author) {
        super(title, author);
    }

    // Implementation of the abstract method
    @Override
    void displayInfo() {
        System.out.println("EBook: " + title + " by " + author);
    }

    // Implementation of the interface method
    @Override
    public void download() {
        System.out.println("Downloading " + title + "...");
    }
}
```

```java
public class LibraryMain {
    public static void main(String[] args) {
        // We cannot create a new LibraryItem() directly

        Book book = new Book("The Alchemist", "Paulo Coelho");
        EBook ebook = new EBook("Digital Fortress", "Dan Brown");

        book.displayInfo(); // Output: Book: The Alchemist by Paulo Coelho
        book.getDueDate();  // Output: Due in 14 days.

        ebook.displayInfo(); // Output: EBook: Digital Fortress by Dan Brown
        ebook.download();    // Output: Downloading Digital Fortress...
    }
}
```