OOPS Advanced concepts

Day 28
07/08/2025

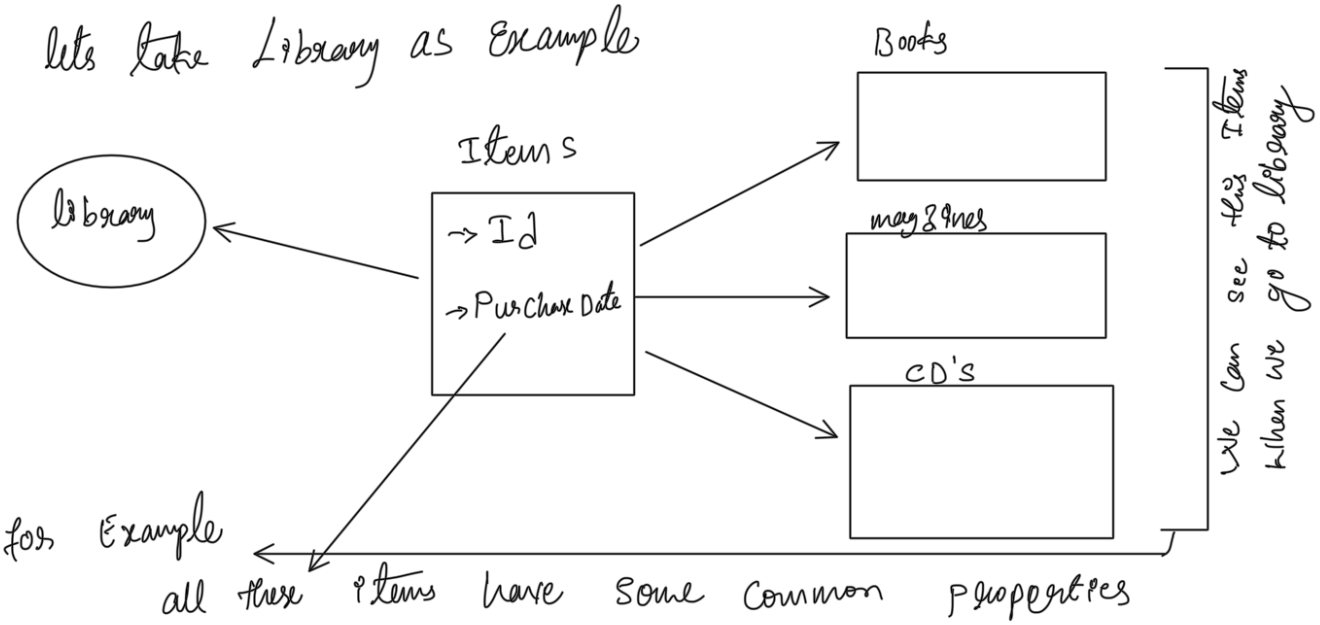
1. Inheritance
   - Parent class, child class
   - Access modifiers
   - Method overloading
   - Multilevel inheritance
2. Abstract class
3. Interface

(1) Inheritance

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

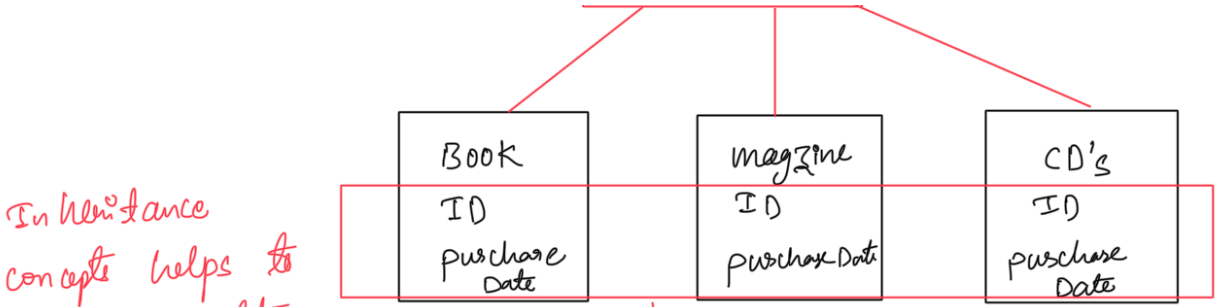
One of the Important Advantage of Inheritance is code Reusability.

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"

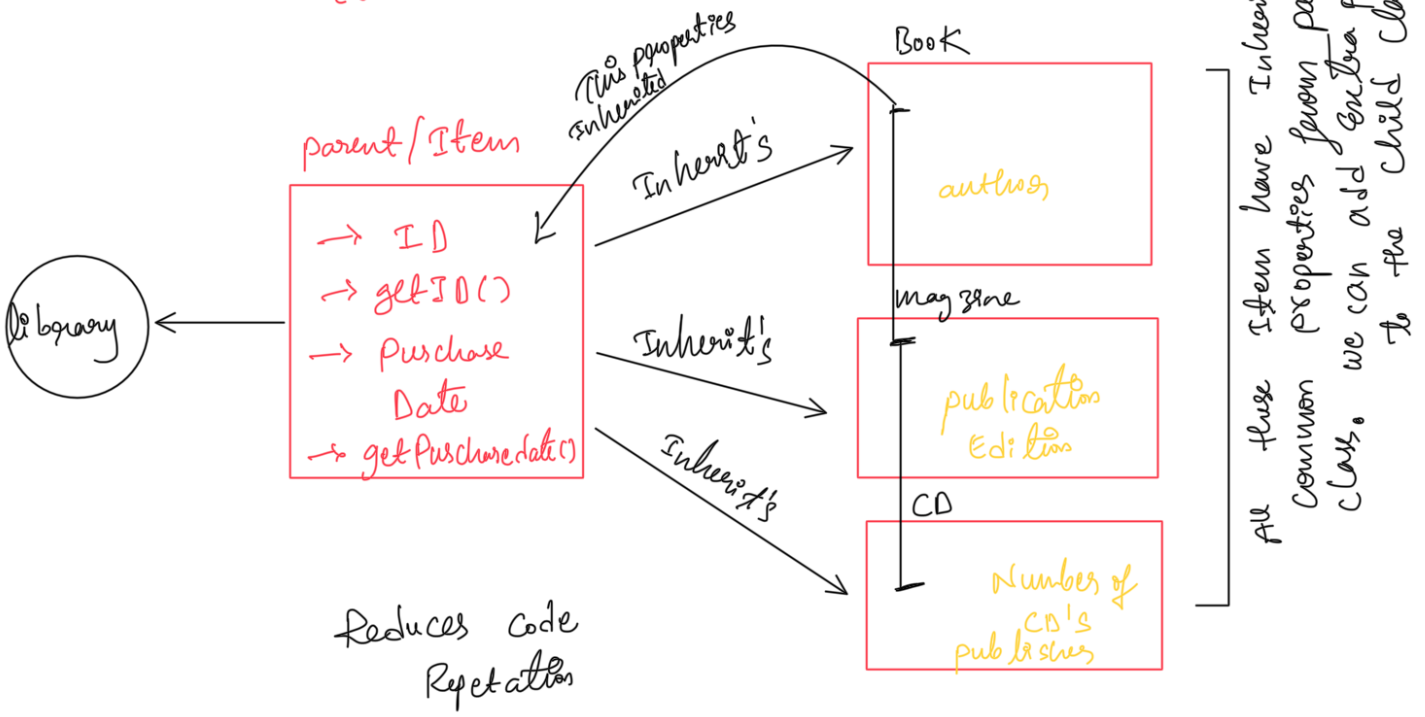
Items



Inheritance concepts helps to avoid wasting Duplicate code.

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

So Inheritance looks like

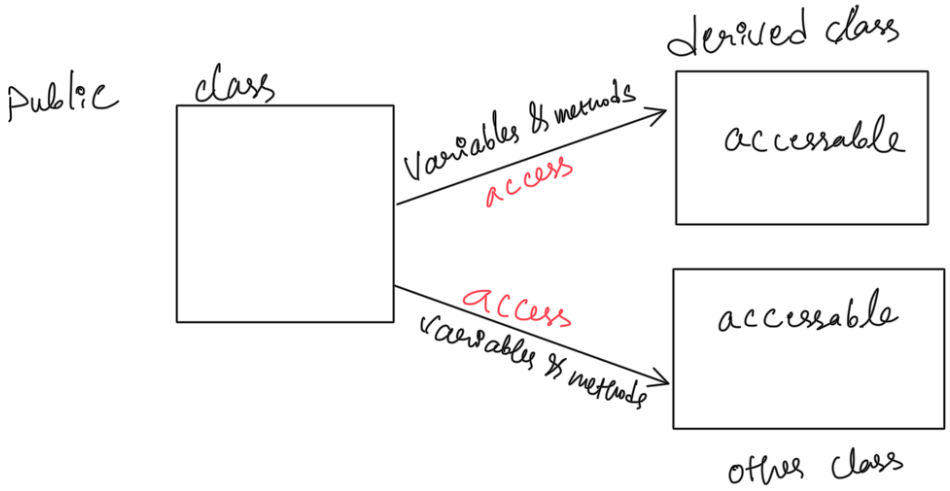


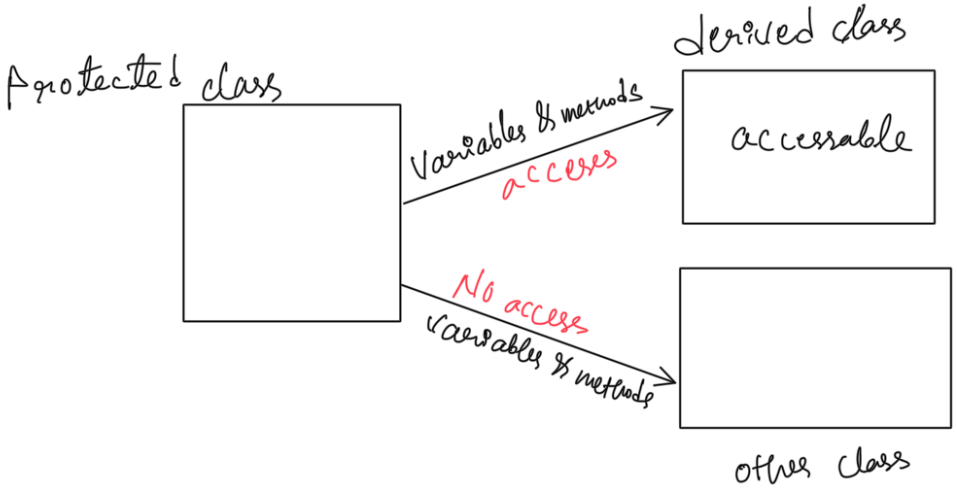
NOTE : We can also Override Some methods or properties In child class (methods & properties which are in parent class)

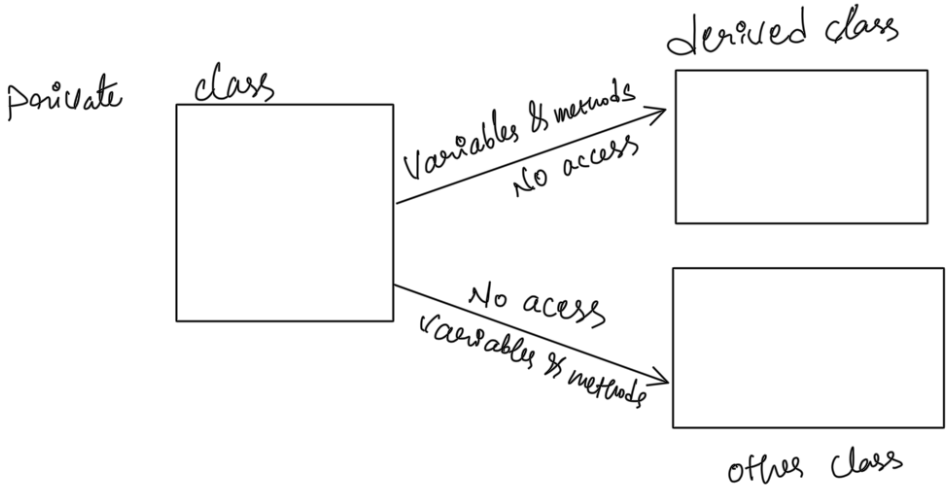
Access Modifier's

public ⟶ Accessable In derived classes and other classes
private ⟶ Not accessable In derived classes
protected ⟶ Accessable In derived classes.

→ Private variables or methods can be accessed only from the class in which they are declared and defined.







OOPs Advanced Concepts Comparison

Feature	Java	Python	C++
Syntax for Defining Inheritance	class Child extends Parent {}	class Child(Parent):	class Child : public Parent {}
Multiple Inheritance	Not supported directly. Achieved through interfaces with implements.	Fully supported. Handled using Method Resolution Order (MRO).	Fully supported. Handled via virtual inheritance to avoid the Diamond Problem.
Access Modifiers	public, protected, private. No support for private inheritance.	No strict keywords for access modifiers, but uses _protected and __private name mangling conventions.	public, protected, private inheritance. Control over visibility in the derived class.
Method Overriding	Uses @Override annotation. super keyword to call the parent method.	Supports overriding. Uses super() to call the parent method.	Supports overriding with virtual keyword. C++11 onwards allows override keyword for clarity.
Constructor and Destructor Calls	Child class constructor calls parent class constructor using super(). No destructors (garbage collector handles cleanup).	super().__init__() or ParentClass.__init__(self) for constructor chaining. Destructors (__del__) rarely needed due to garbage collection.	Parent constructor called automatically, or explicitly in child. Destructors called in reverse order (child first).
Abstract Classes and Interfaces	Abstract classes and interfaces enforce method overriding. Interfaces can have default methods from Java 8 onwards.	Abstract classes use the abc module. Methods marked with @abstractmethod.	Abstract classes created using pure virtual functions (= 0). Any class with at least one is abstract.
Inheritance Types	Single inheritance only (via extends). Multiple inheritance through interfaces (via implements).	Supports single and multiple inheritance directly.	Supports single and multiple inheritance directly.
Diamond Problem	Avoided by design—multiple inheritance not allowed for classes. Interfaces are used to avoid this issue.	Handled using Method Resolution Order (MRO) to ensure the correct method resolution path.	Resolved using virtual inheritance (virtual keyword).

This table provides a structured overview of how inheritance is managed in Java, Python, and C++.

Private : only accessible within the class.

public : can be accessed by Any class

protected : Accessible within the package and subclasses

Method overloading

Method overloading allows multiple methods with the same name But with different parameters.

It's like asking librarian for help with either a book title or an author name.

Class

```
method 1 ( parameters 1) {

}
method 1 (Pas1, Pas2) {

}
```

The Return type doesn't effect the overloading

You **cannot** overload a method **just by changing its return type.**

❌ **Example That Will NOT Compile:**

java Copy

```java
public class Demo {

    public int add(int a, int b) {
        return a + b;
    }

    // ❌ Compile-time error: method already defined
    public double add(int a, int b) {
        return a + b;
    }
}
```

❌ Java will say: *"method add(int, int) is already defined in class Demo"* — even though the return types are different.

✅ **Valid Overloading Examples (Different Parameters):**

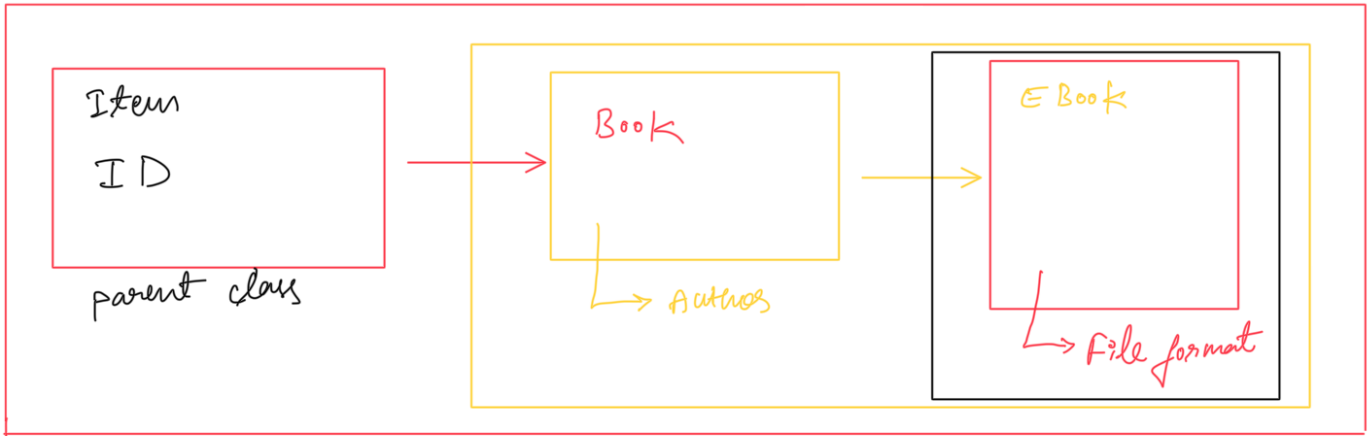
java Copy

```java
public class Demo {

    public int add(int a, int b) {
        return a + b;
    }

    public double add(double a, double b) {
        return a + b;
    }

    public int add(int a, int b, int c) {
        return a + b + c;
    }

    public String add(String a, String b) {
        return a + b;
    }
}
```

📌 **Summary**

Case	Valid Overloading?
Different parameter list	✅ Yes
Different return type **only**	❌ No
Different access modifier	❌ No
Different parameter order or type	✅ Yes

Multilevel Inheritance

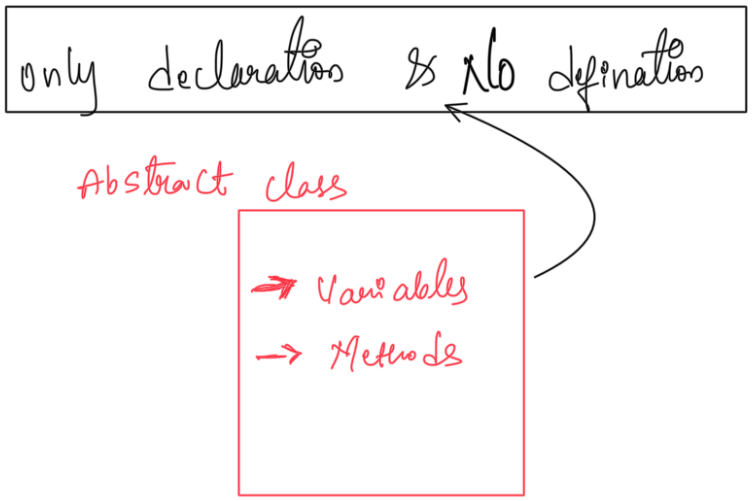


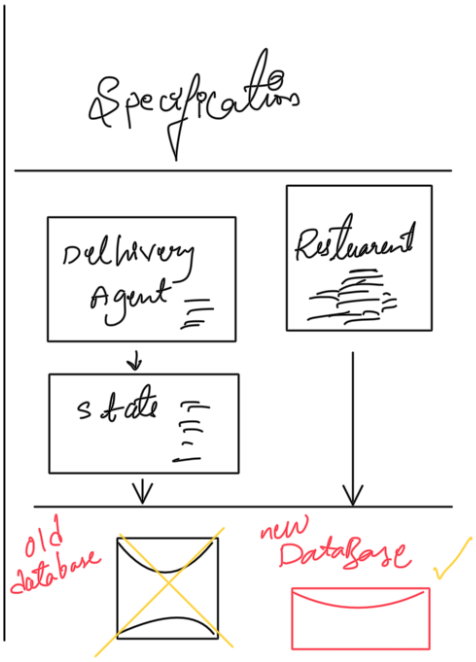
grand parent ⟶ Parent ⟶ child

multilevel inheritance is a type of inheritance where a class is derived from a class, which also derived from another class.

```java
// Base Class: Item
class Item {
    String title;

    public void setTitle(String title) {
        this.title = title;
    }

    public void displayTitle() {
        System.out.println("Title: " + title);
    }
}

// Intermediate Class: Book (inherits from Item)
class Book extends Item {
    String author;

    public void setAuthor(String author) {
        this.author = author;
    }

    public void displayAuthor() {
        System.out.println("Author: " + author);
    }
}

// Derived Class: EBook (inherits from Book)
class EBook extends Book {
    double fileSizeMB;

    public void setFileSize(double fileSizeMB) {
        this.fileSizeMB = fileSizeMB;
    }

    public void displayFileSize() {
        System.out.println("File Size: " + fileSizeMB + " MB");
    }
}
```

Abstractions

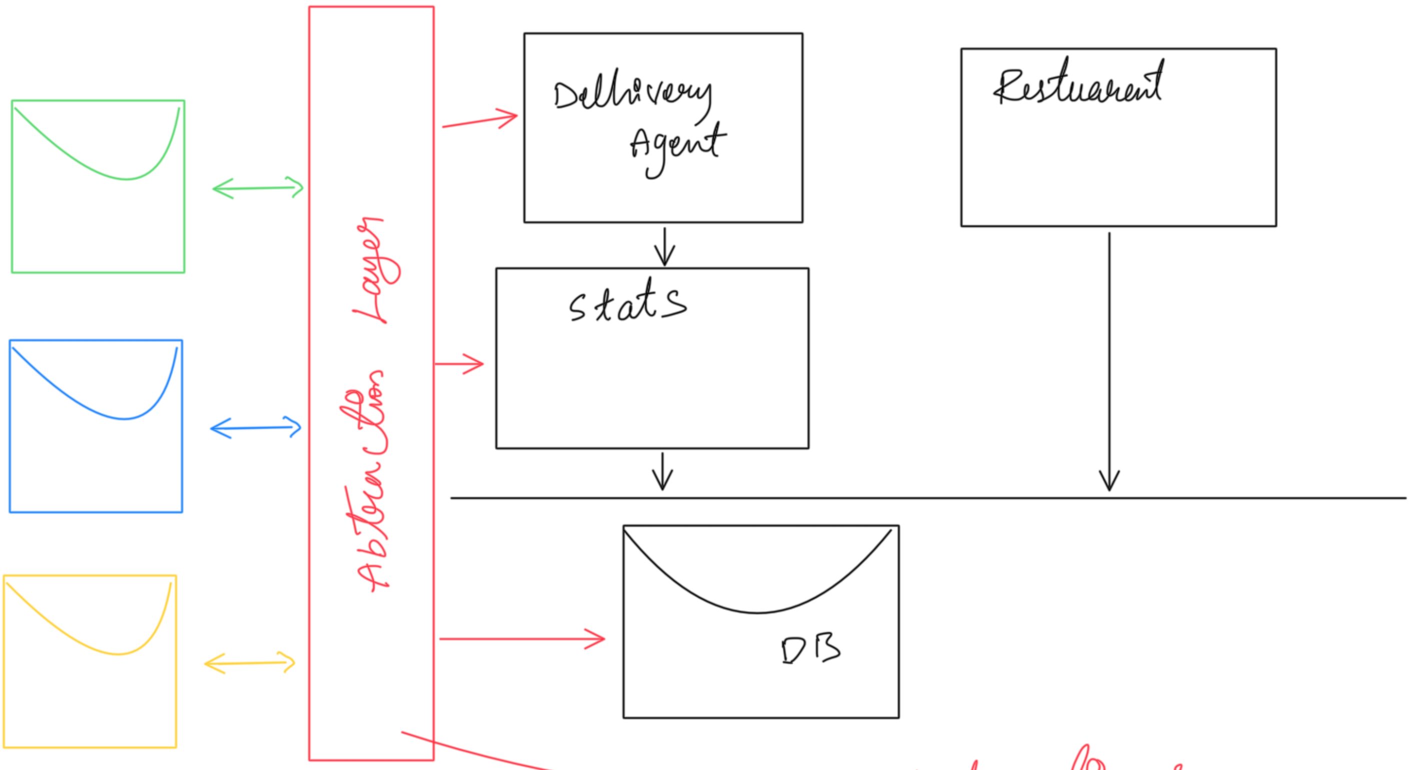




Here let assume that we Need to change the database for our Applications, then different databases support different API calls if i dont use abstractions I Need to change all the API Calls Code from the Applications Source Code.

It like I Need to change tyre for my Bike Instead of Installing new tyre to my car I decide to change the tyre specifications of my Bike. suppose its like stupid me purchased the CAR tyre and to Install it to my Bike I try to change the Bike's tyre Specifications. Is it correct method?

So deal with this type of design Issues the Concept of abstractions came into picture



Abstraction Layes can talk with Any type of Data Base

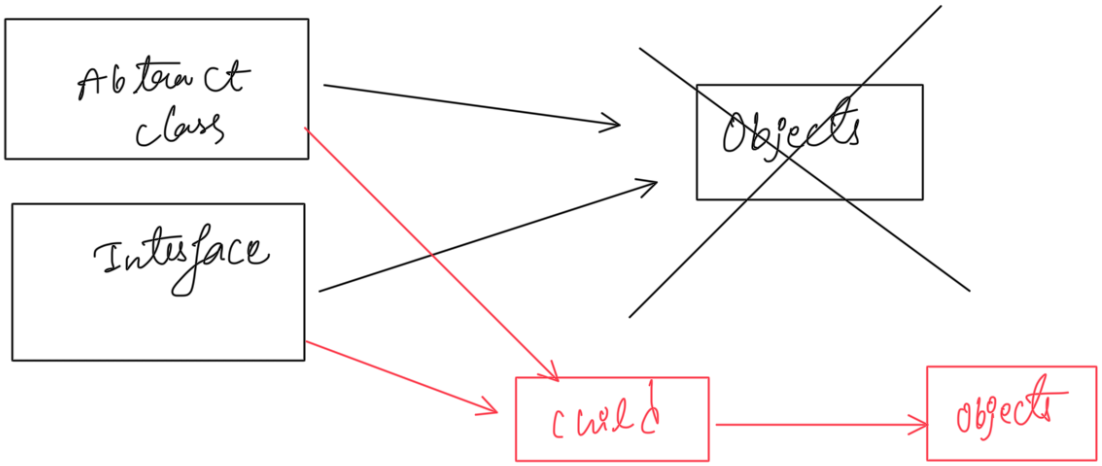
Abtraction is one of the four main pillars of object-oriented Programming. It means Hiding the internal Implementation details and showing only the essential features of an object.

```
Abstract class Item
{
    Private ID;                    ---> declared
    public Name;                   ---> declared
    void Search (String Name);     ---> declared
    void print_Details () {        ---> print Details is defined.
        ____
        ____
        ____
    }
}
```

We cannot create object of Abstract Class directly

Abtract Classe's inheorited Classes we can create objects

Similarly for Interface.



We can change or modify Abstract or Interface classes code in Run time Also.

Interface class Item

{

```
Private  ID ;
public   Name ;
Void     Search (String Name) ;
Void     Print_Details ( ) ;
```

declared

only declaration no definations

}

NOTE : one class can Implement Multiple Interfaces

Interface is like a Rule Book.

✅ Using Abstract Class

java

```java
abstract class LibraryItem {
    String title;
    String author;

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

    // Abstract method
    abstract void displayInfo();
}
```

✅ Using Interface

java

```java
interface Downloadable {
    void download();
}
```

📘 Book Class (Extends Abstract Class)

java

```java
class Book extends LibraryItem {
    Book(String title, String author) {
        super(title, author);
    }

    @Override
    void displayInfo() {
        System.out.println("Book: " + title + " by " + author);
    }
}
```

📗 EBook Class (Extends Abstract + Implements Interface)

java

```java
class EBook extends LibraryItem implements Downloadable {
    EBook(String title, String author) {
        super(title, author);
    }

    @Override
    void displayInfo() {
        System.out.println("EBook: " + title + " by " + author);
    }

    @Override
    public void download() {
        System.out.println("Downloading " + title);
    }
}
```

▶️ Main Method to Run:

java

```java
public class LibraryMain {
    public static void main(String[] args) {
        Book book = new Book("The Alchemist", "Paulo Coelho");
        EBook ebook = new EBook("Digital Fortress", "Dan Brown");

        book.displayInfo();
        ebook.displayInfo();
        ebook.download();
    }
}
```

📄 **Output:**

```
Book: The Alchemist by Paulo Coelho
EBook: Digital Fortress by Dan Brown
Downloading Digital Fortress
```