4️⃣

Abstraction(Detailed Notes)

🧾 Formal Definition of Abstraction (Elaborated)

> Abstraction is one of the fundamental principles of Object-Oriented Programming (OOP), which refers to the process of hiding the complex internal implementation details of a class and exposing only the necessary and relevant features to the user or the outside world.
>
> It allows developers to design systems that focus on **what an object does**, rather than **how it does it**, promoting a **clean separation between interface and implementation**.
>
> In Java, abstraction is achieved through the use of **abstract classes** and **interfaces**, both of which define a contract that must be followed, without necessarily providing a complete implementation.

🧩 Types of Abstraction in Java

Java supports two types of abstraction based on how much detail is hidden:

🔷 1. Partial Abstraction

> Achieved using abstract classes.

- An abstract class can have both:
  - **Abstract methods** (without body)

    - **Concrete methods** (with body)
- Supports partial abstraction since not everything is hidden.
- You can declare variables, constructors, and even implemented methods.

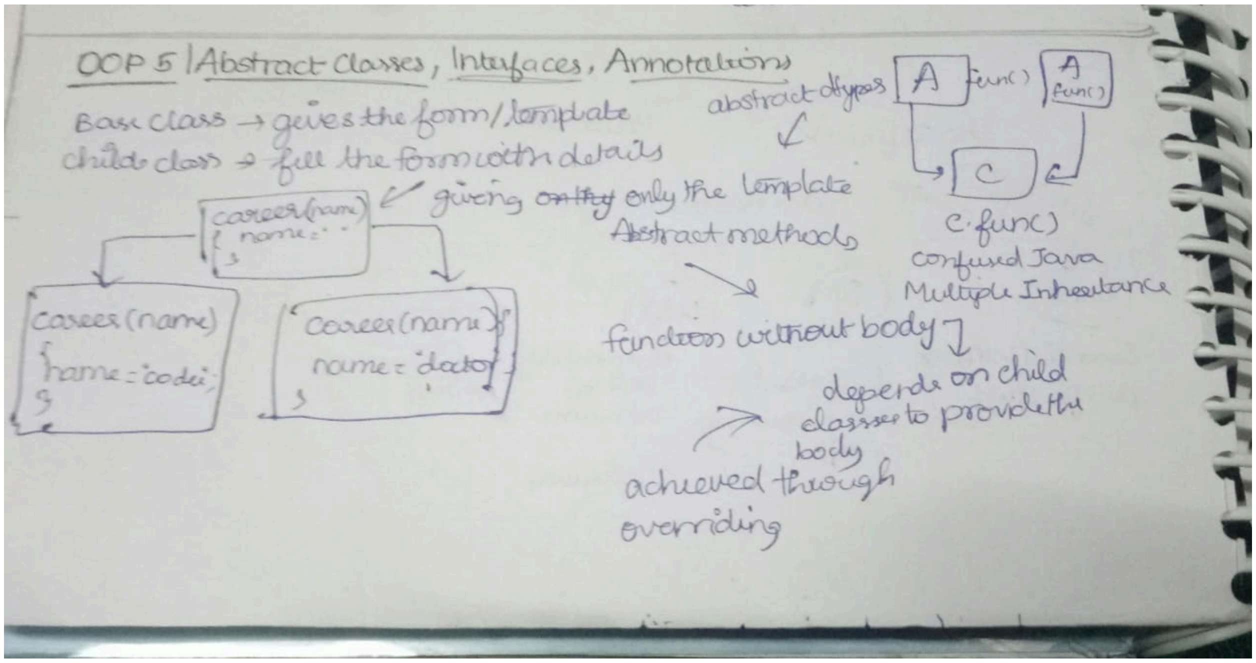
🧠 Useful when you want to provide **default behavior** but still leave room for **overriding specific parts** in child classes.

🔷 2. Full Abstraction

> Achieved using interfaces.

- An interface contains only **abstract method declarations** (before Java 8).
- In Java 8+, you can also include:
    - `default` methods (with body)
    - `static` methods
    - `private` helper methods
- Achieves **100% abstraction** (all behavior must be defined by implementing class).

🧠 Useful when you want to define a **common standard or contract** to be followed by multiple classes.



OOP 5 | Abstract Classes, Interfaces, Annotations

Base class → gives the form/template

child class → fill the form with details

giving ~~only~~ only the template

career(name)
{ name:" "
}

career(name)
{
name="coder";
}

career(name)
{
name="doctor"
}

abstract dtypes

Abstract methods

function without body

depends on child classes to provide the body

achieved through overriding

A fun()

A fun()

C

c.fun()

confused Java

Multiple Inheritance

Parent .java

```
package OOPS.Abstraction;

public abstract class Parent{
    int age;
    final int VALUE;
    //Still a single class cannot extend (inherits) more than 1 parent class(Multiple I
    public Parent(int age){
        this.age=age;
        VALUE=122345456;
    }

    abstract void career(String name);
    abstract void career(String name,int age);
}
```

Daughter.java

```
package OOPS.Abstraction;
```

```
public class Daughter extends Parent{
    @Override
    void career(String name){
        System.out.println("I am going to be a"+name);
    }

    @Override
    void career(String name, int age) {
        System.out.println("I love"+name+"He is : "+age);
    }
}
```

Son.java

```
package OOPS.Abstraction;

public class Son extends Parent{
    @Override
    void career(String name){
        System.out.println("I am going to be a Doctor"+name);
    }

    @Override
    void career(String name, int age) {
        System.out.println("I love"+name+"She is :"+age);
    }
}
```

Main.java

```
package OOPS.Abstraction;

public class Main {
    public static void main(String args[]){
       Son son=new Son();
       son.career(" Doctor");
```

```
        Daughter daughter=new Daughter();
        daughter.career(" Rahmania ",25);

        // Eventhough objects of abstract classes cannot be created
        //but it can be used as reference variable
        Parent daughter1 = new Daughter();
        daughter1.career(" Engineer");

        //Final Abstract classes cannot be created
        //Abstract classes are created to be overriden and inherited and final
        //prevents it from overriding


    }
}
```