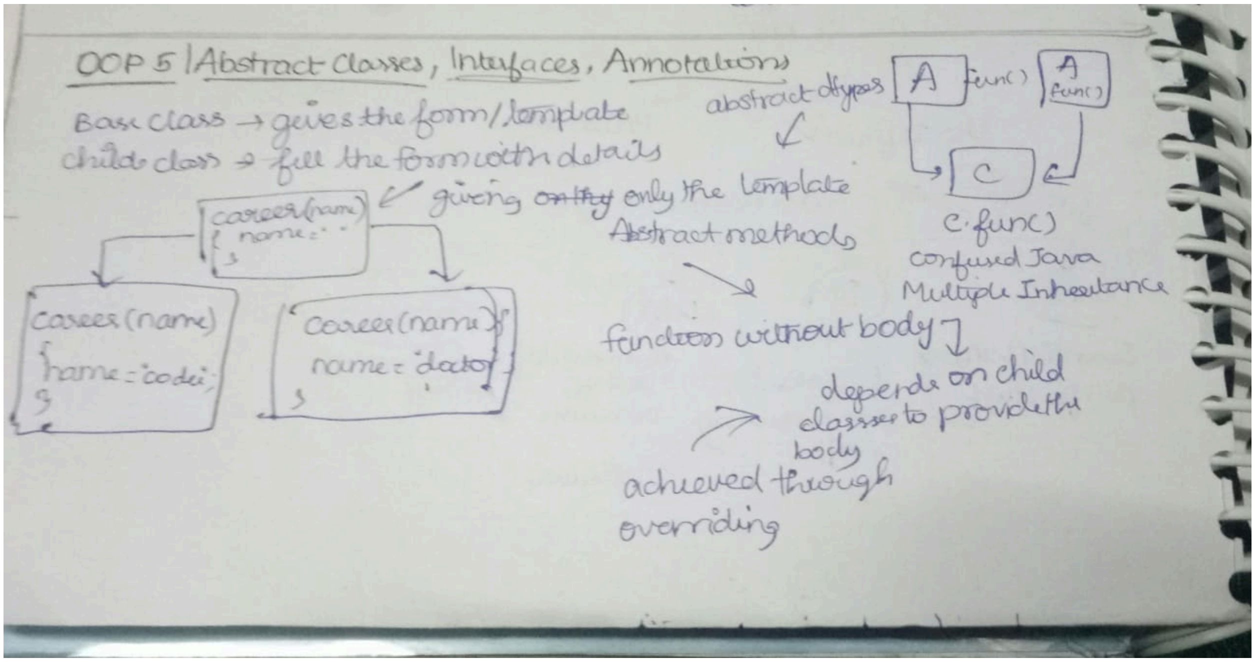
5️⃣

Interfaces

🔗 Interfaces in Java



OOP 5 | Abstract Classes, Interfaces, Annotations

Base class → gives the form/template
child class → fill the form with details

giving ~~on they~~ only the template

Abstract methods

functions without body

depends on child classes to provide the body

achieved through overriding

abstract d types

c.func()
confused Java
Multiple Inheritance

career(name)
{ name: " "
}
career(name)
{
name: "coder";
}
career(name)
{
name = "doctor"
}
A
func()
A
func()
C

✅ Formal Definition:

> An interface in Java is a reference type, similar to a class, that can contain only abstract methods, default methods, static methods, and constants (public static final variables).
>
> It defines a **contract** that any class implementing the interface must fulfill.

🔍 Key Characteristics:

- All methods in an interface are **public and abstract** by default (except `static`, `default`, and `private` methods from Java 8+).

- Variables are **public static final** by default.
- A class **implements** an interface using the `implements` keyword.
- Interfaces support **full abstraction** — they expose only method declarations.
- From Java 8+, interfaces can have:
  - `default` methods (with body)
  - `static` methods
  - `private` methods (for internal use)

🔁 Interface Inheritance:

- An interface can extend **one or more interfaces**.
- A class can implement **multiple interfaces** (supports **multiple inheritance**).
-

🚘 Java Project: Interfaces & Dynamic Composition (NiceCar)

✅ Concepts Covered:

- Interfaces (`Engine`, `Brake`, `Media`)
- Interface Implementation (`PowerEngine`, `ElectricEngine`, `CDPlayer`)
- Multiple Interface Implementation (`Car`)
- Dynamic Object Composition (`NiceCar`)
- Polymorphism with Interfaces

Brake.java

```
package OOPS.Intefaces;

public interface Brake {
    void brake();
    void start();
}
```

Car.java

```
package OOPS.Intefaces;

public class Car implements Engine, Brake, Media {
    @Override
    public void brake() {
        System.out.println("I brake like a normal car");
    }

    @Override
    public void start() {
        System.out.println("I start like a normal car");
    }

    @Override
    public void stop() {
        System.out.println("I stop like a normal car");
    }

    @Override
    public void acc() {
        System.out.println("I accelerate like a normal car");
    }
}
```

CDPlayer.java

```
package OOPS.Intefaces;

public class CDPlayer implements Media {
    @Override
    public void start() {
        System.out.println("Music started");
    }

    @Override
    public void stop() {
        System.out.println("Music stopped");
    }
}
```

ElectricEngine.java

```
package OOPS.Intefaces;

public class ElectricEngine implements Engine {
    @Override
    public void start() {
        System.out.println("Electric engine started");
    }

    @Override
    public void stop() {
        System.out.println("Electric engine stopped");
    }

    @Override
    public void acc() {
        System.out.println("Electric engine accelerating");
    }
}
```

Engine.java

```
package OOPS.Intefaces;

public interface Engine {
    static final int PRICE=78000;
    void start();
    void stop();
    void acc();
}
```

Media.java

```
package OOPS.Intefaces;

public interface Media {
    void start();
    void stop();
}
```

NiceCar.java

```
package OOPS.Intefaces;

public class NiceCar {
    private Engine engine;
    private Media player = new CDPlayer();

    public NiceCar() {
        this.engine = new PowerEngine();
    }

    public NiceCar(Engine engine) {
        this.engine = engine;
    }

    public void start() {
        engine.start();
    }

    public void stop() {
        engine.stop();
    }

    public void accelerate() {
        engine.acc();
    }

    public void startMusic() {
        player.start();
    }

    public void stopMusic() {
        player.stop();
    }

    public void upgradeEngine(Engine engine) {
        this.engine = engine;
    }
}
```

PowerEngine.java

```
package OOPS.Intefaces;
public class PowerEngine implements Engine {
    @Override
    public void start() {
        System.out.println("Power engine started");
    }
    @Override
    public void stop() {
        System.out.println("Power engine stopped");
    }
    @Override
    public void acc() {
        System.out.println("Power engine accelerating");
```

```
  }
}
```