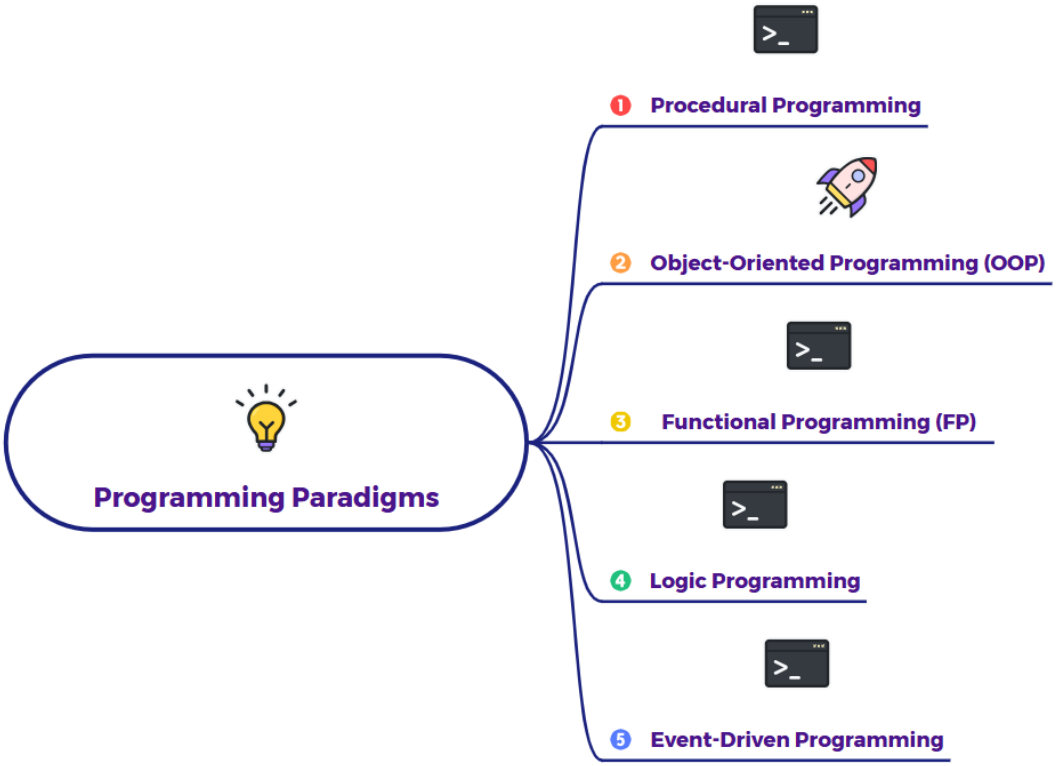
Explain Object oriented programming with an example



Imagine you're the chief designer for a car company.

The "Object-Oriented Vehicle Design" Way:

Instead of just listing steps to build *any* car (like in a procedural recipe), in OOP, you start by thinking about the **"things" (objects)** you'll be dealing with and what they *are* and what they *do*.

1. The Blueprint (Class)

- **Analogy:** You don't just build a random car. You first create a **blueprint** for a Car. This blueprint isn't a physical car; it's a detailed plan that describes:
    - **What every Car will *have* (its "attributes" or "data"):** Number of wheels, color, maximum speed, fuel level, brand, model.

    - **What every Car can *do* (its "methods" or "actions"):** Start engine, stop engine, accelerate, brake, open doors, turn on lights.
- In OOP, this blueprint is called a **Class**.

2. **The Actual Vehicle (Object)**
    - **Analogy:** Once you have the Car blueprint, you can use it to build many actual cars.
        - You build "MyRedHondaCity" using the Car blueprint. It has 4 wheels, is red, can go 180 km/h, etc.
        - You build "YourBlueMahindraThar" using the Car blueprint. It also has 4 wheels, is blue, can go 150 km/h, etc.
    - Each of these individual, real cars is an **Object** (an "instance" of the Car class). They are all cars, but they have their own specific data (color, model) and they can all perform the actions defined in the Car blueprint.
3. **Tucking Things Away (Encapsulation)**
    - **Analogy:** When you drive a car, you don't need to know the intricate details of how the engine's pistons move or how the fuel injectors work. You just press the accelerator, and the car speeds up. The complex internal workings are **hidden** inside the car's engine compartment. You interact with simple controls (accelerator, brake, steering wheel).
    - In OOP, this is **Encapsulation**. It means bundling the data (like fuel level) and the methods (like accelerate()) that operate on that data together inside the object, and largely hiding the internal complexity. This prevents accidental changes to the car's internal state from outside.
4. **Building on Existing Designs (Inheritance)**
    - **Analogy:** Let's say you now want to design a "Sports Car." A Sports Car is still fundamentally a Car (it has wheels, can accelerate, etc.), but it has some extra features: a spoiler, a turbo boost, and perhaps a louder engine sound.
    - Instead of starting a brand new blueprint from scratch, you can say, "A Sports Car blueprint will **inherit** all the features from the Car blueprint, and then I'll just add the specific Sports Car features."

- In OOP, this is **Inheritance**. It allows you to create new classes (like SportsCar) that automatically get all the attributes and methods of an existing class (like Car), promoting code reuse.

5. Different Cars, Same Command (Polymorphism)

- **Analogy:** Imagine you give the command "start engine."
    - On a petrol car, "start engine" might mean sparking the fuel.
    - On an electric car, "start engine" might mean activating the battery and motor.
    - On a diesel bus, "start engine" might mean igniting diesel fuel.
- The **command is the same ("start engine")**, but how each *type* of vehicle (object) performs that action is specific to its own design.
- In OOP, this is **Polymorphism** (meaning "many forms"). It allows different objects to respond to the same command in their own unique ways, making your code very flexible.

6. Focusing on What Matters (Abstraction)

- **Analogy:** When you look at a car's dashboard, you see essential information like speed, fuel level, and warning lights. You don't see the thousands of wires, sensors, and microchips that make it all work. You only see what's essential to operate the car.
- In OOP, this is **Abstraction**. It means simplifying complex systems by showing only the necessary details and hiding the irrelevant complexity. You interact with objects through simple interfaces without needing to understand their internal machinery.

Why Use This "Object-Oriented" Way?

Just like designing cars with blueprints makes it easier to:

- **Manage Complexity:** Break down a big problem (designing vehicles) into smaller, manageable pieces (individual car types).

- **Reuse Designs:** Once you have a Car blueprint, you don't have to redefine wheels and steering every time you make a new model.
- **Fix Problems Easier:** If there's a problem with how accelerate() works, you know to look in the Car blueprint or its specific instances.
- **Add New Features:** If you invent a new type of Flying Car, you can build upon the Car blueprint and just add the "flying" bits without disrupting existing car designs.

So, in short: **Object-Oriented Programming is about organizing your code around self-contained "things" (objects) that combine data and actions, using blueprints (classes), hiding complexity, and allowing new "things" to easily build upon and interact with existing ones.**