### Object-Oriented Programming Structure (OOPS)

OOPS is a programming paradigm based on the concept of “objects,” which can contain data and methods to manipulate that data. It provides a structured and modular approach to software development.

### Key Concepts of OOPS

1. **Reusability**  
   The ability to reuse existing code in different applications or modules to save time and effort.
2. **Modularity**  
   Code is divided into smaller, manageable modules, making it easier to debug, test, and maintain.
3. **Security**  
   Data encapsulation and abstraction help in securing the code by restricting access to specific parts of an object.

### Features of OOPS

1. **Class**  
   A class is a blueprint or template for creating objects. It defines the properties (variables) and methods (functions) that the objects created from the class will have.  
   Syntax:

* class ClassName {  
   // properties and methods  
  }

1. **Object**  
   An object is an instance of a class that holds specific data and behaviors. It is created using the new keyword.  
   Syntax:

* let referenceVariable = new ClassName();

1. **Encapsulation**  
   Encapsulation binds data and methods together within a class and restricts direct access to some of an object’s components for better control and security.
2. **Inheritance**  
   Inheritance allows a class to derive properties and methods from another class, enabling code reusability and a hierarchical relationship between classes.
3. **Polymorphism**  
   Polymorphism enables objects to take on multiple forms, allowing the same operation to behave differently in different contexts (e.g., method overloading and overriding).
4. **Abstraction**  
   Abstraction focuses on hiding the implementation details of a class and showing only the essential features.

### Example 1: Mobile Class

#### Code:

class Mobile {  
 powerButton;  
 camera;  
 volumeButton;  
  
 constructor(powerButton, camera, volumeButton) {  
 this.powerButton = powerButton;  
 this.camera = camera;  
 this.volumeButton = volumeButton;  
 }  
}  
  
// Creating instances (objects) of the Mobile class  
var apple = new Mobile("Metal", "48mp", "Metal");  
var realme = new Mobile("Plastic", "200mp", "Plastic");  
var oppo = new Mobile("Plastic", "108mp", "Plastic");  
  
// Output the object data  
console.log(apple);  
console.log(realme);  
console.log(oppo);

#### Explanation:

* **Class Definition**  
  The Mobile class has three properties: powerButton, camera, and volumeButton.
* **Object Creation**  
  Objects apple, realme, and oppo are created with different property values.
* **Output**  
  The console.log statements display the properties of each object.

### Example 2: Car Class

#### Code:

class Car {  
 engine;  
 power;  
 enginecc;  
 colour;  
 brand;  
  
 constructor(engine, power, enginecc, colour, brand) {  
 this.engine = engine;  
 this.power = power;  
 this.enginecc = enginecc;  
 this.colour = colour;  
 this.brand = brand;  
 }  
  
 // Non-static methods  
 start(key) {  
 console.log(key, this.brand, "Engine Started");  
 }  
 accelaration(speed) {  
 console.log(speed, "Thokkali");  
 }  
 brake() {  
 console.log("Applied Brakes");  
 }  
 stop() {  
 console.log("Engine Stopped", this.brand);  
 }  
}  
  
// Creating an object of Car class  
var thar = new Car("Petrol", "350bhp", "2000cc", "Red", "Mahindra");  
thar.start("Key is available");  
thar.accelaration("100 speed");  
thar.brake();  
thar.stop();

#### Explanation:

1. **Class Definition**  
   The Car class has five properties:
   * engine: Type of engine (e.g., Petrol, Diesel)
   * power: Engine power in bhp or hp
   * enginecc: Engine capacity in cc
   * colour: Color of the car
   * brand: Brand of the car

* The Car class also defines four non-static methods:
  + start(key): Logs a message indicating the engine has started.
  + accelaration(speed): Logs the current speed.
  + brake(): Logs that brakes are applied.
  + stop(): Logs that the engine has stopped.

1. **Object Creation**  
   The object thar is created with specific property values.
2. **Method Calls**  
   Methods are invoked using the thar object reference:
   * start() initializes the car’s engine.
   * accelaration() simulates the car’s acceleration.
   * brake() stops the car momentarily.
   * stop() shuts down the engine.

#### Additional Objects:

var audi = new Car("Diesel", "650hp", "2000cc", "Pink", "Audi");  
audi.start("Key is available");  
audi.accelaration("150 speed");  
audi.brake();  
audi.stop();  
  
var nano = new Car("Diesel", "200bhp", "1000cc", "Yellow", "Tata");  
nano.start("Key available");  
nano.accelaration("60 speed");  
nano.brake();  
nano.stop();

#### Highlights:

* Objects audi and nano demonstrate how the same class can be reused to create instances with different data.
* Methods operate on these objects individually, showing the dynamic behavior of classes and objects.

# Examples

// Example 1: Laptop Class  
class Laptop {  
 brand;  
 processor;  
 ram;  
 storage;  
 color;  
  
 constructor(brand, processor, ram, storage, color) {  
 this.brand = brand;  
 this.processor = processor;  
 this.ram = ram;  
 this.storage = storage;  
 this.color = color;  
 }  
  
 // Method to display laptop details  
 displayDetails() {  
 console.log(`Brand: ${this.brand}, Processor: ${this.processor}, RAM: ${this.ram}, Storage: ${this.storage}, Color: ${this.color}`);  
 }  
  
 // Method to upgrade RAM  
 upgradeRAM(newRAM) {  
 this.ram = newRAM;  
 console.log(`RAM upgraded to ${this.ram}`);  
 }  
  
 // Method to change the color  
 repaint(newColor) {  
 this.color = newColor;  
 console.log(`Color changed to ${this.color}`);  
 }  
}  
  
let dell = new Laptop("Dell", "Intel i5", "8GB", "512GB SSD", "Black");  
dell.displayDetails();  
dell.upgradeRAM("16GB");  
dell.repaint("Silver");  
dell.displayDetails();

// Example 2: Bike Class  
class Bike {  
 model;  
 engine;  
 mileage;  
 color;  
  
 constructor(model, engine, mileage, color) {  
 this.model = model;  
 this.engine = engine;  
 this.mileage = mileage;  
 this.color = color;  
 }  
  
 // Method to display bike details  
 displayDetails() {  
 console.log(`Model: ${this.model}, Engine: ${this.engine}, Mileage: ${this.mileage}, Color: ${this.color}`);  
 }  
  
 // Method to update mileage  
 updateMileage(newMileage) {  
 this.mileage = newMileage;  
 console.log(`Mileage updated to ${this.mileage}`);  
 }  
  
 // Method to repaint the bike  
 repaint(newColor) {  
 this.color = newColor;  
 console.log(`Bike repainted to ${this.color}`);  
 }  
}  
  
let yamaha = new Bike("Yamaha FZ", "150cc", "45kmpl", "Blue");  
yamaha.displayDetails();  
yamaha.updateMileage("50kmpl");  
yamaha.repaint("Red");  
yamaha.displayDetails();

// Example 3: Book Class  
class Book {  
 title;  
 author;  
 pages;  
 price;  
  
 constructor(title, author, pages, price) {  
 this.title = title;  
 this.author = author;  
 this.pages = pages;  
 this.price = price;  
 }  
  
 // Method to display book details  
 displayDetails() {  
 console.log(`Title: ${this.title}, Author: ${this.author}, Pages: ${this.pages}, Price: ${this.price}`);  
 }  
  
 // Method to apply discount  
 applyDiscount(discountPercentage) {  
 this.price -= (this.price \* discountPercentage) / 100;  
 console.log(`Discount applied. New price is ${this.price}`);  
 }  
  
 // Method to change author  
 changeAuthor(newAuthor) {  
 this.author = newAuthor;  
 console.log(`Author updated to ${this.author}`);  
 }  
}  
  
let novel = new Book("The Alchemist", "Paulo Coelho", 208, 500);  
novel.displayDetails();  
novel.applyDiscount(10);  
novel.changeAuthor("Anonymous");  
novel.displayDetails();

// Example 4: Smartphone Class  
class Smartphone {  
 brand;  
 model;  
 storage;  
 battery;  
 color;  
  
 constructor(brand, model, storage, battery, color) {  
 this.brand = brand;  
 this.model = model;  
 this.storage = storage;  
 this.battery = battery;  
 this.color = color;  
 }  
  
 // Method to display smartphone details  
 displayDetails() {  
 console.log(`Brand: ${this.brand}, Model: ${this.model}, Storage: ${this.storage}, Battery: ${this.battery}, Color: ${this.color}`);  
 }  
  
 // Method to upgrade storage  
 upgradeStorage(newStorage) {  
 this.storage = newStorage;  
 console.log(`Storage upgraded to ${this.storage}`);  
 }  
  
 // Method to change color  
 changeColor(newColor) {  
 this.color = newColor;  
 console.log(`Color changed to ${this.color}`);  
 }  
}  
  
let samsung = new Smartphone("Samsung", "Galaxy S21", "128GB", "4000mAh", "Black");  
samsung.displayDetails();  
samsung.upgradeStorage("256GB");  
samsung.changeColor("White");  
samsung.displayDetails();