Object-Oriented Programming (OOP) in JavaScript

Introduction

JavaScript is a prototype-based language but since ES6, it supports 'class' syntax which makes it easier to implement OOP concepts like classes, objects, inheritance, encapsulation, abstraction, and polymorphism.

Classes & Objects

A class is a blueprint for creating objects. An object is an instance of a class. Example:

```
class Person {
constructor(name, age) {
this.name = name;
this.age = age;
}
greet() {
console.log(`Hello, I am ${this.name}, ${this.age} years old.`);
}
}

const p1 = new Person('Rahul', 25);
p1.greet();
```

Inheritance

Inheritance allows one class to extend another. Example:

```
class Animal {
constructor(name) { this.name = name; }
speak() { console.log(`${this.name} makes a sound`); }
}

class Dog extends Animal {
speak() { console.log(`${this.name} barks`); }
}

const d = new Dog('Tommy');
d.speak();
```

Encapsulation

Encapsulation hides internal details and exposes only necessary parts. Example with private fields:

```
class BankAccount {
#balance = 0;
deposit(amount) { this.#balance += amount; }
getBalance() { return this.#balance; }
}
```

```
const acc = new BankAccount();
acc.deposit(100);
console.log(acc.getBalance()); // 100
```

Abstraction

Abstraction hides implementation and shows only essential features. JavaScript does not have abstract classes or interfaces, but we can simulate them:

```
class Vehicle {
startEngine() { throw new Error('startEngine() must be implemented'); }
}

class Car extends Vehicle {
startEngine() { console.log('Car engine started'); }
}
```

Polymorphism

Polymorphism allows the same method to behave differently. Example:

```
class Shape { area() { return 0; } }

class Circle extends Shape {
constructor(r) { super(); this.r = r; }
area() { return Math.PI * this.r * this.r; }
}

class Square extends Shape {
constructor(s) { super(); this.s = s; }
area() { return this.s * this.s; }
}

const shapes = [new Circle(5), new Square(4)];
shapes.forEach(s => console.log(s.area()));
```