JavaScript : Module ⑦ [OOPS]

__/__/____

OOPs :

- OOP (Object-Oriented Programming) is a programming paradigm (style of coding) where we organize code around objects instead of functions alone.

- JS support OOP, but it's a bit different because it's prototype based. Still, modern JS gives us classes to make OOP more familiar.

- Objects = data (properties) + behaviour (methods) packed together.

- Code Reusability
- Modularity
- Maintainability
- Scalability
- Real-world Modeling

① Encapsulation :

- Bundling data & methods together while hiding internal implementation details. JS uses closure and private fields (#) for encapsulation.

```
ex : class BankAccount {
        #balance = 0;          // Private Field
        deposit(amount) {
            if (amout > 0) {
                this.balance += amount;
            }
        }
        getBalance() {
            return this.#balance;
        }
    }
```

② Inheritence

Creating new classes (child / Subclass) based on existing class (parent / Superclass). The child inherits properties & methods from the parent and can add its own or modified inherited ones.

```
ex: class Vehicle {
      constructor (brand) {
        this.brand = brand;
      }

      start() {
        console.log (`${this.brand} started`);
      }
    }

    class Car extends Vehicle {
      constructor (brand, doors) {
        super (brand);   // calls parent class constructor
        this.doors = doors;
      }

      honk () {
        console.log ('Beep Beep !');
      }
    }
```

③ Abstraction

Hiding complex implementation details while exposing only necessary functionality, JS doesn't have true abstract classes, but you can simulate them

```
ex: class Shape {
      constructor {
```

__/__/____

```
if (this.constructor === Shape) {
    throw new Error ('Cannot instantiate abstract class');
}

// Abstract Method
calculateArea() {
    throw new Error ('Must implement calculateArea');
}

class Circle extends Shape {
    constructor (radius) {
        super();
        this.radius = radius;
    }

    calculateArea() {
        return Math.PI * this.radius ** 2;
    }
}
```

④ Polymorphism

Objects of different types responding to the same method call in their own way. JS achieves this through method overriding.

```
ex: class Animal {
    makeSound() {
        console.log ('some generic sound');
    }

    class Dog extends Animal {
        makeSound() {
            console.log ('woof!');
        }
    }
```

__/__/__

```
class Cat extends Animal {
  makeSound() {
    console.log('Meow!');
  }
}

// Polymorphism in action
const animals = [new Dog(), new Cat()];
animals.forEach(animal => animal.makeSound());
// Different sounds
```