OOPS in JS

Object-Oriented Programming (OOP) in JavaScript is a way to structure your code so that it revolves around objects. These objects represent real-world entities and are collections of properties (data) and methods (functions). JavaScript, although prototype-based, supports many concepts of traditional OOP like classes, inheritance, and encapsulation.

What is Object-Oriented Programming ?

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around data, or objects, rather than functions and logic. In OOP, the program is structured as a collection of interacting objects, each of which represents an instance of a class. A class serves as a blueprint for creating objects (instances), defining their properties (attributes) and behaviors (methods).

Core Concepts of OOP:

1. Class
2. Object
3. Encapsulation
4. Inheritance
5. Polymorphism
6. Abstraction

Benefits of OOP:

- **Modularity**: Code is organized into objects, making it easier to manage and modify.
- **Reusability**: Classes and objects can be reused across different parts of the program.
- **Maintainability**: It's easier to maintain and update code due to its modular nature.
- **Scalability**: OOP makes it easier to scale applications by adding new classes or extending existing ones.

Real-World Example:

Consider a bank account system:

- Each account is an object, with attributes like balance and methods like deposit or withdraw.
- You could have different types of accounts (e.g., checking, savings) that inherit from a common Account class, but each has its own implementation of certain methods.

Class, Objects and Constructors

Class : A class is a blueprint for creating objects with shared properties and methods. It defines the structure and behavior that objects created from the class will have.

```
class ClassName {
  constructor() {
    // Constructor function to initialize properties
  }

  method() {
    // Define methods
  }
}
```

constructor(): This is a special method called when an object is instantiated (created) from the class. It is used to initialize the object's properties.

Methods: Functions defined inside the class are called methods. These methods define the behavior of the objects created from the class.

An object is an instance of a class, or simply a collection of key-value pairs. Objects can have properties (attributes) and methods (functions that operate on those attributes).

Key Differences:

1. **Class**: Defines a blueprint.
2. **Object**: An instance of that blueprint.

Example

```
class Car {
  constructor(make, model, year) {
    this.make = make; // Property: car make
    this.model = model; // Property: car model
    this.year = year; // Property: car year
  }

  displayInfo() {
    console.log(`${this.year} ${this.make} ${this.model}`);
  }
}

const myCar = new Car('Toyota', 'Corolla', 2020);

// Accessing the properties
console.log(myCar.make);  // Output: Toyota
console.log(myCar.model); // Output: Corolla
console.log(myCar.year);  // Output: 2020

// Calling the method
myCar.displayInfo();  // Output: 2020 Toyota Corolla
```

Key Points

- You create an object from a class using the new keyword.
- The constructor method initializes the properties of the object when it's created.
- The object can then access methods defined in the class.

Private Variables

In JavaScript, you can define private fields within a class using the # symbol. This ensures that the variables are only accessible within the class and cannot be accessed from outside.

```
class Person {
  #name; // Private field

  constructor(name, age) {
    this.#name = name; // Initialize private field
    this.age = age; // Public field
  }

  getName() {
    return this.#name; // Access private field within a method
  }
}

const person1 = new Person('Alice', 30);
console.log(person1.getName());  // Output: Alice
console.log(person1.#name); // Error: Private field '#name' must be declared in an enclosing class
```

Static functions and Static variables

In JavaScript, static functions and static variables are used within a class to define properties and methods that belong to the class itself, rather than to instances of the class. This means that you can call static methods and access static variables without having to create an instance of the class.

Static methods are functions that belong to the class itself and can be called without creating an instance of the class. They are typically used for utility functions or operations that don't depend on instance properties.

```
class ClassName {
  static staticMethod() {
    console.log('This is a static method');
  }
}

ClassName.staticMethod(); // Calling the static method without an instance
```

Static variables (also known as static properties) are properties that are shared by all instances of the class. These properties are not tied to a specific instance but to the class itself.

```
class Car {
  static wheels = 4;  // Static property

  constructor(make, model) {
    this.make = make; // Instance property
    this.model = model; // Instance property
  }

  static displayWheels() {
    console.log(`A car typically has ${Car.wheels} wheels.`);
  }
}

// Accessing static property
console.log(Car.wheels);  // Output: 4

// Calling static method
Car.displayWheels(); // Output: A car typically has 4 wheels.
```

Inheritance

Inheritance is one of the key features of **Object-Oriented Programming (OOP)**. In JavaScript, inheritance allows a class to inherit properties and methods from another class. This promotes code reuse and helps organize and extend functionality.

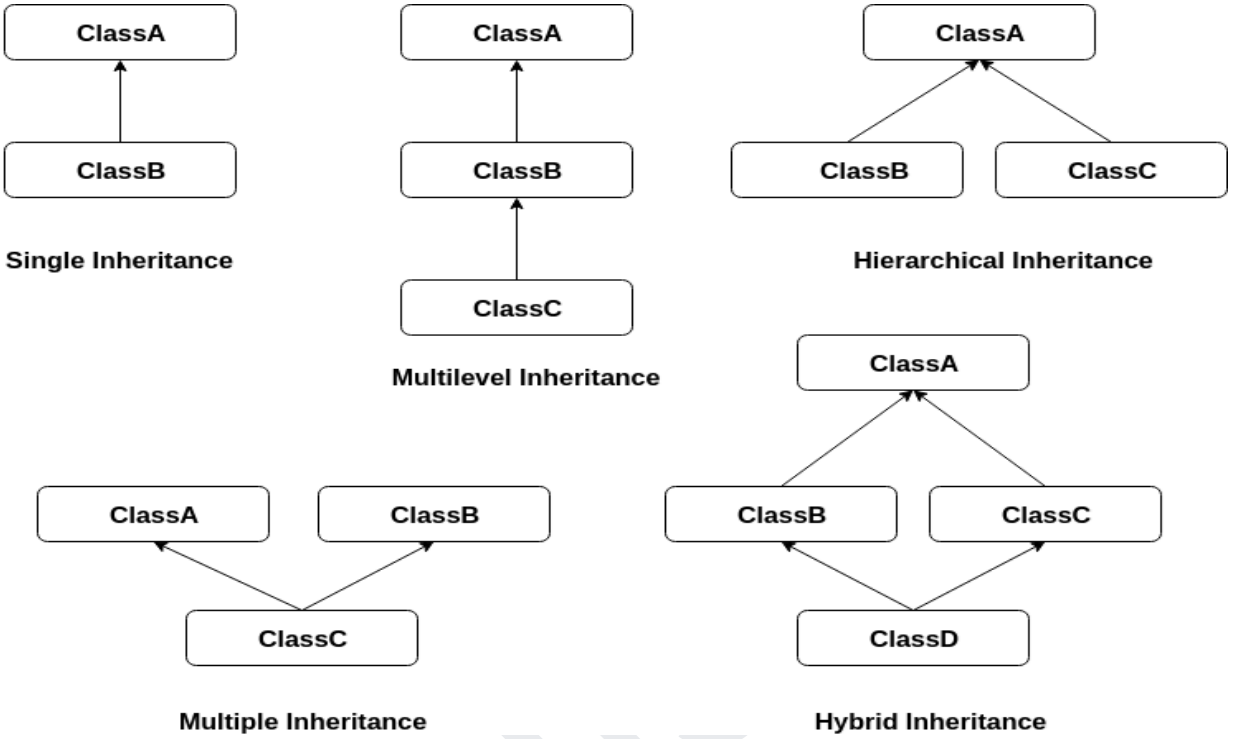
JavaScript implements inheritance using the `extends` keyword, which allows one class (a subclass or child class) to inherit from another class (a superclass or parent class).

```
class ParentClass {
  constructor() {
    // Parent class constructor
  }

  parentMethod() {
    console.log('This is a method in the parent class');
  }
}

class ChildClass extends ParentClass {
  constructor() {
    super();  // Call the parent class constructor
    // Child class constructor
  }

  childMethod() {
    console.log('This is a method in the child class');
  }
}
```

Types of inheritance:

- **Single Inheritance**: A class inherits from a single parent class.
- **Multilevel Inheritance**: A chain of inheritance where a class inherits from another class, which itself inherits from another class.
- **Hierarchical Inheritance**: Multiple classes inherit from a single parent class.
- **Multiple Inheritance**: A class inherits from multiple parent classes (not directly supported in JavaScript).

- **Hybrid Inheritance**: A combination of inheritance types.



Polymorphism

Polymorphism is a fundamental concept in object-oriented programming (OOP), including JavaScript. The term comes from Greek, meaning "many shapes." In programming, polymorphism refers to the ability of different classes or objects to respond to the same method in different ways, depending on their specific types.
Essentially, polymorphism allows a single function or method to work in a variety of ways based on the context or object it's operating on.

There are two main types of polymorphism in OOP:

1. Method Overriding (Runtime Polymorphism): When a subclass provides its own implementation of a method that is already defined in the parent class.
2. Method Overloading: A concept where multiple methods with the same name but different parameters exist (Note: JavaScript does not support traditional method overloading directly, but we can simulate it).

Method Overriding Example

```
// Parent class
class Animal {
  speak() {
    console.log("The animal makes a sound");
```

```
  }
}

// Child class overriding the speak method
class Dog extends Animal {
  speak() {
    console.log("The dog barks");
  }
}

const animal = new Animal();
animal.speak();  // Output: The animal makes a sound
const dog = new Dog();
dog.speak();  // Output: The dog barks
```

Questions List

Create a Person class:

- Create a class Person that has two properties: name and age.
- Add a method greet that logs a greeting message to the console.
- Create an instance of Person and call the greet method.

Encapsulation:

- Create a BankAccount class with private properties balance and accountHolder.
- Provide public methods deposit(amount) and withdraw(amount) to modify the balance.
- Ensure the balance cannot be set directly from outside the class (use private fields or closures).

Inheritance:

- Create a Vehicle class with properties like make, model, and year.
- Create a subclass Car that inherits from Vehicle and adds a property numberOfDoors.
- Override a method getDetails() in the Car class to return a more detailed description than Vehicle's method.

Method Overriding:

- Create a Shape class with a method area() that returns 0 (as a base class).
- Create two subclasses Circle and Rectangle that override the area() method to return the area of the circle and rectangle respectively.

Static Methods:

- Create a MathUtils class with a static method add(a, b) that returns the sum of two numbers.
- Show how to call the static method without instantiating the class.

Getter and Setter:

- Create a Student class with a private score property.
- Create getter and setter methods for score, ensuring that the score cannot be set to a negative value.