OOP

In JavaScript

CodeMaster Noyon

JS OOP Introduction

In this tutorial we will explore the fundamentals for **Object Oriented Programming** (OOP) in JavaScript.

OOP has four pillars:

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

We will explore these four pillars using JavaScript classes.

JS OOP Class

The basic syntax of class is:

```
class Animal {
    constructor(name, age){
        // these are properties of class
        this.Name = name;
        this.Age = age;
    }
    // this is a method of class
    makeSound(){
        console.log('woof');
    }
};
```

Creating objects of Animal class:

```
// A1 is an object of Animal class
const A1 = new Animal('Rex', 5);
console.log(A1.Name, A1.Age);  // Rex 5
```

JS OOP Inheritance

It’s a concept in which some properties and methods of a class can be reused by other objects. Properties & methods inherits from parent class to child class using `extends` keyword.

```
//child class
class Cat extends Animal{
  // Cat inherits from Animal
}
```

```
// C1 is an object of Cat class
const C1 = new Cat('Rex', 5);
console.log(C1.Name, C1.Age);  // Rex 5
```

```
// parent class
class Animal {
    constructor(name, age){
        // these are properties of
class
        this.Name = name;
        this.Age = age;
    }
    // this is a method of class
    makeSound(){
        console.log('woof');
    }
};
```

JS OOP Inheritance & super() method

If parent class & child class both have constructor, then you must be used `super()` method. Other wise you get Errors. The `super()` keyword used passing parameters to the parent class.

```
//child class
class Cat extends Animal{
    constructor(name, age, weight){
        super(name, age);
        this.Weight = weight;
    }
};
```

```
// C1 is an object of Cat class
const C1 = new Cat('Rex', 5, '10kg');
console.log(C1.Name, C1.Age, C1.Weight);  // Rex 5 10kg
```

```
// parent class
class Animal {
    constructor(name, age){
        // these are properties of
class
        this.Name = name;
        this.Age = age;
    }
    // this is a method of class
    makeSound(){
        console.log('woof');
    }
};
```

JS OOP Polymorphism

Polymorphism is a concept that utilizes inheritance for reusing methods multiple times with different behavior depending on class types.

```
// child class
class Dog extends Animal{
    constructor(name, age){
        super(name, age)
    }
    makeSound(){
        console.log('woof');
    }
};
```

```
//child class
class Cat extends Animal{
    constructor(name, age){
        super(name, age);
    }
    makeSound(){
        console.log('Mew');
    }
};
```

```
// D1 is an object of Dog class
const D1 = new Dog('Tomy', 5, '10kg');
D1.makeSound(); // woof
```

```
// C1 is an object of Cat calss
const C1 = new Cat('Rex', 5, '10kg');
C1.makeSound(); // Mew
```

JS OOP Encapsulation

Encapsulation is a restriction mechanism making accessing the data impossible without using special methods dedicated for this.

```
//child class
class Cat extends Animal{
    #Weight;  // mark as private
    constructor(name, age, weight){
        super(name, age);
        this.#Weight = weight
    }
};
```

```
// child class
class Dog extends Animal{
    Weight; // public property
    constructor(name, age, weight){
        super(name, age);
        this.Weight = weight;
    }
};
```

```
const C1 = new Cat(‘Rex', 5, '10kg’);

console.log(C1.Weight);  // undefined
```

```
const D1 = new Dog('Tomy', 5, '10kg’);

console.log(D1.Weight);  // 10kg
```

JS OOP Access Private Properties

We can access the private properties, using methods.

```
class Cat extends Animal{
    #Weight; // Mark as private
    constructor(name, age, weight){
        super(name, age);
        this.#Weight = weight;
    }
    getWeight(){
        return this.#Weight;
    }
};
```

```
const c1 = new Cat('mew', 5, '10kg’);

console.log(c1.getWeight()); // 10kg
```

JS OOP Getter & setter

In JavaScript, getters and setters allows you to define Object Accessors.

`get` for getter using to getting values and `set` for setter using to setting values.

```
class Cat extends Animal{
    #Weight;
    constructor(name, age, weight){
        super(name, age);
        this.#Weight = weight;
    }
    get weight(){
        return this.#Weight;
    }
}; // getter & setter used as property NOT method.
```

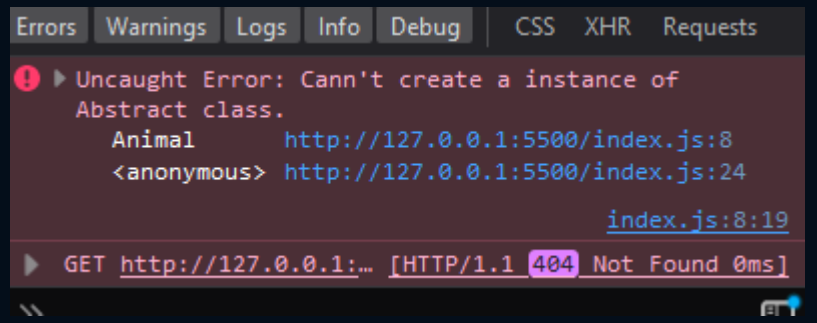
```
const c1 = new Cat('mew', 5, '10kg’);

console.log(c1.weight); // 10kg
```

JS OOP Abstract

Abstract class is a class which can’t make any object.

```
// parent class is abstracted
class Animal {
    constructor(name, age){
        this.Name = name;
        this.Age = age;
        //Make abstact class
        if(this.constructor == Animal){
            throw new Error(`Cann't create a instance of Abstract class.`);
        }
    }
};
```

```
// trying to create an object of Animal class
const A1 = new Animal('pk', 29);
```



CodeMaster Noyon