JS

Classes

#JavaScript Notes

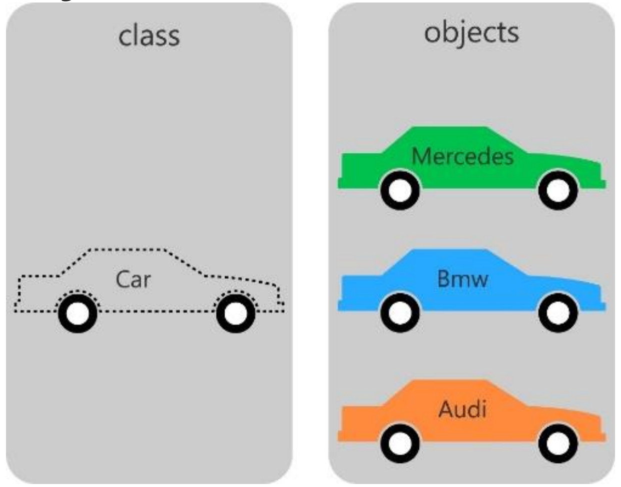
Classes

Class

- Classes are one of the features introduced in the **ES6** version of JavaScript.
- A class is a blueprint for the object. You can create an object from the class.

Class :

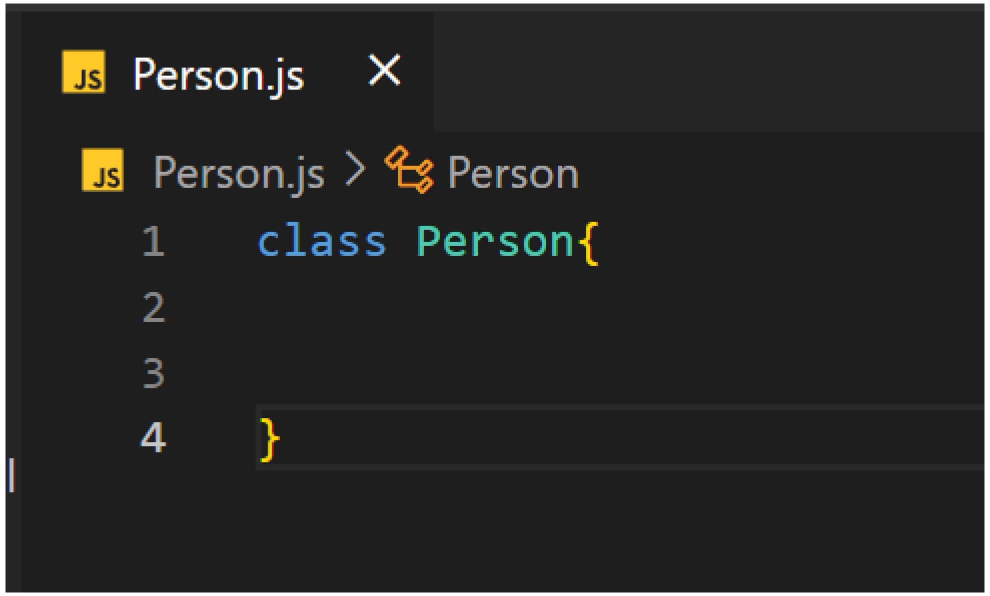
- Class is a programmer-defined data type, which includes local methods and local variables.
- Class is a collection of objects. Object has properties and behavior.
- First we have to define a class, where **class name should be same as filename.**
- When class is created, we can create any number of objects in that class. The object is created with the help of the new keyword.



How to create classes?

- In order to create a **class**, we group the code that handles a certain topic into one place.
  1. We declare the class with the **class** keyword.
  2. We write the name of the class and capitalize the first letter.
     - Example : Car
  3. If the class name contains more than one word, we capitalize each word. This is known as **upper camel case**
     - Example : Car_Class
  4. We circle the class body within curly braces. Inside the curly braces, we put our code.
     - Class Car {}

Class Example



```
class Person{


}
```

Constructor

Constructor

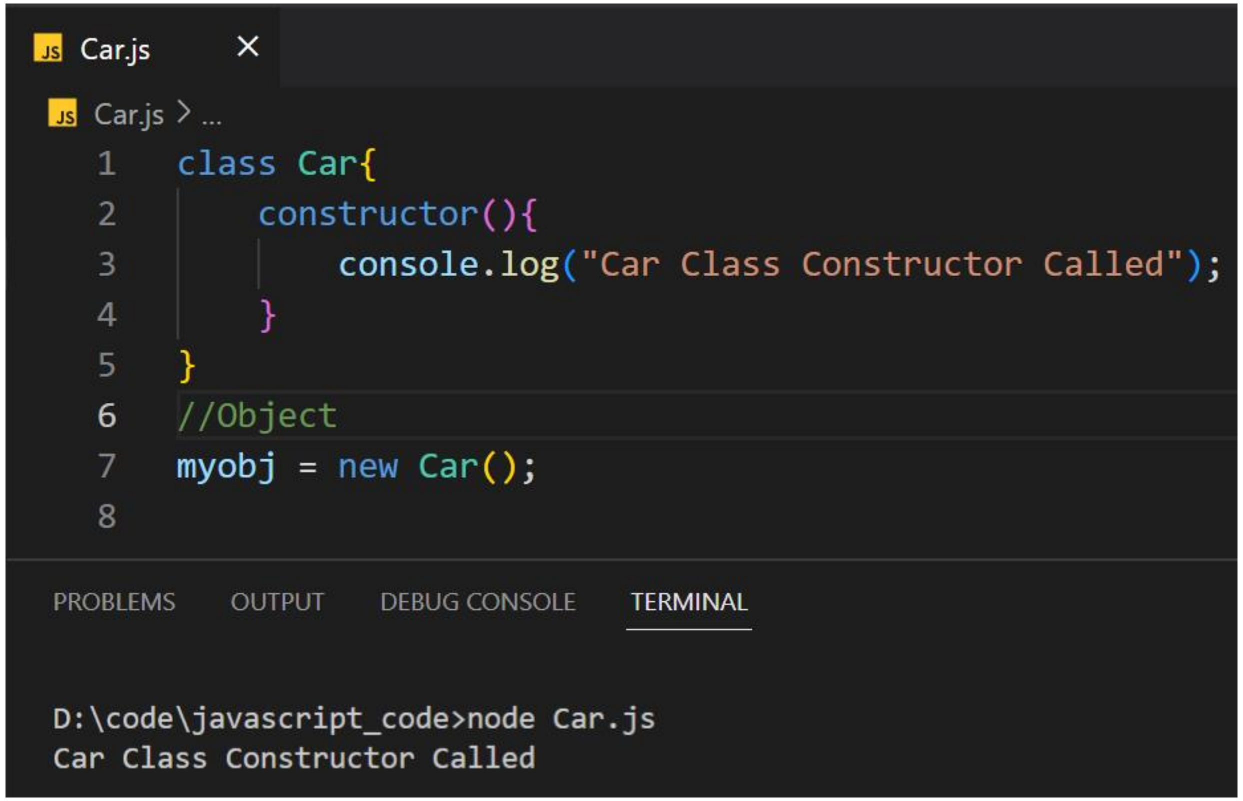
- A constructor is **a special function that creates and initializes an object instance of a class**.
- In JavaScript, a constructor gets called when an object is created using the new keyword.
- The purpose of a constructor is to create a new object and set values for any existing object properties

Object

How to create objects from a class?

- We can create several objects from the same class, with each object having its own set of properties.
- In order to work with a class, we need to create an object from it.
- In order to create an object, we use the **new** keyword.
- Example :

- bmw = new Car ();
- mercedes = new Car ();

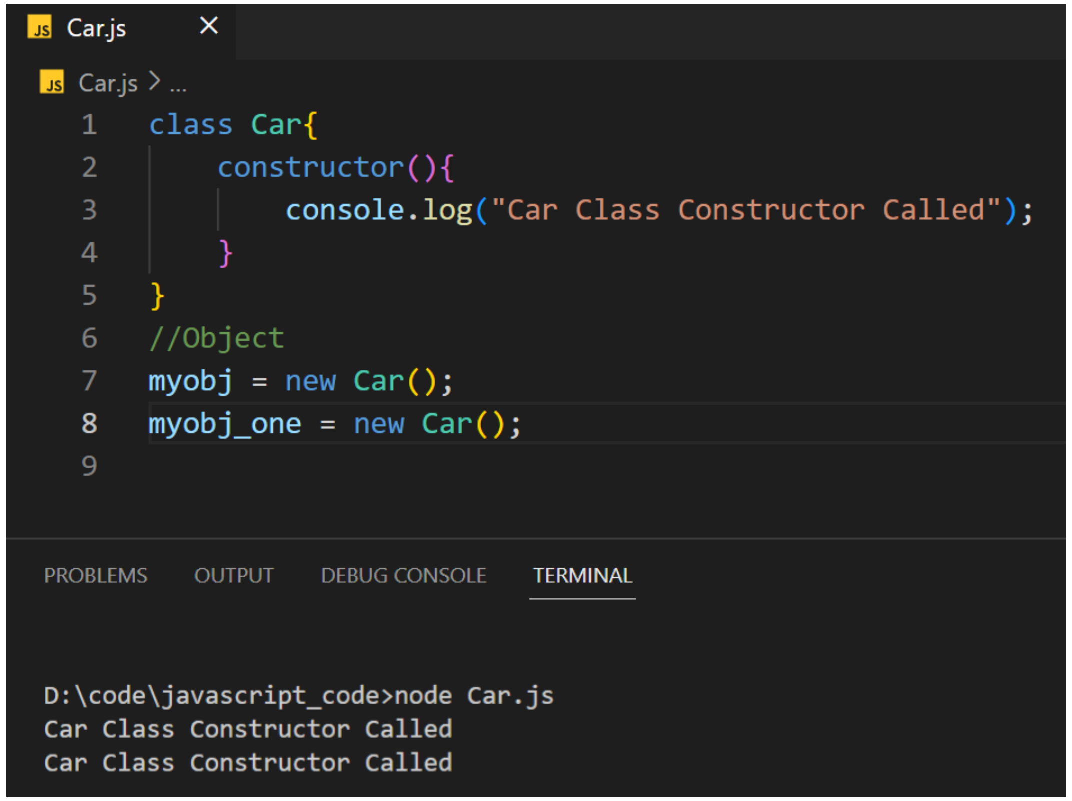


Car.js

```
class Car{
    constructor(){
        console.log("Car Class Constructor Called");
    }
}
//Object
myobj = new Car();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
D:\code\javascript_code>node Car.js
Car Class Constructor Called
```



Car.js

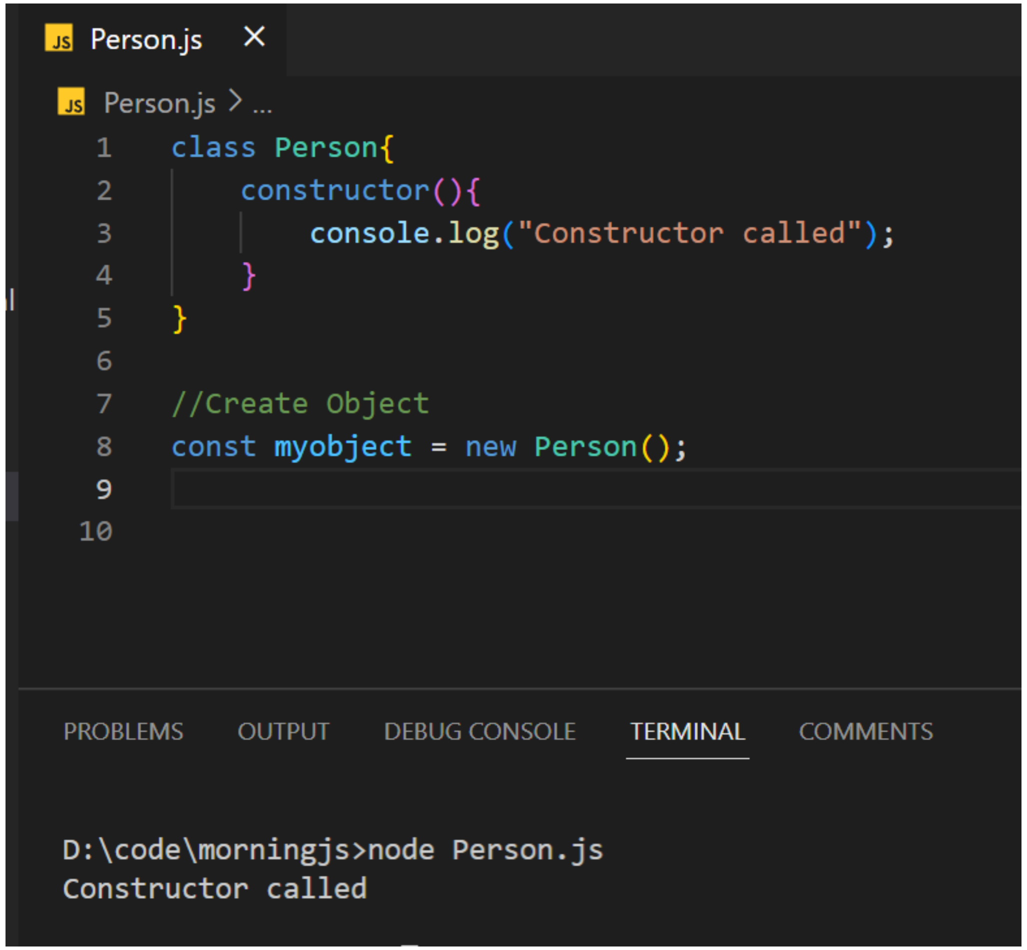
```
class Car{
    constructor(){
        console.log("Car Class Constructor Called");
    }
}
//Object
myobj = new Car();
myobj_one = new Car();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
D:\code\javascript_code>node Car.js
Car Class Constructor Called
Car Class Constructor Called
```

The this keyword

- The this keyword indicates that we use the class's own methods and properties, and allows us to have access to them within the class's scope.
- The this keyword allows us to approach the class properties and methods from within the class using the following syntax:



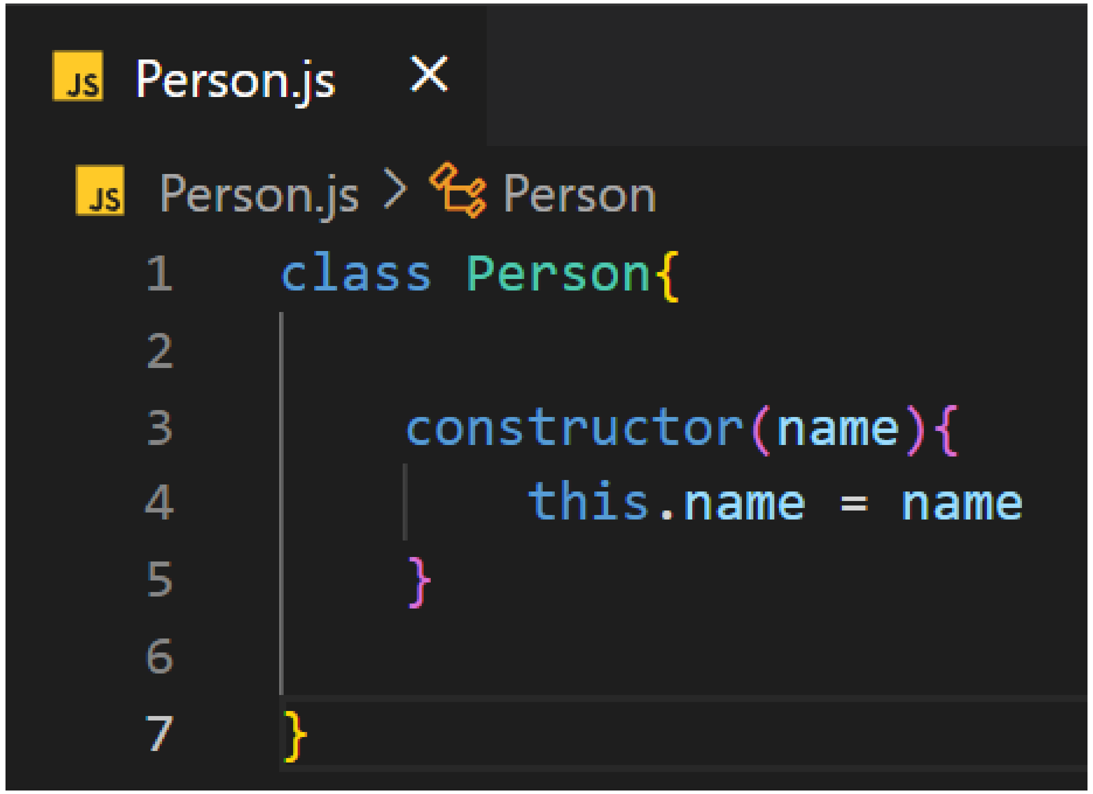
Person.js

Person.js > ...

```javascript
class Person{
    constructor(){
        console.log("Constructor called");
    }
}

//Create Object
const myobject = new Person();

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

```
D:\code\morningjs>node Person.js
Constructor called
```

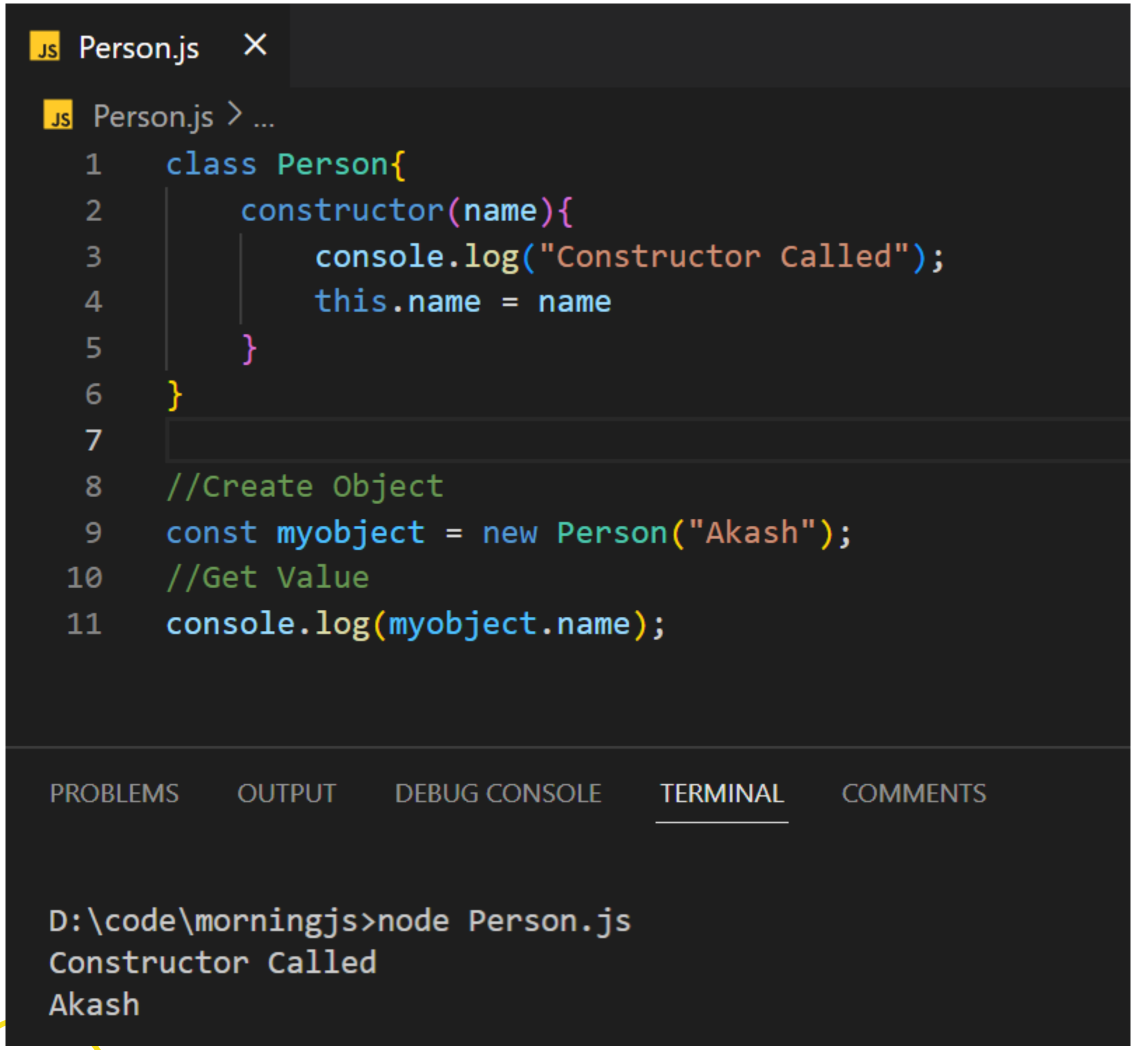


Person.js

Person.js > Person

```javascript
class Person{

    constructor(name){
        this.name = name
    }

}
```

Calling Member Variable using Object

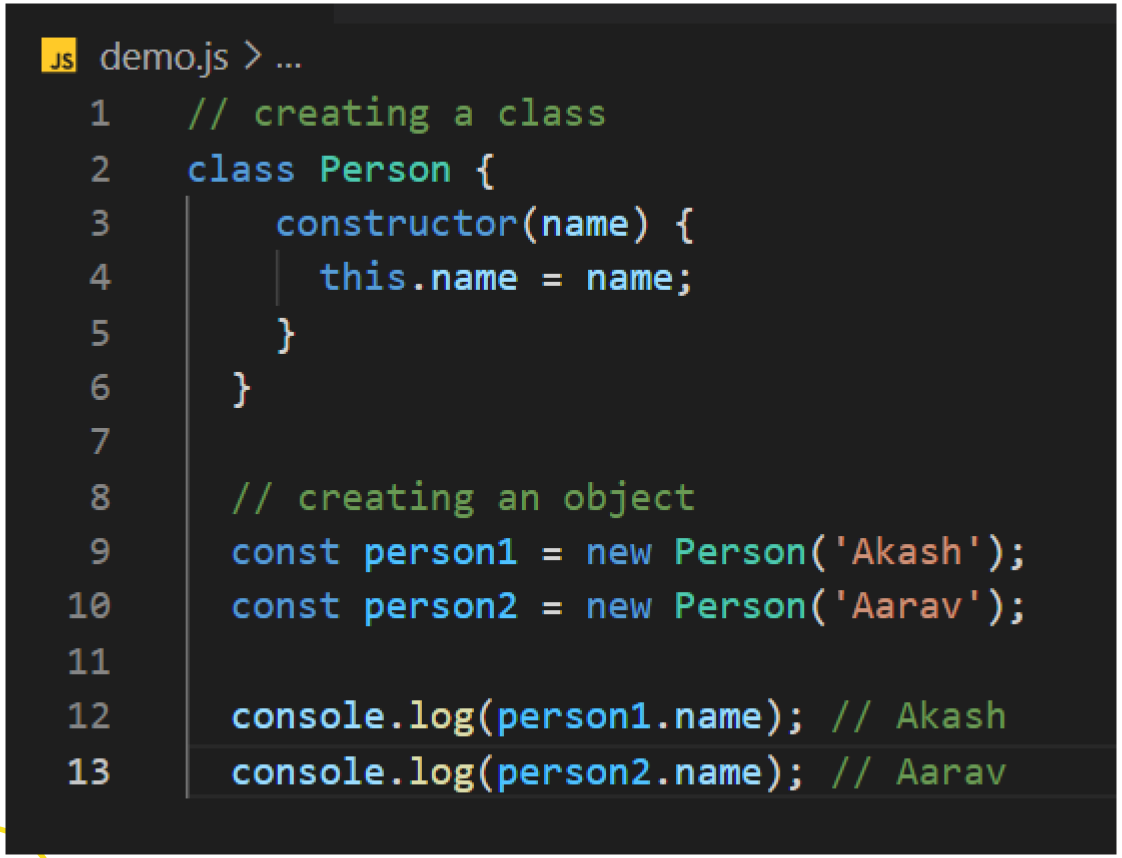


```
class Person{
    constructor(name){
        console.log("Constructor Called");
        this.name = name
    }
}

//Create Object
const myobject = new Person("Akash");
//Get Value
console.log(myobject.name);
```

```
D:\code\morningjs>node Person.js
Constructor Called
Akash
```

Creating JavaScript Class with Multiple Objecy

- The constructor() method inside a class gets called automatically each time an object is created.



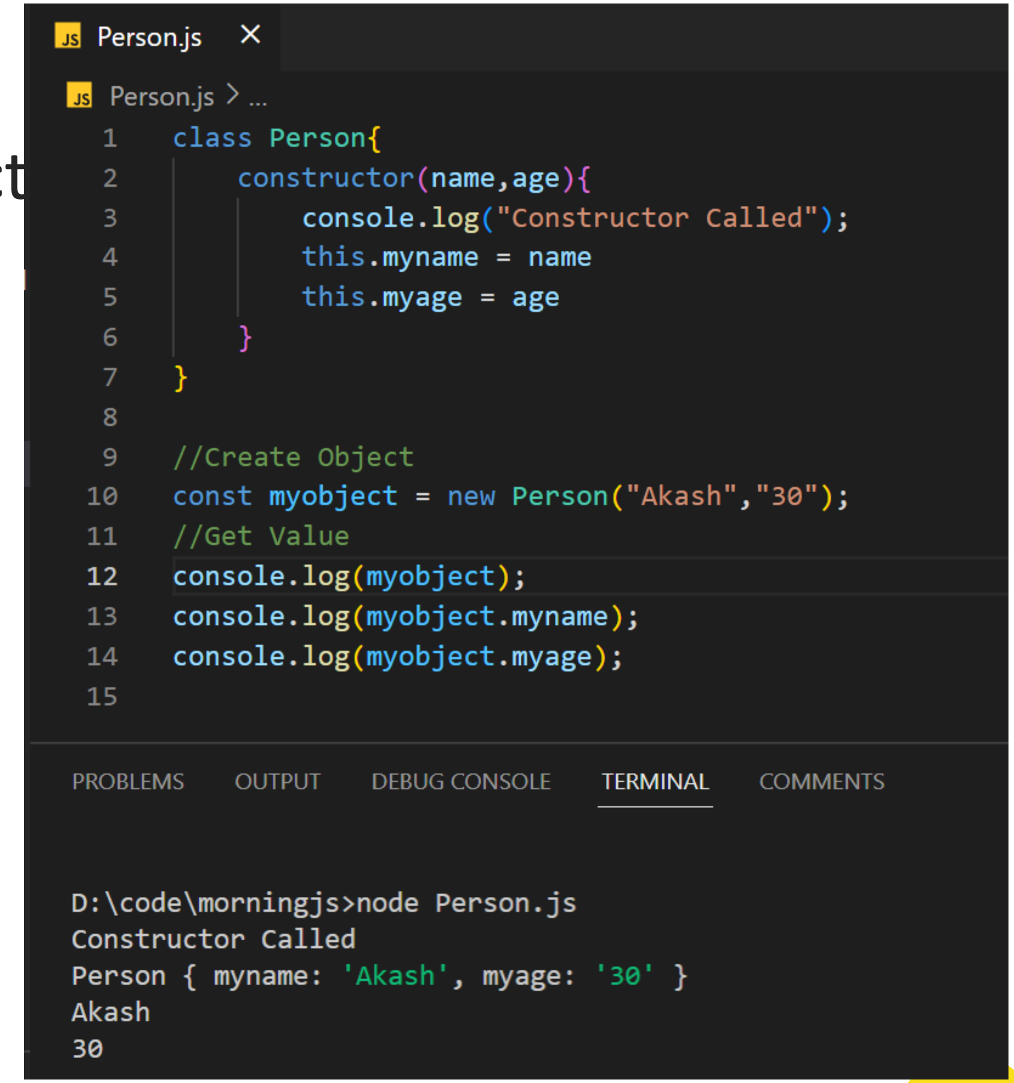
demo.js > ...

```
// creating a class
class Person {
    constructor(name) {
      this.name = name;
    }
  }

  // creating an object
  const person1 = new Person('Akash');
  const person2 = new Person('Aarav');

  console.log(person1.name); // Akash
  console.log(person2.name); // Aarav
```

```
// creating a class
class Person {
   constructor(name) {
     this.name = name;
   }
  }

 // creating an object
 const person1 = new Person('Akash');
 const person2 = new Person('Aarav');

 console.log(person1.name); // Akash
 console.log(person2.name); // Aarav
```

Object with Multiple Parameter

- We can pass multiple parameter in Object
- We can access member variable using object
  - Object.variablename

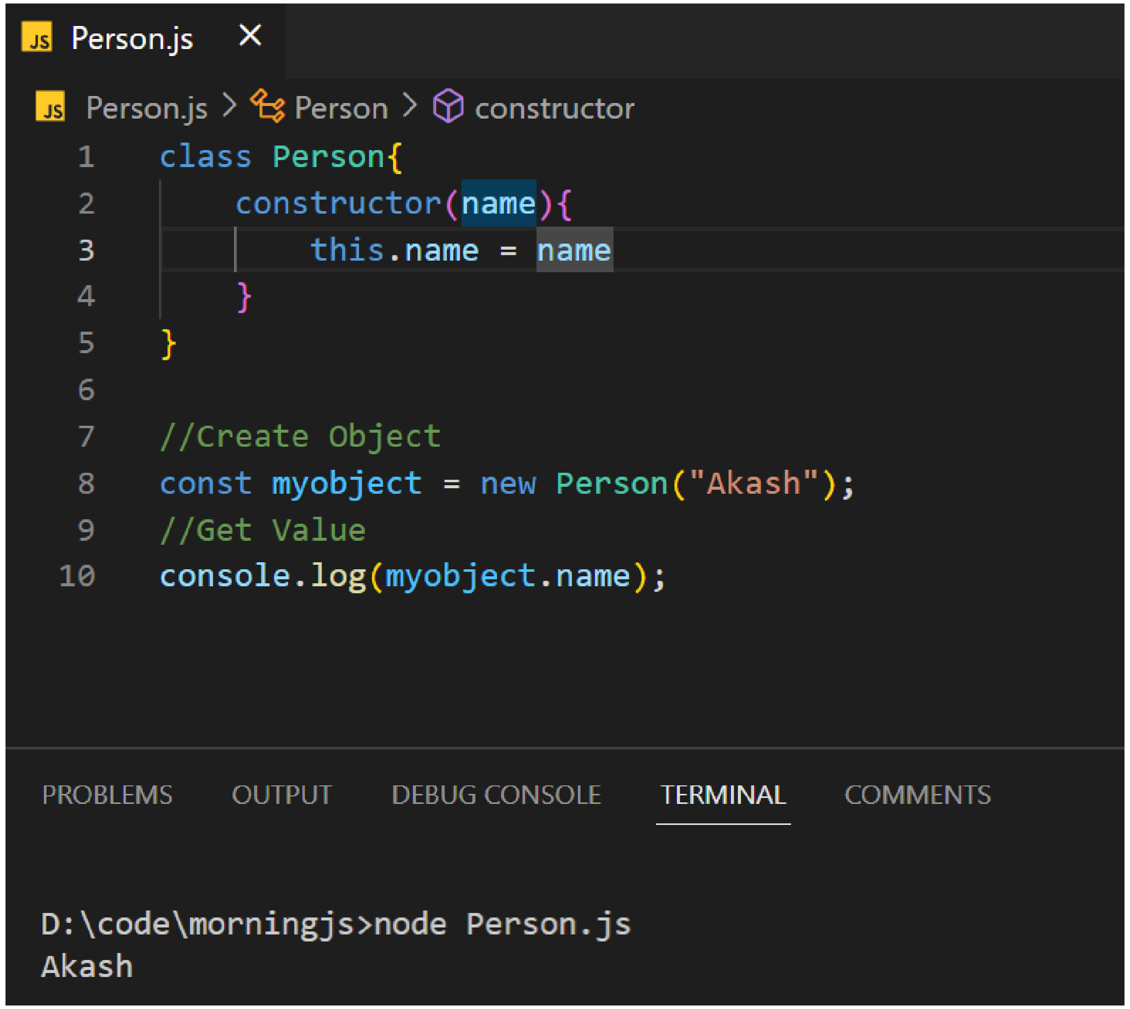


```javascript
class Person{
    constructor(name,age){
        console.log("Constructor Called");
        this.myname = name
        this.myage = age
    }
}

//Create Object
const myobject = new Person("Akash","30");
//Get Value
console.log(myobject);
console.log(myobject.myname);
console.log(myobject.myage);
```

```
D:\code\morningjs>node Person.js
Constructor Called
Person { myname: 'Akash', myage: '30' }
Akash
30
```

Access Property Outside Class Using Object

- We can access variable
  - Objectname.variablename



```javascript
class Person{
    constructor(name){
        this.name = name
    }
}

//Create Object
const myobject = new Person("Akash");
//Get Value
console.log(myobject.name);
```

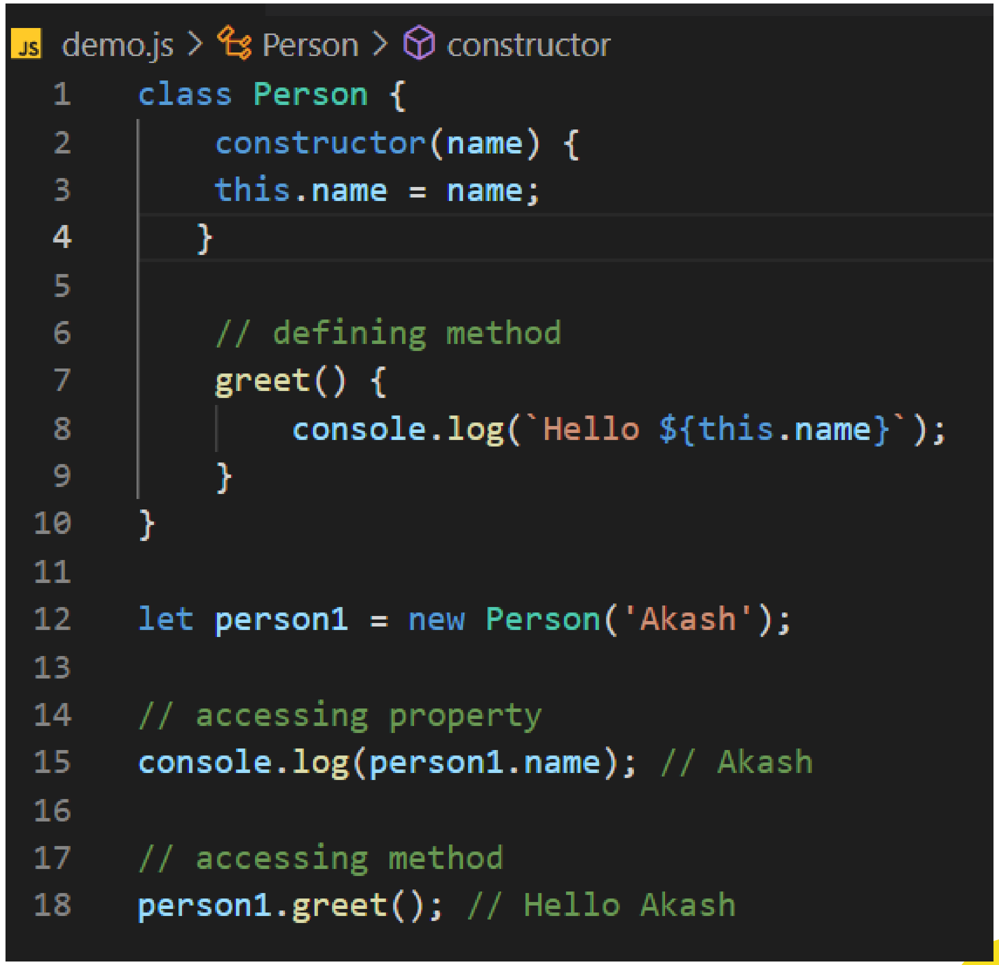
```
D:\code\morningjs>node Person.js
Akash
```

Class & Method

Javascript Class Methods

- To access the method of an object, you need to call the method using its name followed by ().

```
class Person {
   constructor(name) {
   this.name = name;
  }

   // defining method
   greet() {
      console.log(`Hello ${this.name}`);
   }
}

let person1 = new Person('Akash');

// accessing property
console.log(person1.name); // Akash

// accessing method
person1.greet(); // Hello Akash
```

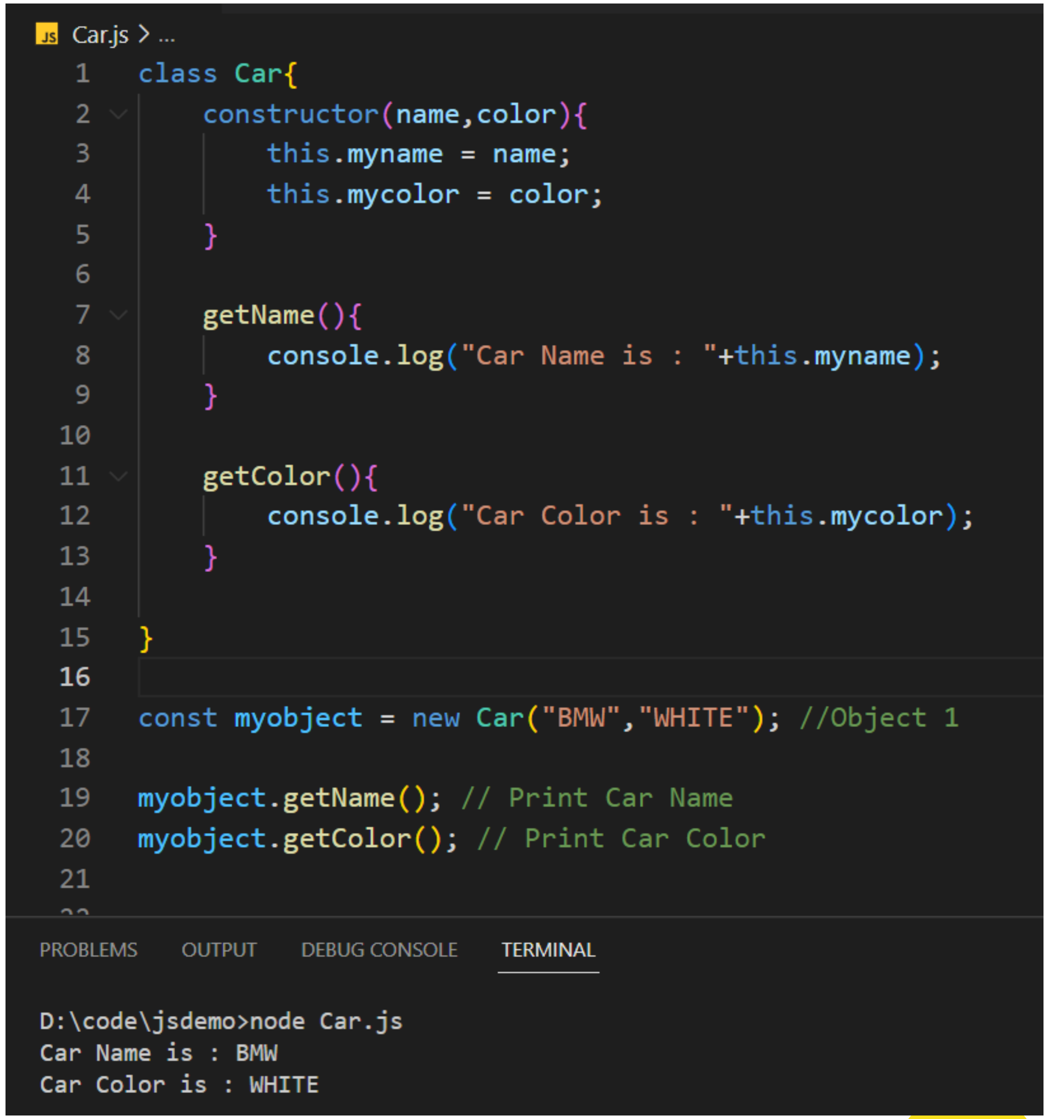

demo.js > Person > constructor

```
class Person {
    constructor(name) {
    this.name = name;
   }

    // defining method
    greet() {
        console.log(`Hello ${this.name}`);
    }
}

let person1 = new Person('Akash');

// accessing property
console.log(person1.name); // Akash

// accessing method
person1.greet(); // Hello Akash
```

Class & Method

- We can define various method and we can access using objectname.method

```
class Car{
   constructor(name,color){
      this.myname = name;
      this.mycolor = color;
   }
   getName(){
      console.log("Car Name is : "+this.myname);
   }
   getColor(){
      console.log("Car Color is : "+this.mycolor);
   }

}
const myobject = new Car("BMW","WHITE"); //Object 1

myobject.getName(); // Print Car Name
myobject.getColor(); // Print Car Color
```

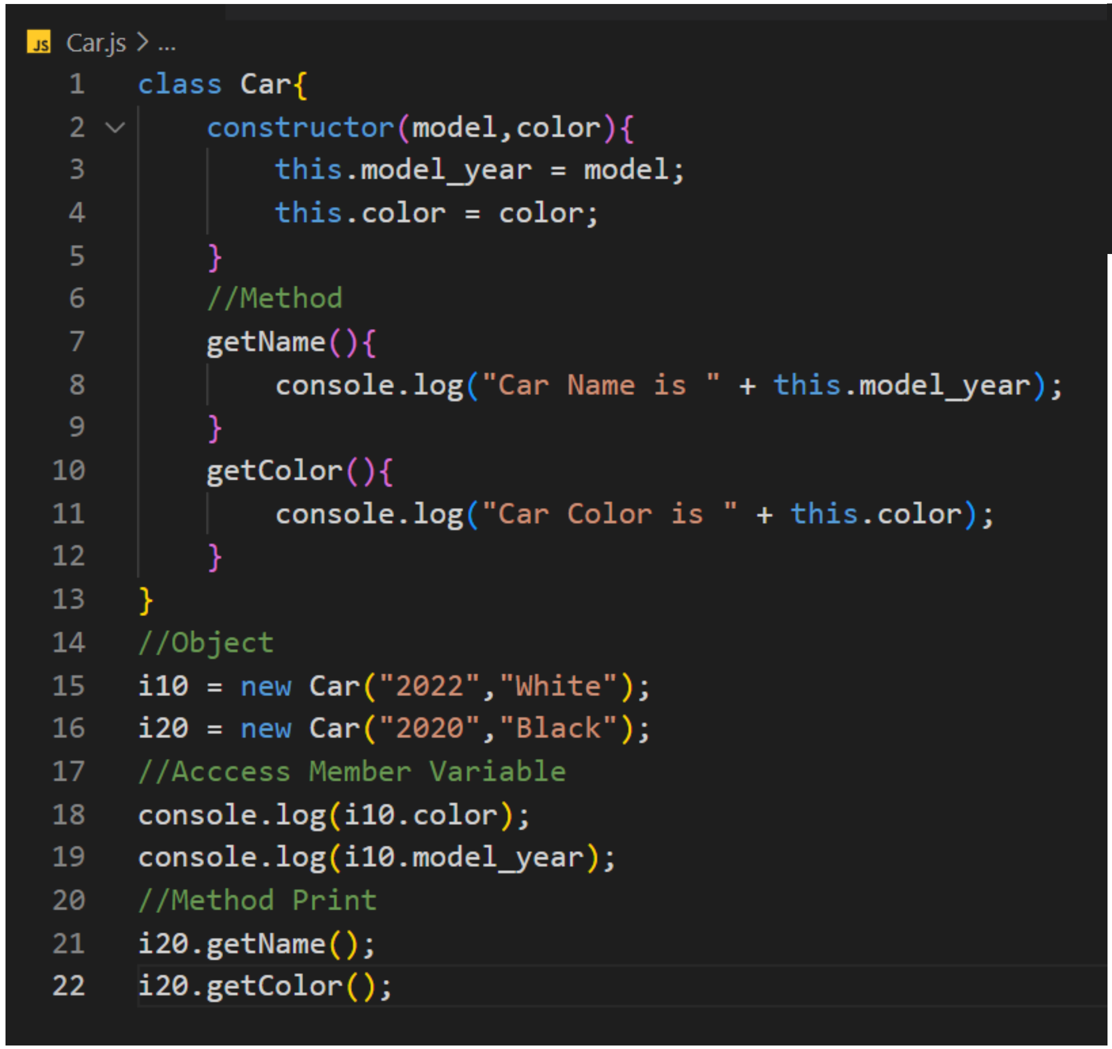


Car.js > ...

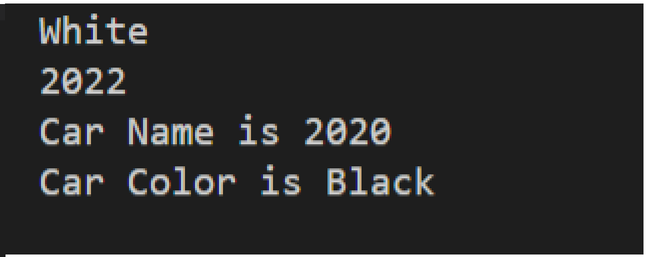
```
class Car{
    constructor(name,color){
        this.myname = name;
        this.mycolor = color;
    }

    getName(){
        console.log("Car Name is : "+this.myname);
    }

    getColor(){
        console.log("Car Color is : "+this.mycolor);
    }

}

const myobject = new Car("BMW","WHITE"); //Object 1

myobject.getName(); // Print Car Name
myobject.getColor(); // Print Car Color

```

```
D:\code\jsdemo>node Car.js
Car Name is : BMW
Car Color is : WHITE
```

Revision

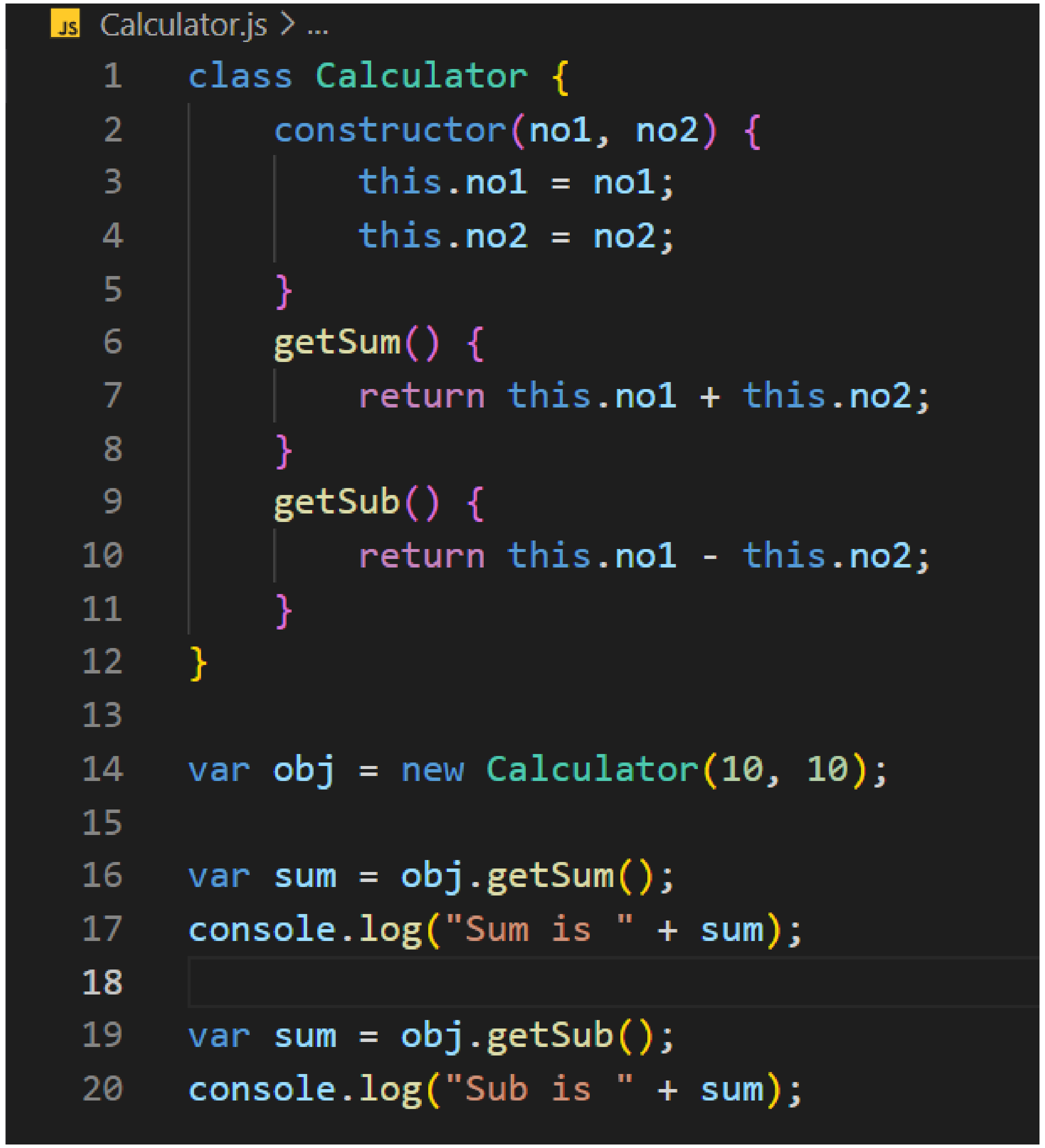


```
Car.js > ...
class Car{
    constructor(model,color){
        this.model_year = model;
        this.color = color;
    }
    //Method
    getName(){
        console.log("Car Name is " + this.model_year);
    }
    getColor(){
        console.log("Car Color is " + this.color);
    }
}
//Object
i10 = new Car("2022","White");
i20 = new Car("2020","Black");
//Acccess Member Variable
console.log(i10.color);
console.log(i10.model_year);
//Method Print
i20.getName();
i20.getColor();
```



```
White
2022
Car Name is 2020
Car Color is Black
```

```
class Car{
  constructor(model,color){
    this.model_year = model;
    this.color = color;
  }
  //Method
  getName(){
    console.log("Car Name is " + this.model_year);
  }
  getColor(){
    console.log("Car Color is " + this.color);
  }
}
//Object
i10 = new Car("2022","White");
i20 = new Car("2020","Black");
//Acccess Member Variable
console.log(i10.color);
console.log(i10.model_year);
//Method Print
i20.getName();
i20.getColor();
```


Calculator.js > ...

```javascript
class Calculator {
    constructor(no1, no2) {
        this.no1 = no1;
        this.no2 = no2;
    }
    getSum() {
        return this.no1 + this.no2;
    }
    getSub() {
        return this.no1 - this.no2;
    }
}

var obj = new Calculator(10, 10);

var sum = obj.getSum();
console.log("Sum is " + sum);

var sum = obj.getSub();
console.log("Sub is " + sum);
```

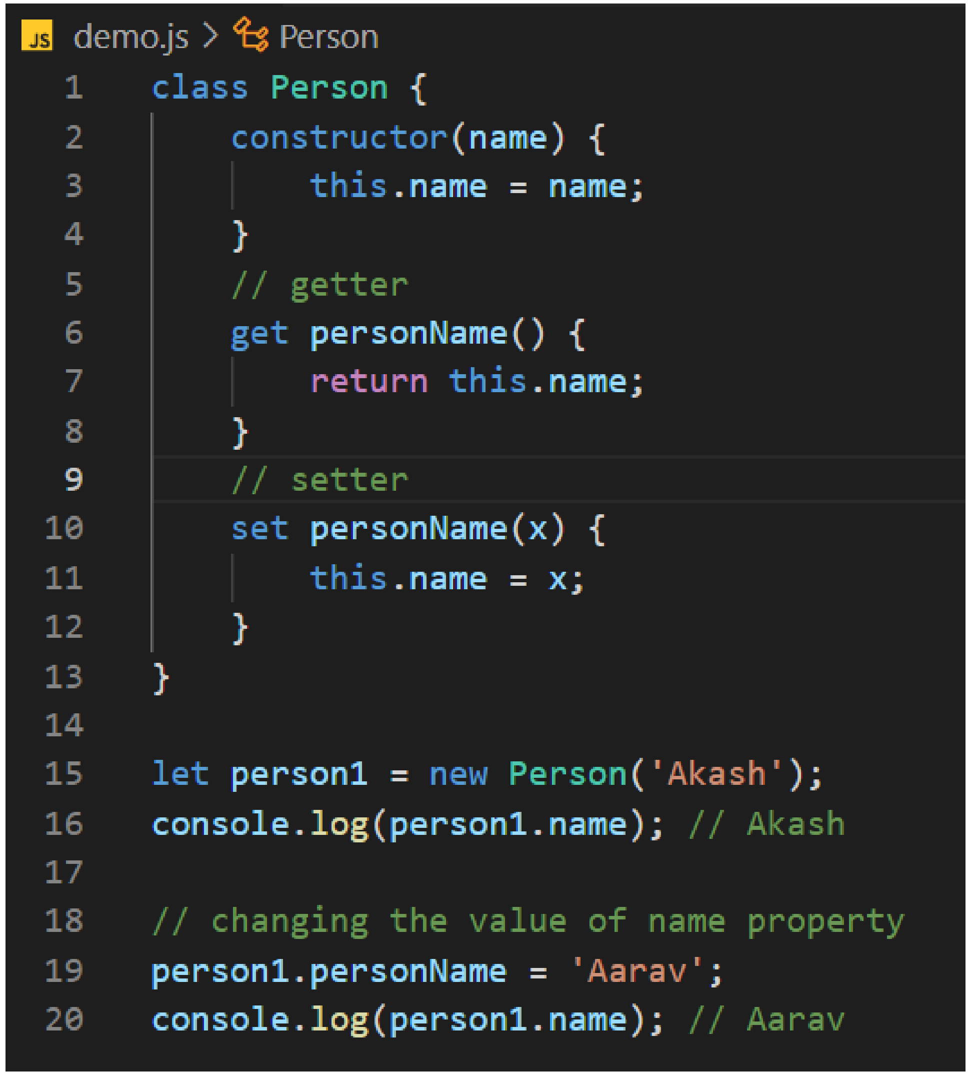
Getter and Setter

JS

Getters and Setters

- In JavaScript, getter methods get the value of an object and setter methods set the value of an object.
- use the get keyword for getter methods and set for setter methods.

Getters and Setters

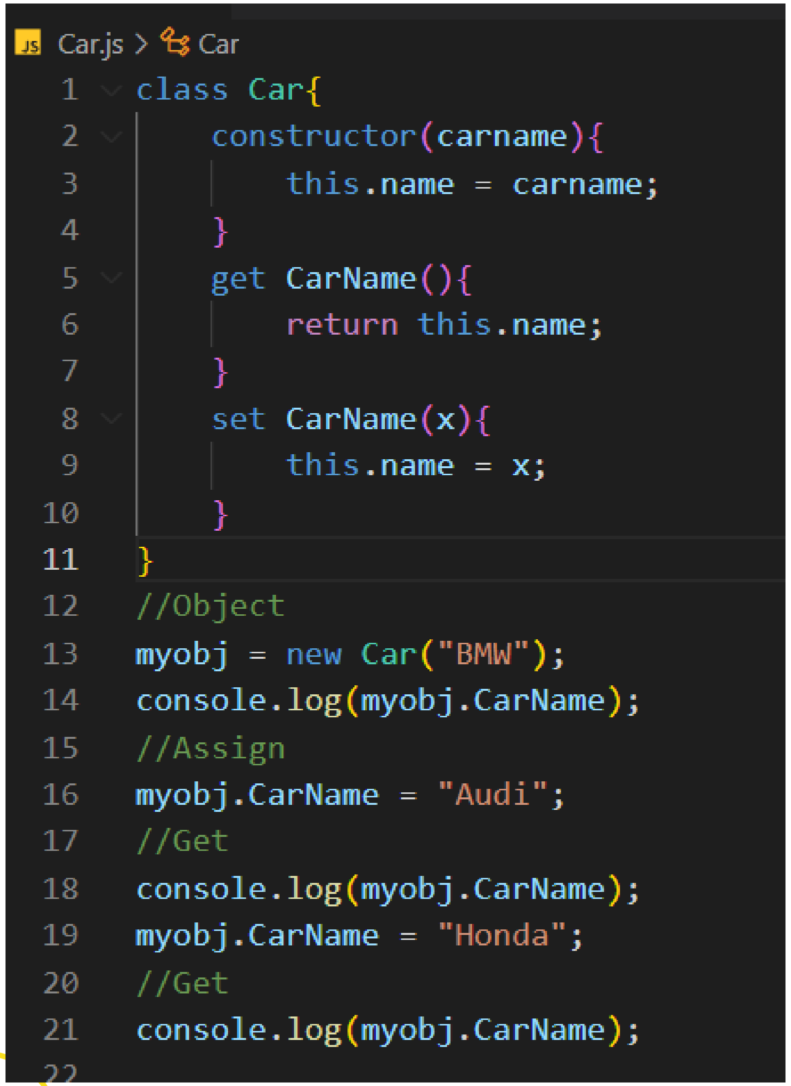


demo.js > Person

```
class Person {
    constructor(name) {
        this.name = name;
    }
    // getter
    get personName() {
        return this.name;
    }
    // setter
    set personName(x) {
        this.name = x;
    }
}

let person1 = new Person('Akash');
console.log(person1.name); // Akash

// changing the value of name property
person1.personName = 'Aarav';
console.log(person1.name); // Aarav
```

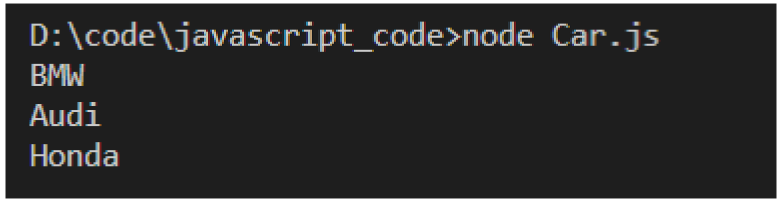
```
class Person {
    constructor(name) {
        this.name = name;
    }
    // getter
    get personName() {
        return this.name;
    }
    // setter
    set personName(x) {
        this.name = x;
    }
}

let person1 = new Person('Akash');
console.log(person1.name); // Akash

// changing the value of name property
person1.personName = 'Aarav';
console.log(person1.name); // Aarav
```

Get & Set



Car.js > Car

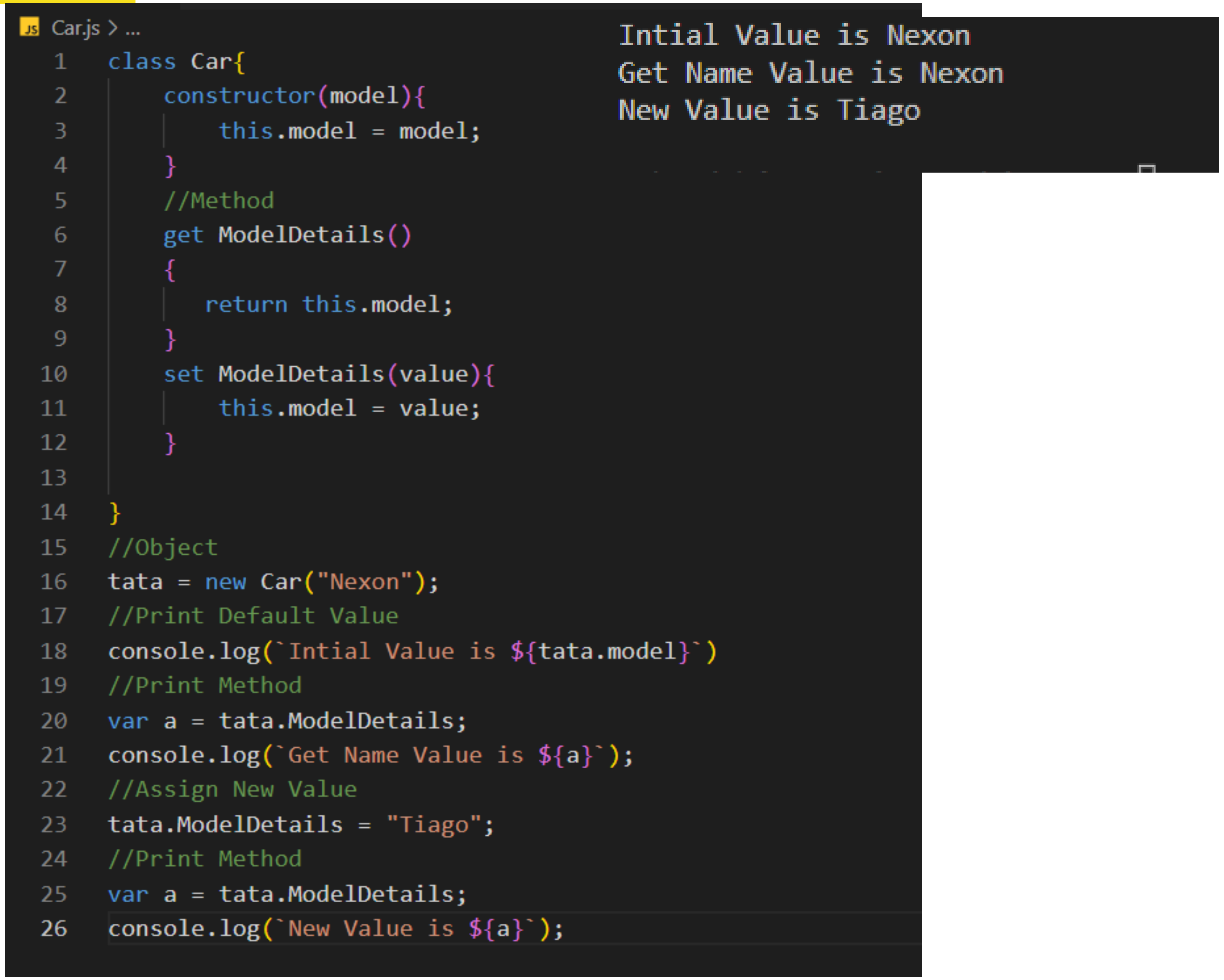
```
class Car{
    constructor(carname){
        this.name = carname;
    }
    get CarName(){
        return this.name;
    }
    set CarName(x){
        this.name = x;
    }
}
//Object
myobj = new Car("BMW");
console.log(myobj.CarName);
//Assign
myobj.CarName = "Audi";
//Get
console.log(myobj.CarName);
myobj.CarName = "Honda";
//Get
console.log(myobj.CarName);
```



```
D:\code\javascript_code>node Car.js
BMW
Audi
Honda
```

```
class Car{
   constructor(carname){
      this.name = carname;
   }
   get CarName(){
      return this.name;
   }
   set CarName(x){
      this.name = x;
   }
}
//Object
myobj = new Car("BMW");
console.log(myobj.CarName);
//Assign
myobj.CarName = "Audi";
//Get
console.log(myobj.CarName);
myobj.CarName = "Honda";
//Get
console.log(myobj.CarName);
```

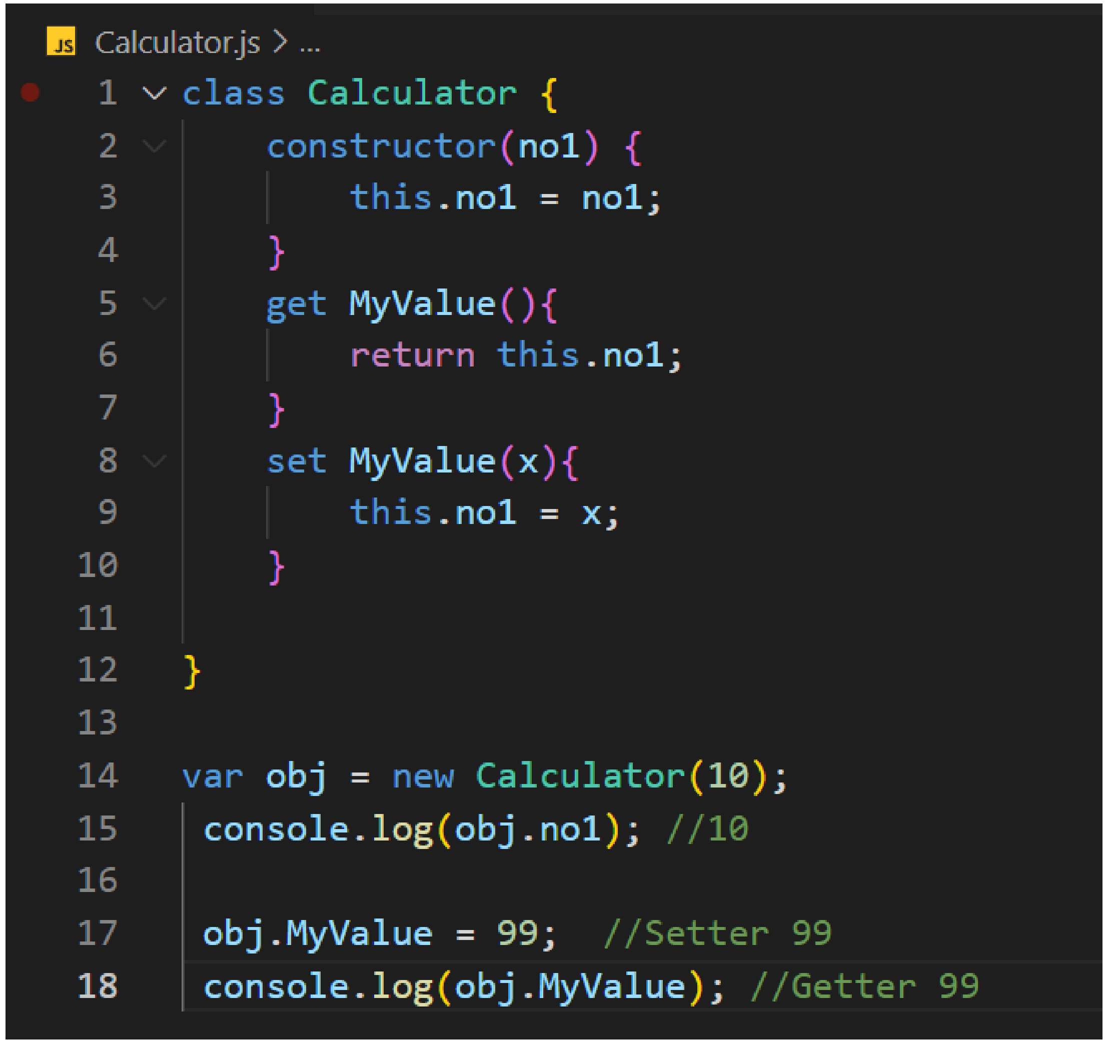
Example



```
Car.js > ...
class Car{
    constructor(model){
        this.model = model;
    }
    //Method
    get ModelDetails()
    {
      return this.model;
    }
    set ModelDetails(value){
        this.model = value;
    }

}
//Object
tata = new Car("Nexon");
//Print Default Value
console.log(`Intial Value is ${tata.model}`)
//Print Method
var a = tata.ModelDetails;
console.log(`Get Name Value is ${a}`);
//Assign New Value
tata.ModelDetails = "Tiago";
//Print Method
var a = tata.ModelDetails;
console.log(`New Value is ${a}`);
```

```
Intial Value is Nexon
Get Name Value is Nexon
New Value is Tiago
```

```
class Car{
   constructor(model){
      this.model = model;
   }
   //Method
   get ModelDetails()
   {
     return this.model;
   }
   set ModelDetails(value){
      this.model = value;
   }
}
//Object
tata = new Car("Nexon");
//Print Default Value
console.log(`Intial Value is ${tata.model}`)
//Print Method
var a = tata.ModelDetails;
console.log(`Get Name Value is ${a}`);
//Assign New Value
tata.ModelDetails = "Tiago";
//Print Method
var a = tata.ModelDetails;
console.log(`New Value is ${a}`);
```


Calculator.js > ...

```
class Calculator {
    constructor(no1) {
        this.no1 = no1;
    }
    get MyValue(){
        return this.no1;
    }
    set MyValue(x){
        this.no1 = x;
    }

}

var obj = new Calculator(10);
 console.log(obj.no1); //10

 obj.MyValue = 99;  //Setter 99
 console.log(obj.MyValue); //Getter 99
```

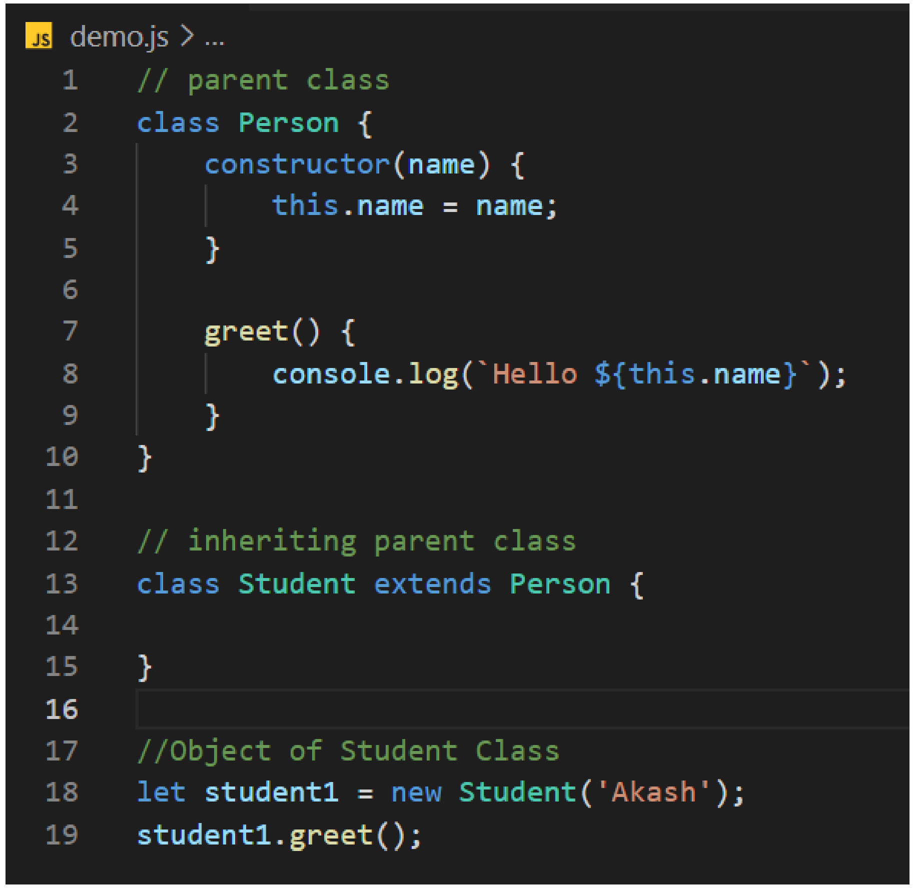
JavaScript Class Inheritance

Class Inheritance

- When the properties and the methods of the parent class are accessed by the child class, we call the concept has inheritance.
- To use class inheritance, you use the **extends** keyword.
- The child class can inherit the parent method and give own method implementation, this property is called overridden method.
- Inheritance enables you to define a class that takes all the functionality from a parent class and allows you to add more.
- Using class inheritance, a class can inherit all the methods and properties of another class.
- Inheritance is a useful feature that allows code reusability.

Class Inheritance

- Student class inherits all the methods and properties of the Person class. Hence, the Student class will now have the name property and the greet() method.

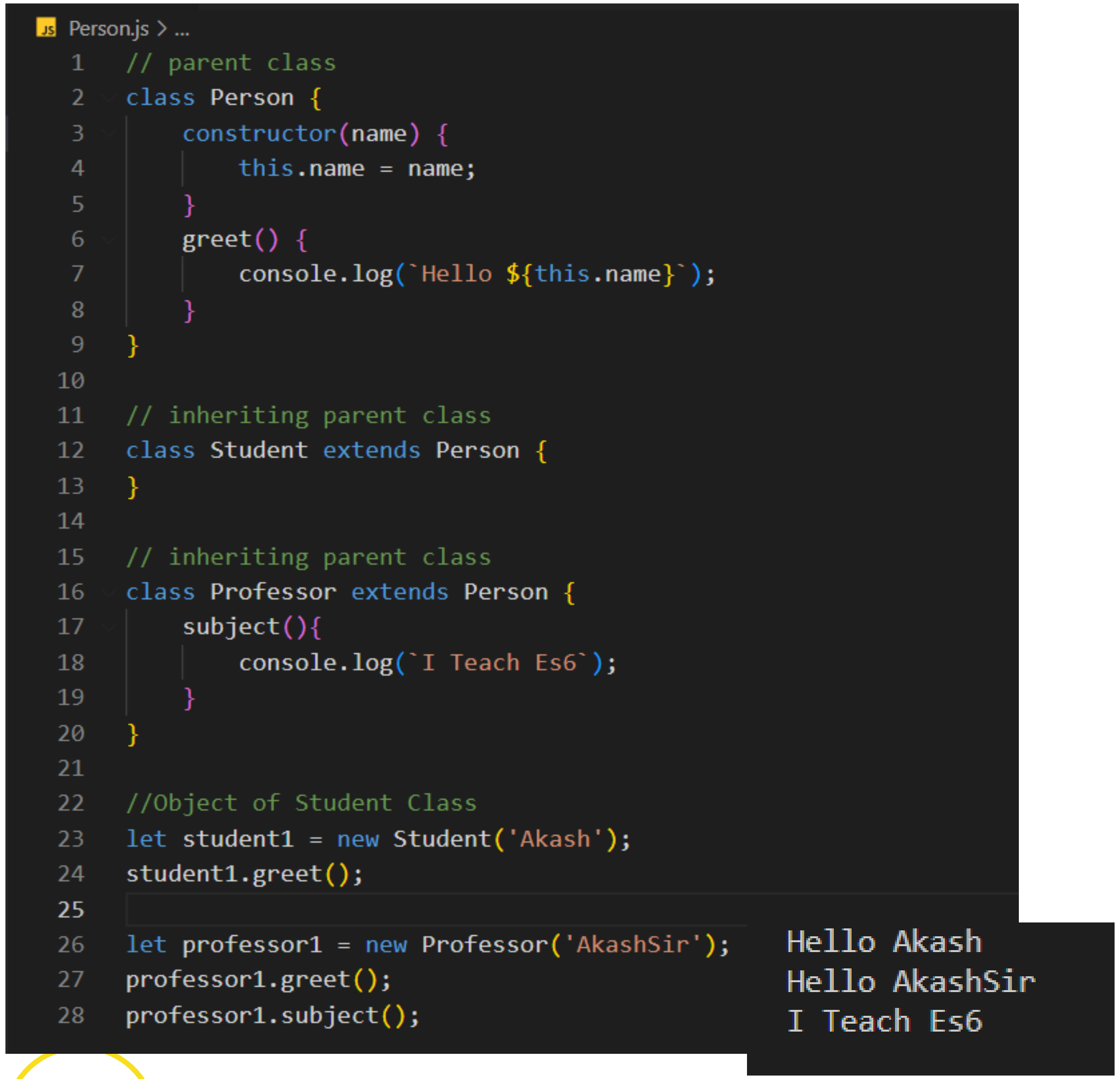


demo.js > ...

```javascript
// parent class
class Person {
    constructor(name) {
        this.name = name;
    }

    greet() {
        console.log(`Hello ${this.name}`);
    }
}

// inheriting parent class
class Student extends Person {

}

//Object of Student Class
let student1 = new Student('Akash');
student1.greet();
```

```javascript
// parent class
class Person {
   constructor(name) {
      this.name = name;
   }

   greet() {
      console.log(`Hello ${this.name}`);
   }
}

// inheriting parent class
class Student extends Person {

}

//Object of Student Class
let student1 = new Student('Akash');
student1.greet();
```

Extends to Multiple Class



Person.js > ...

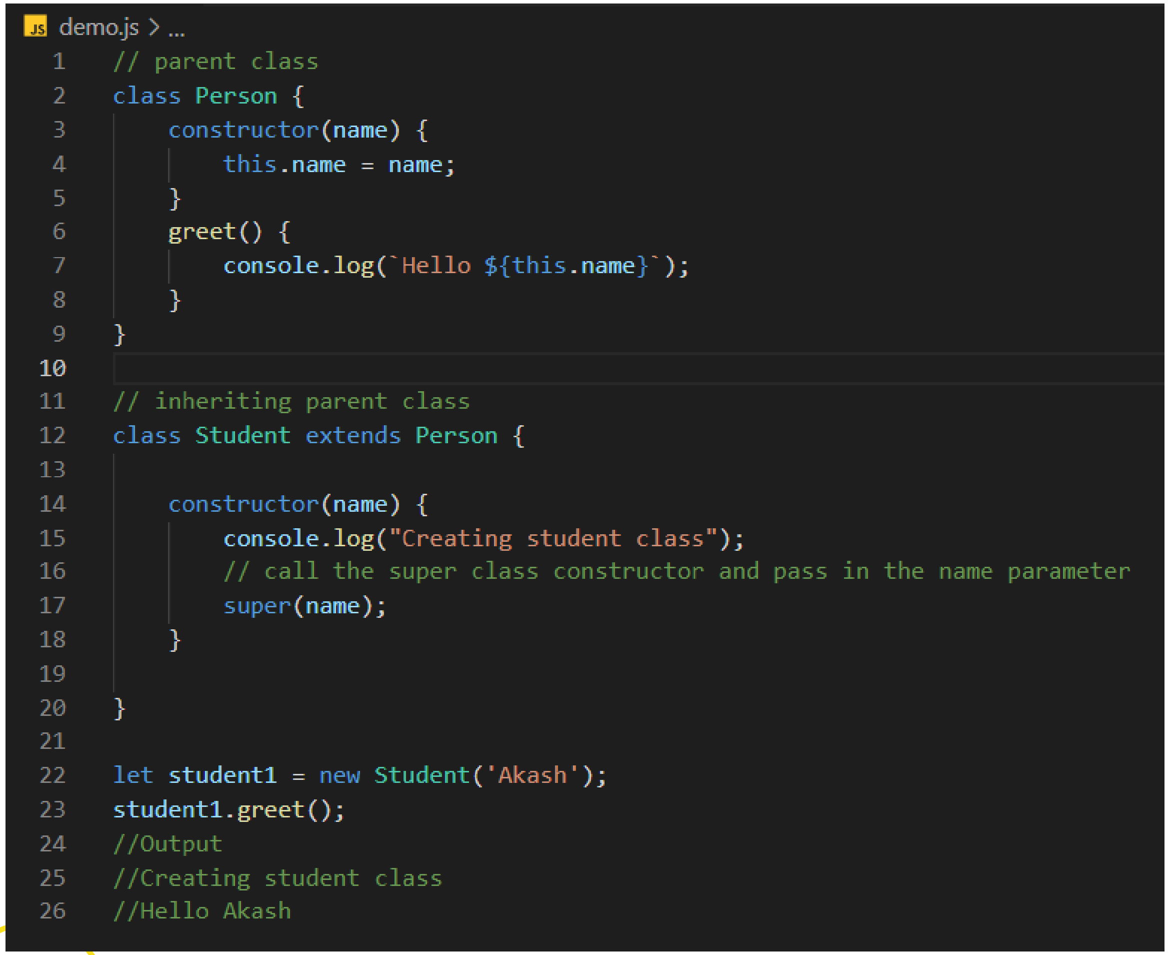
```
// parent class
class Person {
    constructor(name) {
        this.name = name;
    }
    greet() {
        console.log(`Hello ${this.name}`);
    }
}

// inheriting parent class
class Student extends Person {
}

// inheriting parent class
class Professor extends Person {
    subject(){
        console.log(`I Teach Es6`);
    }
}

//Object of Student Class
let student1 = new Student('Akash');
student1.greet();

let professor1 = new Professor('AkashSir');
professor1.greet();
professor1.subject();
```

```
Hello Akash
Hello AkashSir
I Teach Es6
```

```
// parent class
class Person {
   constructor(name) {
      this.name = name;
   }
   greet() {
      console.log(`Hello ${this.name}`);
   }
}

// inheriting parent class
class Student extends Person {
}

// inheriting parent class
class Professor extends Person {
   subject(){
      console.log(`I Teach Es6`);
   }
}

//Object of Student Class
let student1 = new Student('Akash');
student1.greet();

let professor1 = new Professor('AkashSir');
professor1.greet();
professor1.subject();
```

JavaScript super() keyword

- The super keyword used inside a child class denotes its parent class.

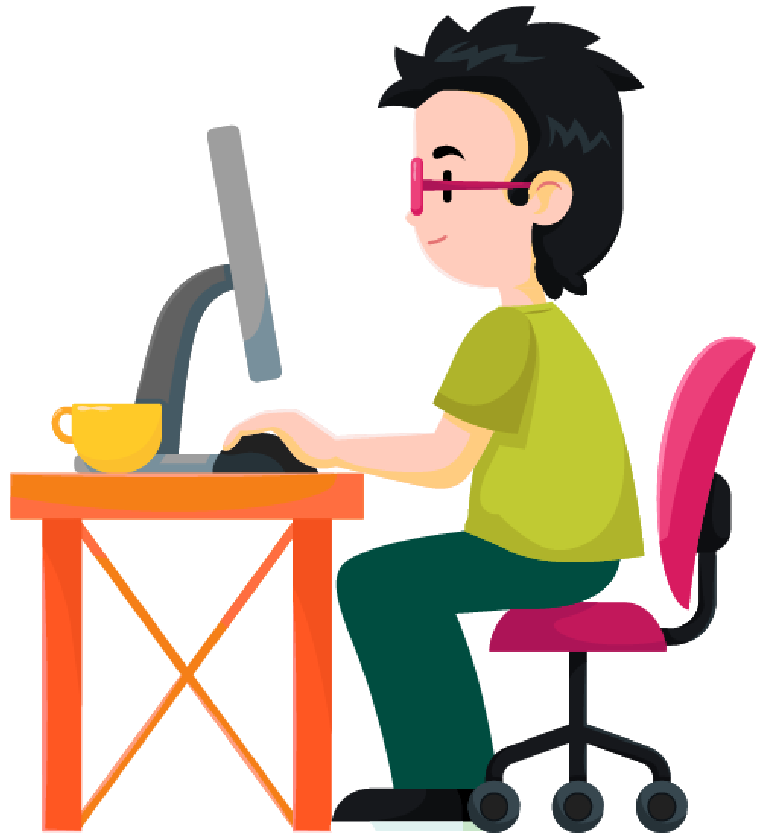


```javascript
// parent class
class Person {
    constructor(name) {
        this.name = name;
    }
    greet() {
        console.log(`Hello ${this.name}`);
    }
}

// inheriting parent class
class Student extends Person {

    constructor(name) {
        console.log("Creating student class");
        // call the super class constructor and pass in the name parameter
        super(name);
    }

}

let student1 = new Student('Akash');
student1.greet();
//Output
//Creating student class
//Hello Akash
```

```javascript
// parent class
class Person {
   constructor(name) {
      this.name = name;
   }
   greet() {
      console.log(`Hello ${this.name}`);
   }
}

// inheriting parent class
class Student extends Person {

   constructor(name) {
      console.log("Creating student class");
      // call the super class constructor and pass in the name parameter
      super(name);
   }

}

let student1 = new Student('Akash');
student1.greet();
//Output
//Creating student class
//Hello Akash
```

Get Exclusive
Video Tutorials

www.aptutorials.com
https://www.youtube.com/user/Akashtips



Get More Details

www.akashsir.com

If You Liked It !

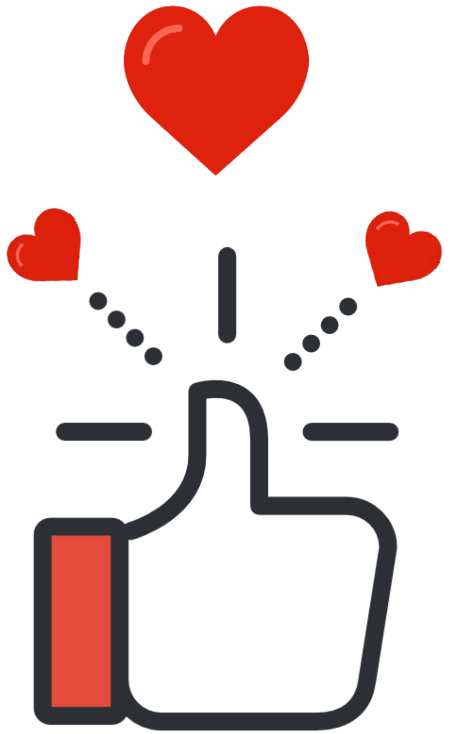
Rating Us Now

Just Dial

https://www.justdial.com/Ahmedabad/Akash-Technolabs-Navrangpura-Bus-Stop-Navrangpura/079PXX79-XX79-170615221520-S5C4_BZDET

Sulekha

https://www.sulekha.com/akash-technolabs-navrangpura-ahmedabad-contact-address/ahmedabad



Connect With Me

Akash Padhiyar
#AkashSir

www.akashsir.com
www.akashtechnolabs.com
www.akashpadhiyar.com
www.aptutorials.com

# Social Info

Akash.padhiyar

Akashpadhiyar

Akash_padhiyar

+91 99786-21654

#Akashpadhiyar
#aptutorials