Day :14 JavaScript Notes.

JavaScript Classes and Inheritance

In modern JavaScript (ES6+), **classes** provide a structured way to create reusable code through **objects** and support for **inheritance**. A class is essentially a blueprint for creating objects with specific properties and methods.

---

1. Class with Constructor

A constructor method is used to initialize object properties when the object is created using the new keyword.

```
class CarShowRoom {
    brand;
    color;
    Reviews;
    constructor(brand, color, Reviews) {
        this.brand = brand;
        this.color = color;
        this.Reviews = Reviews;
    }
    display() {
        console.log("Car Brand: " + this.brand);
        console.log("Car Color: " + this.color);
        console.log("Car Reviews: " + this.Reviews);
    }
}

let obj1 = new CarShowRoom("BMW", "Red", 4.5);
let obj2 = new CarShowRoom("Ford", "White", 3.5);
let obj3 = new CarShowRoom("Tata", "Black", 4.6);

obj1.display();
obj2.display();
obj3.display();
```

Each object holds its own values for brand, color, and reviews, and the display() method prints them.

---

2. Class without Constructor

You can also create a class **without a constructor** and define a method that takes parameters for setting and displaying values.

```
class CarShowRoom {
  brand;
  color;
  Reviews;

  display(brand, color, Reviews) {
    console.log("Car Brand: " + brand);
    console.log("Car Color: " + color);
    console.log("Car Reviews: " + Reviews);
  }
}

let obj = new CarShowRoom();
obj.display("BMW", "Red", 4.5);
```

This method works but does not store the values in the object for later use — it's mainly for one-time display.

}

---

3. Constructor Overloading and Inheritance

JavaScript **does not support constructor overloading natively**, but you can simulate it using **default values or conditional logic**. You can also create a child class that extends a parent class using the extends keyword.

```
class CarShowRoom {
    constructor(brand, color, Reviews) {
        this.brand = brand;
        this.color = color;
        this.Reviews = Reviews;
    }
    display() {
        console.log("Car Brand: " + this.brand);
        console.log("Car Color: " + this.color);
        console.log("Car Reviews: " + this.Reviews);
    }
}
class Car extends CarShowRoom {
    constructor(brand, color) {
        super(brand, color, 0); // Calls parent constructor with default review
    }

    setReview(Reviews) {
        this.Reviews = Reviews;
    }

    display() {
        super.display();
        console.log("This is an overridden display method in the Car class.");
    }
}
```

The super() keyword calls the constructor of the parent class, allowing the child class to reuse and extend functionality.

---

4. Function Overriding

JavaScript allows **function overriding**, where a child class redefines a method with the same name as one in the parent class.

```
class CarShowRoom {
    display(brand, color, Reviews) {
        console.log("Car Brand: " + brand);
        console.log("Car Color: " + color);
        console.log("Car Reviews: " + Reviews);
    }
}
class Car extends CarShowRoom {
    display(brand, color) {
        console.log("Car Brand: " + brand);
        console.log("Car Color: " + color);
    }
}
```

In this example, the Car class overrides the display() method from the parent class and provides its own logic.

---

5. Incorrect Constructor Overloading (Common Mistake)

A common mistake is to **declare variables inside the constructor without using this**, which results in properties not being assigned to the object.

```
class CarShowRoom {
    constructor(brand, color, Reviews) {
        let br = brand;
        let co = color;
        let Re = Reviews;
        // These variables are local and do not attach to the object
    }

    display() {
        console.log("Car Brand: " + this.brand);   // undefined
        console.log("Car Color: " + this.color);   // undefined
        console.log("Car Reviews: " + this.Reviews); // undefined
    }
}
```

To fix this, always assign values using this.propertyName = value; inside constructors.
