

Session 8

Javascript Classes

JavaScript Classes, introduced in ES6, provide a clean, syntactic sugar over JavaScript's prototype-based inheritance, enabling true object-oriented programming (OOP) patterns. While JavaScript has always supported objects and prototypes, classes offer a familiar, template-driven approach to defining blueprints for objects with shared properties and methods. This session shifts focus from procedural scripting to OOP design, covering class declarations, constructors, instantiation, getters/setters, and inheritance.

Objectives

✕ Do not copy. Read for understanding and the viva

- Understand OOP architecture in JavaScript
- Define classes and instantiate objects with methods
- Implement getters and setters for controlled property access
- Apply inheritance using extends and super
- Solve hands-on exercises modeling real-world entities (library books, products)

What is a JavaScript Class?

- A class is a template/blueprint for creating objects.
- It encapsulates data (properties) and behavior (methods).
- Under the hood, classes still use JavaScript's prototype chain, but the syntax is more readable and maintainable.

Class Declarations vs. Expressions

Type	Syntax	Hoisting/Scope
Declaration	<code>class MyClass {}</code>	Not hoisted. Must be defined before use. Global/module scope.
Expression	<code>const MyClass = class {}</code>	Runs immediately. Scope limited to assignment. Can be named or unnamed.

Constructor & Instantiation

- The `constructor()` method runs automatically when a new object is created.
- Used to initialize instance properties.
- Objects are instantiated using the `new` keyword.

```
1 class Bike {  
2   constructor(make, year) {  
3     this.make = make;  
4     this.year = year;  
5   }  
6   getAge() {  
7     return new Date().getFullYear() - this.year;  
8   }  
9 }  
10 const myBike = new Bike("Honda", 2021);  
11 console.log(myBike.getAge()); // e.g., 5
```

JS

Getters & Setters

- Allow controlled access and validation for internal properties.
- Syntax uses `get` and `set` keywords.

Important: Getter/setter names must not match the internal property name to avoid infinite recursion.

```
1 class User {
2   constructor(name) { this._name = name; }
3   get name() { return this._name; }
4   set name(val) {
5     if (val.length > 0) this._name = val.toUpperCase();
6   }
7 }
```

JS

Inheritance (`extends` & `super`)

- Child classes inherit properties/methods from parent classes using `extends`.
- `super()` **must** be called in the child constructor before accessing `this`.
- Enables code reuse, method overriding, and polymorphism.

```
1 class Animal {
2   constructor(name) { this.name = name; }
3   speak() { return `${this.name} makes a sound.`; }
4 }
5 class Dog extends Animal {
6   constructor(name, breed) {
7     super(name); // Calls parent constructor
8     this.breed = breed;
9   }
10  speak() { return `${this.name} barks!`; } // Overrides parent method
11 }
```

JS

Best Practices & Common Pitfalls

✗ Do not copy. Read for understanding and the viva

Pitfall	Best Practice
Forgetting <code>super()</code> in derived constructor	Always call <code>super()</code> first in child class constructors
Naming getters/setters identical to properties	Use <code>_property</code> internally to prevent infinite recursion
Treating classes like functions	Classes are not hoisted; define before use
Overusing deep inheritance chains	Prefer composition when relationships aren't strict <code>is-a</code>

Quick Guide

✗ Do not copy. Read for understanding and the viva

JS

```
1 // ♦ Basic Class
2 class Product {
3   constructor(id, name) { this.id = id; this.name = name; }
4   getInfo() { return `${this.id}: ${this.name}`; }
5 }
6
7 // ♦ Getters & Setters
8 class User {
9   constructor(age) { this._age = age; }
10  get age() { return this._age; }
11  set age(val) { if(val ≥ 0) this._age = val; }
12 }
13
14 // ♦ Inheritance
15 class Student extends User {
16   constructor(age, grade) { super(age); this.grade = grade; }
17   status() { return `${this.age}yo, Grade ${this.grade}`; }
18 }
```

Question 1

Problem Statement

■ Write in lab record

Let us assume you need to organize a library of some electronic manuals and novels for a particular company. For each book, you need to store following information:

- The title
- The author
- The copyright date
- The ISBN
- The number of pages
- The number of times the book has been checked out.
- Whether the book has been discarded

Company wants to perform certain actions when the any book is outdated. The manual books must be exited after 5 years while a novel book should be exited if its checked out time crossed 100.

Task 1.Construct three classes that hold the information needed by company. One class should be a **Book** class and two child classes of the Book class called **Manual** and **Novel**. Each class will contain two methods. One will be a constructor. The other one will either be in charge of disposal of the book or updating the property related to the number of times a book has been checked out.

Task 2: Create an object of the Novel class and Manual class with some valid details.

Task 3: Write the method to count the duration of a manual book and checkout time for novel book so that they can be discarded based on the duration.

Requirements

■ Write in lab record

- Create `Book` base class + `Manual` & `Novel` child classes
- Store: title, author, copyright date, ISBN, pages, checkout count, discarded status

- Manual: discard if age > 5 years
- Novel: discard if checkouts > 100
- Each class: 1 constructor + 1 method (disposal or checkout update)
- Create objects & implement duration/checkout tracking

Demo

✕ Do not copy. Read for understanding and the viva

Library Book Management

Run Simulation

Code

■ Write in lab record

library.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Session 8 Ex1: Library System</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         padding: 20px;
10        max-width: 700px;
11        margin: auto;
12      }
13      .card {
14        background: #f9f9f9;
15        padding: 15px;
16        margin-bottom: 10px;
17        border-radius: 5px;
18        border-left: 4px solid #0056b3;
19      }
20    </style>
21  </head>
22  <body>
23    <h2>📖 Library Book Management</h2>
24    <div id="output"></div>
25
26    <script>
27      // Task 1: Base Class
28      class Book {
29        constructor(title, author, copyrightDate, ISBN, pages) {
30          this.title = title;
31          this.author = author;
32          this.copyrightDate = copyrightDate;
33          this.ISBN = ISBN;
34          this.pages = pages;
35          this.checkoutCount = 0;
36          this.discarded = false;
37        }
38      }
39
40      // Manual Child Class (Discard after 5 years)
41      class Manual extends Book {
42        constructor(title, author, copyrightDate, ISBN, pages) {
43          super(title, author, copyrightDate, ISBN, pages);
```

```
44         }
45         checkAndDiscardByAge() {
46             const currentYear = new Date().getFullYear();
47             const age = currentYear - this.copyrightDate;
48             if (age > 5) this.discarded = true;
49             return { age, discarded: this.discarded };
50         }
51     }
52
53     // Novel Child Class (Discard after 100 checkouts)
54     class Novel extends Book {
55         constructor(title, author, copyrightDate, ISBN, pages) {
56             super(title, author, copyrightDate, ISBN, pages);
57         }
58         updateCheckoutAndCheck() {
59             this.checkoutCount++;
60             if (this.checkoutCount > 100) this.discarded = true;
61             return {
62                 count: this.checkoutCount,
63                 discarded: this.discarded,
64             };
65         }
66     }
67
68     // Task 2 & 3: Instantiate & Test
69     const manual1 = new Manual(
70         "C++ Programming",
71         "Bjarne Stroustrup",
72         2018,
73         "978-0321563842",
74         1300,
75     );
76     const novel1 = new Novel(
77         "1984",
78         "George Orwell",
79         1949,
80         "978-0451524935",
81         328,
82     );
83
84     const mStatus = manual1.checkAndDiscardByAge();
85     for (let i = 0; i < 101; i++) novel1.updateCheckoutAndCheck();
86     const nStatus = {
87         count: novel1.checkoutCount,
```

```
88         discarded: novel1.discarded,
89     };
90
91     // Display Results
92     document.getElementById("output").innerHTML = `
93     <div class="card">
94         <strong>📖 Manual:</strong> "${manual1.title}"<br>
95         Age: ${mStatus.age} years | Checkouts: ${manual1.checkoutCount}<br>
96         Status: <span style="color:${mStatus.discarded ? "red" : "green"}">$
97     </div>
98     <div class="card">
99         <strong>📖 Novel:</strong> "${novel1.title}"<br>
100
101         Checkouts: ${nStatus.count} | Max Allowed: 100<br>
102
103         Status: <span style="color:${nStatus.discarded ? "red" : "green"}">$
104
105     </div>
106 `;
107
108     </script>
109
110 </body>
111
112 </html>
```

HTML View

Question 2

Problem Statement

■ Write in lab record

Create a `product` with properties: ID, Name, Description and Price. The create function for this object called `displayProductDetails()`. This function when invoked should display the name, description and discounted price of the product. For the calculation of discounted price you need to create another function `CalculateDisc(percentage)`. The discount percentage would be given by user.

Requirements

■ Write in lab record

- Create `product` object with: ID, Name, Description, Price
- Method `displayProductDetails()` → shows name, description, discounted price
- Method `CalculateDisc(percentage)` → computes discounted price
- Discount percentage provided by user

Demo

✕ Do not copy. Read for understanding and the viva

Product Discount Calculator

Discount %

e.g., 20

Calculate & Display Details

Code

■ Write in lab record

product.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Session 8 Ex2: Product Discount</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         padding: 20px;
10        max-width: 500px;
11        margin: auto;
12      }
13      input,
14      button {
15        padding: 10px;
16        margin: 5px;
17        width: 100%;
18        box-sizing: border-box;
19      }
20      pre {
21        background: #f4f4f4;
22        padding: 15px;
23        border-radius: 5px;
24        white-space: pre-wrap;
25      }
26    </style>
27  </head>
28  <body>
29    <h2>Product Discount Calculator</h2>
30    <input
31      id="discPercent"
32      type="number"
33      placeholder="Enter Discount % (e.g., 20)"
34    />
35    <button id="calcBtn">Calculate & Display Details</button>
36    <pre id="output">Waiting for input...</pre>
37
38    <script>
39      // Lab Requirement: Object with properties & methods
40      const product = {
41        id: "P1001",
42        name: "Wireless Mechanical Keyboard",
43        description: "RGB backlit, USB-C charging, Blue switches",
```

```
44         price: 149.99,
45
46         CalculateDisc: function (percentage) {
47             return this.price - this.price * (percentage / 100);
48         },
49
50         displayProductDetails: function (percentage) {
51             const discPrice = this.CalculateDisc(percentage);
52             return `📦 Name: ${this.name}\n📝 Description: ${this.de
53         },
54     };
55
56     document.getElementById("calcBtn").addEventListener("click", ()
57         const percent =
58             parseFloat(document.getElementById("discPercent").value)
59             0;
60     document.getElementById("output").textContent =
61         product.displayProductDetails(percent);
62     });
63 </script>
64 </body>
65 </html>
```

HTML View

[✎ Edit this page](#)

< Previous
Session 7

Next >
Session 9