Class 8: Mastering OOP Concepts in JavaScript

Topics Covered:

- Advanced OOP Concepts: call, bind, apply
- The this keyword
- Inheritance
- Error Handling in JavaScript

Objective: Deepen understanding of Object-Oriented Programming (OOP) in JavaScript, including advanced concepts such as function binding and context management, inheritance, and robust error handling. Apply these concepts in the context of e-commerce to ensure a resilient and scalable application.

Advanced OOP Concepts: call, bind, and apply and 'this' keyword

In JavaScript, call, bind, and apply are powerful methods for controlling the context (this) in which functions execute. Understanding these methods is crucial for advanced object-oriented programming (OOP) concepts in JavaScript.

1. call()

Definition: The call() method invokes a function with a specified this context and arguments provided individually.

Explanation:

- fetchData.call(this, url, callback) calls the fetchData function with the this context set to the current context (if any) and passes the URL and a callback function.
- Inside the callback function, populateTable.call(this, data) ensures populateTable is executed with the same this context.

2. bind()

Explanation:

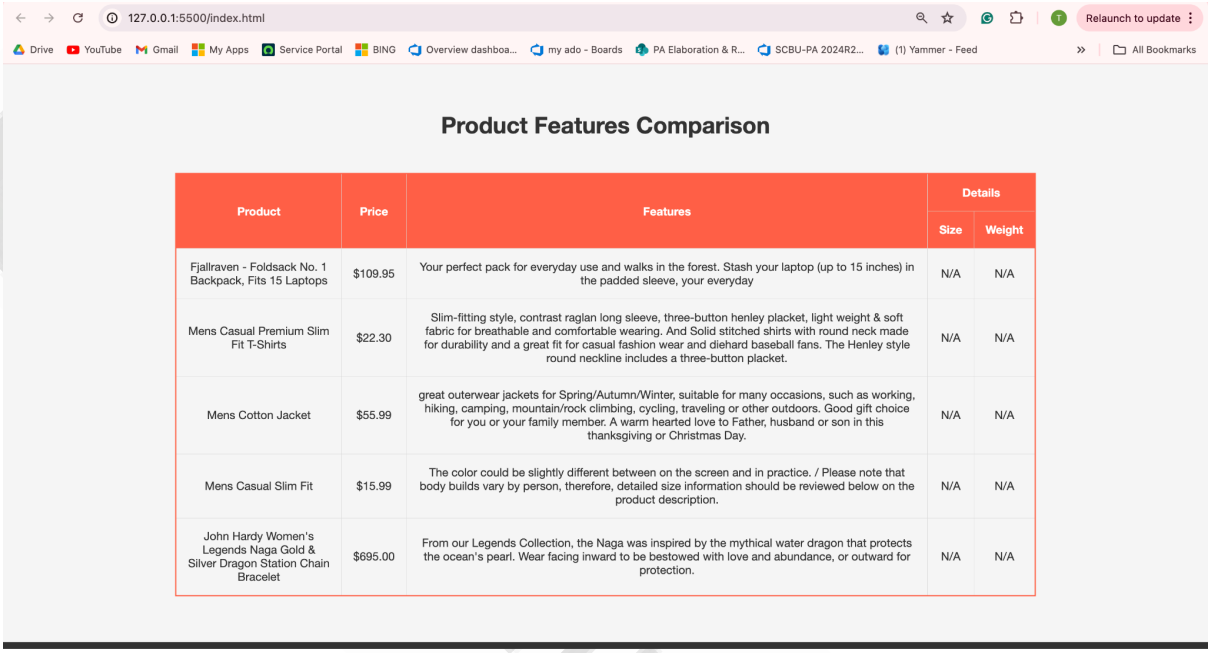
- fetchAndPopulateTable.bind(null, 'https://fakestoreapi.com/products') creates a new function fetchAndPopulate with the URL preset.
- This allows calling fetchAndPopulate without needing to specify the URL again, simplifying repeated function calls.

3. apply()

Explanation:

- fetchAndPopulate.apply(null, []) executes fetchAndPopulate with null as the this context and an empty array for arguments.
- This ensures the function executes without changing its arguments, useful for invoking functions in a specific context.

Output Screenshot:



Product Features Comparison

Product	Price	Features	Details	
			Size	Weight
Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops	$109.95	Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday	N/A	N/A
Mens Casual Premium Slim Fit T-Shirts	$22.30	Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.	N/A	N/A
Mens Cotton Jacket	$55.99	great outerwear jackets for Spring/Autumn/Winter, suitable for many occasions, such as working, hiking, camping, mountain/rock climbing, cycling, traveling or other outdoors. Good gift choice for you or your family member. A warm hearted love to Father, husband or son in this thanksgiving or Christmas Day.	N/A	N/A
Mens Casual Slim Fit	$15.99	The color could be slightly different between on the screen and in practice. / Please note that body builds vary by person, therefore, detailed size information should be reviewed below on the product description.	N/A	N/A
John Hardy Women's Legends Naga Gold & Silver Dragon Station Chain Bracelet	$695.00	From our Legends Collection, the Naga was inspired by the mythical water dragon that protects the ocean's pearl. Wear facing inward to be bestowed with love and abundance, or outward for protection.	N/A	N/A

Inheritance in JavaScript with the E-Commerce Product Page

Introduction to Inheritance

Inheritance is a fundamental concept in object-oriented programming (OOP) where one class (the child or derived class) inherits properties and methods from another class (the parent or base class). This allows for the reuse of code and the extension of functionalities.

In JavaScript, inheritance is achieved using the class syntax introduced in ES6. A derived class can use the extends keyword to inherit from a base class. The derived class can then override or extend the methods and properties of the base class.

Constructor: The Product class constructor initializes the name, price, description, and image properties of the product.

display Method: This method returns a string of HTML that represents how the product card will be rendered on the page.

- **Constructor**: The DiscountedProduct class constructor calls the parent Product class constructor using super() and initializes the additional discount property.

- **display Method**: This method overrides the display method of the Product class to include a discounted price and a discount badge. It calculates the discounted price and modifies the HTML output to reflect the discount.

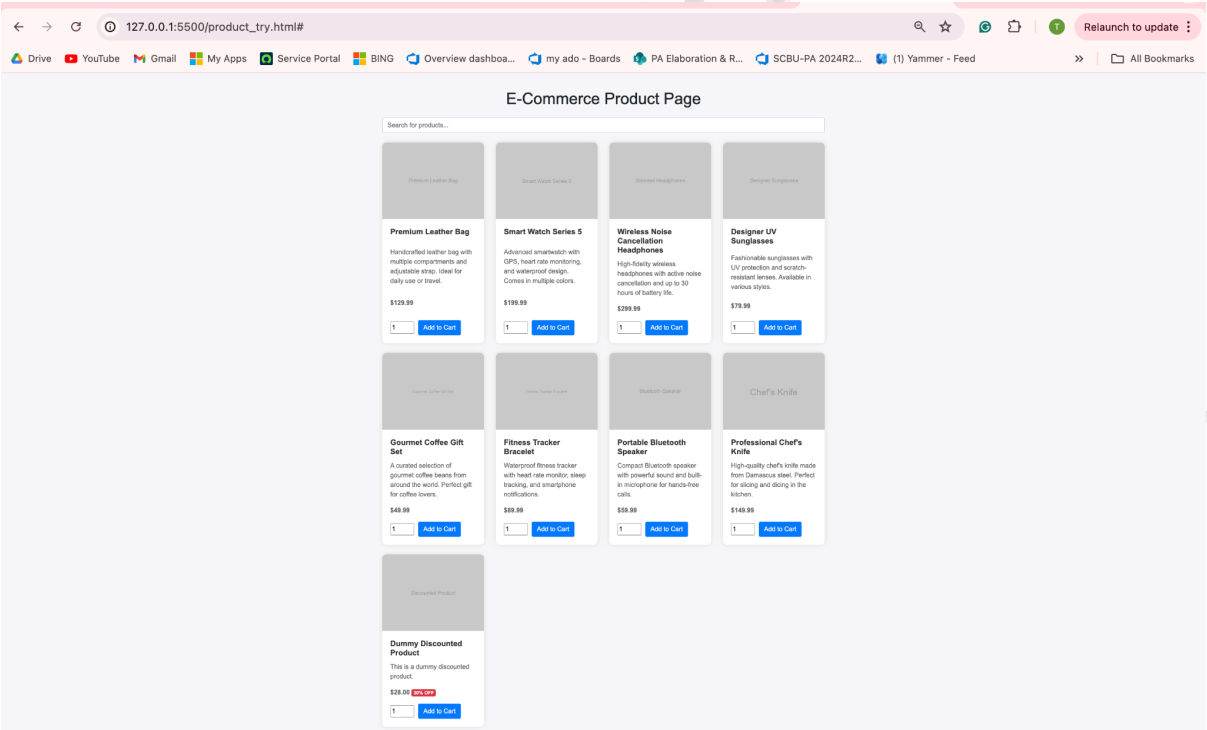
How Inheritance is Implemented in This Example

1. **Base Class (Product)**: The Product class serves as the blueprint for all standard products. It provides the basic structure and rendering for product cards.
2. **Derived Class (DiscountedProduct)**: The DiscountedProduct class extends the Product class to handle products with discounts. It reuses the product card layout from the Product class but adds additional functionality to display the discounted price and a discount badge.
3. **Use of super()**: The super() function in the DiscountedProduct class constructor is used to call the constructor of the Product class, ensuring that the base properties are properly initialized.
4. **Method Overriding**: The display method in DiscountedProduct overrides the display method of Product to provide a customized HTML output that includes discount details. This demonstrates how a derived class can modify or extend the behavior of a base class method.

Conclusion

Inheritance in JavaScript allows for the creation of more specialized classes (like DiscountedProduct) from a general base class (like Product). This approach promotes code reuse, simplifies maintenance, and enhances the organization of code by allowing derived classes to extend or override base class functionalities.

Output Screenshot:



Error Handling on the E-Commerce Product Page

Overview: Error handling is crucial for a smooth user experience, especially when dealing with dynamic data from external sources or user inputs. This page implements various error handling techniques to manage potential issues effectively.

1. Network Errors Handling

- **Issue**: Errors might occur during the fetching of product data from the server, such as network failures or server issues.

Explanation: The fetch function is used to retrieve product data. If the response status is not OK (response.ok is false), an error is thrown. This error is caught in the catch block, where it is logged to the console, and a message is shown to the user indicating that the products could not be loaded.

2. JSON Parsing Errors Handling

- **Issue**: Errors can occur if the JSON data fetched from the server is malformed or cannot be parsed.

Explanation: JSON parsing is done in a then block. If parsing fails, it will throw an error that is caught by the catch block, which then handles the error by displaying a message to the user.

3. Local Storage Errors Handling

- **Issue**: Errors might occur when reading from or writing to local storage, such as quota exceeded or data corruption.

Explanation: Local storage operations are wrapped in a try-catch block to handle any potential issues. If an error occurs during reading, writing, or parsing local storage, it is caught and logged, and the user is alerted.

4. UI/Interaction Errors Handling

- **Issue**: Errors might occur due to invalid user input or issues with interacting with UI elements.
- **Explanation**: Event listeners are set up to handle interactions with UI elements like buttons. Any errors during interaction (e.g., invalid quantity or issues parsing product data) are caught and handled by showing an appropriate alert to the user and logging the error.

5. Search Function Errors Handling

- **Issue**: Errors might occur during the search functionality, such as filtering issues or issues with the data format.

Explanation: The search functionality is enclosed in a `try-catch` block to handle any errors during the search operation. If an error occurs, it is logged and a user-friendly message is displayed.

Summary

- **Try-Catch Blocks**: Used extensively throughout the script to handle and log errors, while providing feedback to the user.
- **User Feedback**: Error messages are displayed on the page to inform users of issues, enhancing the user experience.
- **Logging**: Errors are logged to the console for debugging and maintenance purposes.

This approach ensures that errors are managed gracefully, providing a robust and user-friendly experience even when issues arise.

Interview and FAQ References

Understanding Advanced OOP Concepts

Object-Oriented Programming (OOP) in JavaScript involves using objects and classes to model real-world entities and their interactions. Advanced concepts include methods for controlling function context (call, bind, apply), the this keyword, inheritance, and error handling.

Interview Questions and Answers

1. What are call, bind, and apply in JavaScript? How do they differ?

Answer: These are methods used to control the context (this) of a function:

- **call()**: Calls a function with a specified this value and arguments provided individually.

```
function greet(greeting) {
    console.log(greeting + ', ' + this.name);
}
const person = { name: 'Alice' };
greet.call(person, 'Hello'); // Output: "Hello, Alice"
```

apply(): Similar to call(), but arguments are passed as an array.

```
function greet(greeting) {
    console.log(greeting + ', ' + this.name);
}
const person = { name: 'Alice' };
greet.apply(person, ['Hello']); // Output: "Hello, Alice"
```

bind(): Creates a new function with a specified this value and initial arguments, but does not invoke the function immediately.

```
function greet(greeting) {
    console.log(greeting + ', ' + this.name);
}
const person = { name: 'Alice' };
const greetAlice = greet.bind(person);
greetAlice('Hello'); // Output: "Hello, Alice"
```

2. How does the this keyword work in JavaScript?

Answer: The this keyword refers to the object from which the function was called:

- **Global Context**: Refers to the global object (window in browsers).
- **Object Method**: Refers to the object the method is called on.

```
const person = {
    name: 'Alice',
    greet() {
        console.log('Hello, ' + this.name);
    }
};
person.greet(); // Output: "Hello, Alice"
```

Constructor Function: Refers to the new instance being created.

```
function Person(name) {
    this.name = name;
}
const alice = new Person('Alice');
console.log(alice.name); // Output: "Alice"
```

Arrow Functions: Do not have their own this context and inherit this from the surrounding lexical context.

```
const person = {
    name: 'Alice',
    greet: () => {
        console.log('Hello, ' + this.name); // 'this' refers to the global
object
    }
};
person.greet(); // Output: "Hello, undefined"
```

3. How do you implement inheritance in JavaScript using classes?

Answer: Inheritance in JavaScript can be implemented using the extends keyword in classes:

```
class Animal {
    constructor(name) {
        this.name = name;
    }
    speak() {
        console.log(this.name + ' makes a noise.');
    }
}

class Dog extends Animal {
    speak() {
        console.log(this.name + ' barks.');
    }
}

const dog = new Dog('Rex');
dog.speak(); // Output: "Rex barks."
```

4. Explain the concept of method overriding and how it works in inheritance.

Answer: Method overriding occurs when a derived class redefines a method from its base class. The derived class's method will be used instead of the base class's method.

```
class Animal {
    speak() {
        console.log('Animal speaks');
    }
}

class Dog extends Animal {
    speak() {
        console.log('Dog barks');
    }
}

const dog = new Dog();
dog.speak(); // Output: "Dog barks"
```

Advanced Topics

5. What is the role of the super keyword in inheritance?

Answer: The super keyword is used to call methods on a parent class. It allows derived classes to access properties and methods from their base class.

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

```
    speak() {
        console.log(this.name + ' makes a noise.');
    }
}

class Dog extends Animal {
    constructor(name, breed) {
        super(name);
        this.breed = breed;
    }
    speak() {
        super.speak(); // Call the parent class's speak method
        console.log(this.name + ' barks.');
    }
}

const dog = new Dog('Rex', 'Labrador');
dog.speak();
// Output:
// "Rex makes a noise."
// "Rex barks."
```

6. How do you handle errors in JavaScript using try-catch?

Answer: Errors in JavaScript can be handled using try-catch blocks:

```
try {
    // Code that may throw an error
    let result = someFunction();
} catch (error) {
    // Code to handle the error
    console.error('Error occurred:', error.message);
} finally {
    // Optional code to execute regardless of whether an error occurred
    console.log('Execution completed');
}
```

7. Explain custom error classes in JavaScript.

Answer: Custom error classes extend the built-in Error class to create specific error types.

```
class CustomError extends Error {
    constructor(message) {
        super(message);
        this.name = 'CustomError';
    }
}

try {
```

```
    throw new CustomError('Something went wrong');
} catch (error) {
    console.error(error.name + ': ' + error.message); // Output: "CustomError:
Something went wrong"
}
```

8. How do you create and throw custom errors?

Answer: You can create custom errors by extending the Error class and then throw them as needed:

```
class ValidationError extends Error {
    constructor(message) {
        super(message);
        this.name = 'ValidationError';
    }
}

function validate(input) {
    if (input <= 0) {
        throw new ValidationError('Input must be greater than 0');
    }
}

try {
    validate(-1);
} catch (error) {
    if (error instanceof ValidationError) {
        console.error('Validation error:', error.message);
    } else {
        console.error('General error:', error.message);
    }
}
```