**Exercise 1: Creating a Class**

Create a Person class with the following properties:

* firstName
* lastName
* age

Add a method getFullName() that returns the full name of the person.

**Solution:**

class Person {

constructor(firstName, lastName, age) {

this.firstName = firstName;

this.lastName = lastName;

this.age = age;

}

getFullName() {

return `${this.firstName} ${this.lastName}`;

}

}

// Example usage

const person = new Person('John', 'Doe', 30);

console.log(person.getFullName()); // Output: John Doe

**Exercise 2: Inheritance**

Create a class Employee that inherits from the Person class. The Employee class should have an additional property called salary. Add a method getDetails() that returns the full name and salary of the employee.

**Solution:**

class Employee extends Person {

constructor(firstName, lastName, age, salary) {

super(firstName, lastName, age); // Calling the parent class constructor

this.salary = salary;

}

getDetails() {

return `${this.getFullName()} earns $${this.salary} per year.`;

}

}

// Example usage

const employee = new Employee('Jane', 'Smith', 28, 60000);

console.log(employee.getDetails()); // Output: Jane Smith earns $60000 per year.

**Exercise 3: Encapsulation**

Create a BankAccount class that has a private property \_balance. Add methods deposit(amount) to increase the balance and withdraw(amount) to decrease the balance, ensuring the balance never goes below 0. Provide a method getBalance() to view the current balance.

**Solution:**

class BankAccount {

#balance;

constructor(initialBalance = 0) {

this.#balance = initialBalance;

}

deposit(amount) {

this.#balance += amount;

}

withdraw(amount) {

if (amount <= this.#balance) {

this.#balance -= amount;

} else {

console.log('Insufficient balance!');

}

}

getBalance() {

return this.#balance;

}

}

// Example usage

const account = new BankAccount(100);

account.deposit(50);

console.log(account.getBalance()); // Output: 150

account.withdraw(30);

console.log(account.getBalance()); // Output: 120

account.withdraw(200); // Output: Insufficient balance!

**Exercise 4: Polymorphism**

Create a Shape class with a method getArea(). Create two subclasses: Rectangle and Circle. Override the getArea() method in both subclasses to calculate the area of a rectangle and a circle, respectively.

**Solution:**

class Shape {

getArea() {

return 0;

}

}

class Rectangle extends Shape {

constructor(width, height) {

super();

this.width = width;

this.height = height;

}

getArea() {

return this.width \* this.height;

}

}

class Circle extends Shape {

constructor(radius) {

super();

this.radius = radius;

}

getArea() {

return Math.PI \* this.radius \* this.radius;

}

}

// Example usage

const rect = new Rectangle(10, 5);

console.log(rect.getArea()); // Output: 50

const circle = new Circle(7);

console.log(circle.getArea()); // Output: approximately 153.94

**Exercise 5: Abstraction**

Create an abstract class Vehicle with a method move(). Create two subclasses: Car and Bicycle. Implement the move() method in each subclass to describe how each vehicle moves.

**Solution:**

class Vehicle {

move() {

throw new Error('Method "move()" must be implemented.');

}

}

class Car extends Vehicle {

move() {

return 'The car drives on the road.';

}

}

class Bicycle extends Vehicle {

move() {

return 'The bicycle pedals on the path.';

}

}

// Example usage

const car = new Car();

console.log(car.move()); // Output: The car drives on the road.

const bicycle = new Bicycle();

console.log(bicycle.move()); // Output: The bicycle pedals on the path.

**Exercise 6: Static Methods**

Create a MathHelper class with a static method square() that takes a number as an argument and returns its square.

**Solution:**

class MathHelper {

static square(number) {

return number \* number;

}

}

// Example usage

console.log(MathHelper.square(5)); // Output: 25

console.log(MathHelper.square(9)); // Output: 81

**Exercise 7: Getter and Setter**

Create a Product class with properties name and price. Use getter and setter methods to allow setting and getting the price but ensure the price cannot be set to a negative value.

**Solution:**

class Product {

constructor(name, price) {

this.name = name;

this.\_price = price;

}

get price() {

return this.\_price;

}

set price(value) {

if (value >= 0) {

this.\_price = value;

} else {

console.log('Price cannot be negative.');

}

}

}

// Example usage

const product = new Product('Laptop', 1000);

console.log(product.price); // Output: 1000

product.price = -200; // Output: Price cannot be negative.

product.price = 1200;

console.log(product.price); // Output: 1200

**Exercise 8: User Registration Form**

Create a form to register users with the following fields:

* Name
* Email
* Password

The form should validate that:

* The Name is not empty.
* The Email is in a valid format.
* The Password is at least 6 characters long.

Create a User class that stores the user's data. After submission, display the user's name and email in a separate section.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Registration Form</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="registrationForm">

<h2>Register</h2>

<label for="name">Name:</label>

<input type="text" id="name" name="name" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<button type="submit">Register</button>

</form>

<div id="userInfo"></div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 400px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #f7f7f7;

padding: 20px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

margin-bottom: 5px;

}

input {

margin-bottom: 15px;

padding: 8px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

#userInfo {

margin-top: 20px;

padding: 15px;

background-color: #e0f7fa;

border-radius: 5px;

}

**JavaScript (script.js):**

class User {

constructor(name, email, password) {

this.name = name;

this.email = email;

this.password = password;

}

getUserInfo() {

return `Name: ${this.name}<br>Email: ${this.email}`;

}

}

document.getElementById('registrationForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('name').value;

const email = document.getElementById('email').value;

const password = document.getElementById('password').value;

if (name && email && password.length >= 6) {

const user = new User(name, email, password);

document.getElementById('userInfo').innerHTML = user.getUserInfo();

} else {

alert('Please fill in all fields and ensure the password is at least 6 characters long.');

}

});

**Exercise 9: Product Inventory Form**

Create a form that adds products to an inventory with fields:

* Product Name
* Price
* Quantity

Create a Product class that stores product details and includes a method to calculate the total value (price \* quantity). After submission, display all products in a list, including their total value.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Product Inventory</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="productForm">

<h2>Add Product</h2>

<label for="productName">Product Name:</label>

<input type="text" id="productName" name="productName" required>

<label for="price">Price:</label>

<input type="number" id="price" name="price" min="0" required>

<label for="quantity">Quantity:</label>

<input type="number" id="quantity" name="quantity" min="1" required>

<button type="submit">Add Product</button>

</form>

<ul id="productList"></ul>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 400px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #f7f7f7;

padding: 20px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

margin-bottom: 5px;

}

input {

margin-bottom: 15px;

padding: 8px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

ul {

margin-top: 20px;

list-style-type: none;

padding: 0;

}

li {

background-color: #e0f7fa;

padding: 10px;

margin-bottom: 10px;

border-radius: 5px;

}

**JavaScript (script.js):**

class Product {

constructor(name, price, quantity) {

this.name = name;

this.price = parseFloat(price);

this.quantity = parseInt(quantity);

}

getTotalValue() {

return this.price \* this.quantity;

}

getProductDetails() {

return `Product: ${this.name}, Price: $${this.price}, Quantity: ${this.quantity}, Total Value: $${this.getTotalValue()}`;

}

}

document.getElementById('productForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('productName').value;

const price = document.getElementById('price').value;

const quantity = document.getElementById('quantity').value;

if (name && price > 0 && quantity > 0) {

const product = new Product(name, price, quantity);

const productDetails = document.createElement('li');

productDetails.innerHTML = product.getProductDetails();

document.getElementById('productList').appendChild(productDetails);

} else {

alert('Please provide valid product details.');

}

});

**Exercise 10: Contact Form with Validation**

Create a contact form with fields:

* Name
* Email
* Message

Create a Contact class that stores the form data. Validate the form, ensuring all fields are filled, the email is valid, and the message is at least 10 characters long. Display an error message if validation fails, otherwise, display the contact details.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Contact Form</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="contactForm">

<h2>Contact Us</h2>

<label for="contactName">Name:</label>

<input type="text" id="contactName" name="contactName" required>

<label for="contactEmail">Email:</label>

<input type="email" id="contactEmail" name="contactEmail" required>

<label for="contactMessage">Message:</label>

<textarea id="contactMessage" name="contactMessage" required></textarea>

<button type="submit">Send Message</button>

</form>

<div id="contactInfo"></div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

textarea {

height: 100px;

}

#contactInfo {

margin-top: 20px;

padding: 15px;

background-color: #e0f7fa;

border-radius: 5px;

}

**JavaScript (script.js):**

class Contact {

constructor(name, email, message) {

this.name = name;

this.email = email;

this.message = message;

}

getContactInfo() {

return `Name: ${this.name}<br>Email: ${this.email}<br>Message: ${this.message}`;

}

}

document.getElementById('contactForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('contactName').value;

const email = document.getElementById('contactEmail').value;

const message = document.getElementById('contactMessage').value;

const emailPattern = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/;

if (name && emailPattern.test(email) && message.length >= 10) {

const contact = new Contact(name, email, message);

document.getElementById('contactInfo').innerHTML = contact.getContactInfo();

} else {

alert('Please provide valid details. Message must be at least 10 characters long.');

}

});

**Explanation of the Code**

* **HTML:** The form fields include input elements for name and email and a textarea for the message. The form has a submit button that triggers the JavaScript code.
* **CSS:** The textarea is given a height of 100px for the message, and the contact information is displayed in a styled div once the form is submitted successfully.
* **JavaScript:**
  + A Contact class is used to store the form data (name, email, and message).
  + The form validation ensures that the name is not empty, the email matches a simple regular expression for validation, and the message is at least 10 characters long.
  + Upon successful validation, the contact details are displayed; otherwise, an error message is shown.

**Exercise 11: Vehicle Rental System**

Create a form for renting vehicles, including fields for:

* Vehicle Type (Car, Bike, Truck)
* Rental Days
* Price per Day

The form should calculate the total rental cost based on the selected vehicle type and the number of rental days. Create separate classes for each vehicle type (Car, Bike, Truck) that inherit from a Vehicle base class. After form submission, display the rental details and the total cost.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Vehicle Rental System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="rentalForm">

<h2>Rent a Vehicle</h2>

<label for="vehicleType">Vehicle Type:</label>

<select id="vehicleType" name="vehicleType" required>

<option value="Car">Car</option>

<option value="Bike">Bike</option>

<option value="Truck">Truck</option>

</select>

<label for="rentalDays">Rental Days:</label>

<input type="number" id="rentalDays" name="rentalDays" min="1" required>

<label for="pricePerDay">Price per Day:</label>

<input type="number" id="pricePerDay" name="pricePerDay" min="0" required>

<button type="submit">Calculate</button>

</form>

<div id="rentalInfo"></div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 400px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #f7f7f7;

padding: 20px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

margin-bottom: 5px;

}

input, select {

margin-bottom: 15px;

padding: 8px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

#rentalInfo {

margin-top: 20px;

padding: 15px;

background-color: #e0f7fa;

border-radius: 5px;

}

**JavaScript (script.js):**

class Vehicle {

constructor(rentalDays, pricePerDay) {

this.rentalDays = parseInt(rentalDays);

this.pricePerDay = parseFloat(pricePerDay);

}

getTotalCost() {

return this.rentalDays \* this.pricePerDay;

}

}

class Car extends Vehicle {

constructor(rentalDays, pricePerDay) {

super(rentalDays, pricePerDay);

}

getDetails() {

return `Car rented for ${this.rentalDays} days at $${this.pricePerDay} per day.`;

}

}

class Bike extends Vehicle {

constructor(rentalDays, pricePerDay) {

super(rentalDays, pricePerDay);

}

getDetails() {

return `Bike rented for ${this.rentalDays} days at $${this.pricePerDay} per day.`;

}

}

class Truck extends Vehicle {

constructor(rentalDays, pricePerDay) {

super(rentalDays, pricePerDay);

}

getDetails() {

return `Truck rented for ${this.rentalDays} days at $${this.pricePerDay} per day.`;

}

}

document.getElementById('rentalForm').addEventListener('submit', function (e) {

e.preventDefault();

const vehicleType = document.getElementById('vehicleType').value;

const rentalDays = document.getElementById('rentalDays').value;

const pricePerDay = document.getElementById('pricePerDay').value;

let vehicle;

switch (vehicleType) {

case 'Car':

vehicle = new Car(rentalDays, pricePerDay);

break;

case 'Bike':

vehicle = new Bike(rentalDays, pricePerDay);

break;

case 'Truck':

vehicle = new Truck(rentalDays, pricePerDay);

break;

default:

alert('Please select a valid vehicle type');

return;

}

const totalCost = vehicle.getTotalCost();

const details = vehicle.getDetails();

document.getElementById('rentalInfo').innerHTML = `${details}<br>Total Cost: $${totalCost}`;

});

**Exercise 12: Library Management System**

Create a form for adding books to a library with fields:

* Title
* Author
* Genre
* Pages

Use OOP principles to create a Library class that stores multiple books. A Book class should handle individual books. The form should allow users to add multiple books, and after submission, display a list of all added books.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Library Management</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="bookForm">

<h2>Add a Book</h2>

<label for="title">Title:</label>

<input type="text" id="title" name="title" required>

<label for="author">Author:</label>

<input type="text" id="author" name="author" required>

<label for="genre">Genre:</label>

<input type="text" id="genre" name="genre" required>

<label for="pages">Pages:</label>

<input type="number" id="pages" name="pages" required>

<button type="submit">Add Book</button>

</form>

<ul id="bookList"></ul>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 400px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #f7f7f7;

padding: 20px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

margin-bottom: 5px;

}

input {

margin-bottom: 15px;

padding: 8px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

ul {

margin-top: 20px;

list-style-type: none;

padding: 0;

}

li {

background-color: #e0f7fa;

padding: 10px;

margin-bottom: 10px;

border-radius: 5px;

}

**JavaScript (script.js):**

class Book {

constructor(title, author, genre, pages) {

this.title = title;

this.author = author;

this.genre = genre;

this.pages = parseInt(pages);

}

getBookDetails() {

return `${this.title} by ${this.author} (${this.genre}) - ${this.pages} pages`;

}

}

class Library {

constructor() {

this.books = [];

}

addBook(book) {

this.books.push(book);

}

getBookList() {

return this.books.map(book => book.getBookDetails());

}

}

const library = new Library();

document.getElementById('bookForm').addEventListener('submit', function (e) {

e.preventDefault();

const title = document.getElementById('title').value;

const author = document.getElementById('author').value;

const genre = document.getElementById('genre').value;

const pages = document.getElementById('pages').value;

if (title && author && genre && pages > 0) {

const book = new Book(title, author, genre, pages);

library.addBook(book);

const bookList = document.getElementById('bookList');

bookList.innerHTML = '';

library.getBookList().forEach(bookDetails => {

const li = document.createElement('li');

li.textContent = bookDetails;

bookList.appendChild(li);

});

} else {

alert('Please fill in all the fields with valid information.');

}

});

**Explanation:**

* **Vehicle Rental System** involves inheritance and allows calculation of total rental costs for different vehicle types.
* **Library Management System** allows users to add multiple books using OOP principles, creating Book objects stored in a Library.

**Exercise 13: Hotel Reservation System**

Create a hotel reservation form that includes fields for:

* Name
* Email
* Room Type (Single, Double, Suite)
* Check-in Date
* Check-out Date
* Number of Guests

After form submission, calculate the total cost based on room type and number of days, and ensure the following validations:

* The check-out date must be after the check-in date.
* The number of guests should be appropriate for the selected room type:
  + Single: 1 guest max
  + Double: 2 guests max
  + Suite: 4 guests max

Use JavaScript OOP principles to create a Reservation class to handle the logic, and display the reservation details and total cost upon submission.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Hotel Reservation System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="reservationForm">

<h2>Hotel Reservation</h2>

<label for="name">Name:</label>

<input type="text" id="name" name="name" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<label for="roomType">Room Type:</label>

<select id="roomType" name="roomType" required>

<option value="Single">Single</option>

<option value="Double">Double</option>

<option value="Suite">Suite</option>

</select>

<label for="checkInDate">Check-in Date:</label>

<input type="date" id="checkInDate" name="checkInDate" required>

<label for="checkOutDate">Check-out Date:</label>

<input type="date" id="checkOutDate" name="checkOutDate" required>

<label for="guests">Number of Guests:</label>

<input type="number" id="guests" name="guests" min="1" max="4" required>

<button type="submit">Book Now</button>

</form>

<div id="reservationInfo"></div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 500px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #f7f7f7;

padding: 20px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

margin-bottom: 5px;

}

input, select {

margin-bottom: 15px;

padding: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 12px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

#reservationInfo {

margin-top: 20px;

padding: 15px;

background-color: #e0f7fa;

border-radius: 5px;

}

**JavaScript (script.js):**

class Reservation {

constructor(name, email, roomType, checkInDate, checkOutDate, guests) {

this.name = name;

this.email = email;

this.roomType = roomType;

this.checkInDate = new Date(checkInDate);

this.checkOutDate = new Date(checkOutDate);

this.guests = parseInt(guests);

this.pricePerNight = this.getPricePerNight();

}

getPricePerNight() {

switch (this.roomType) {

case 'Single':

return 100;

case 'Double':

return 150;

case 'Suite':

return 300;

default:

return 0;

}

}

getNumberOfNights() {

const oneDay = 24 \* 60 \* 60 \* 1000;

return Math.round((this.checkOutDate - this.checkInDate) / oneDay);

}

validateDates() {

return this.checkOutDate > this.checkInDate;

}

validateGuests() {

if (this.roomType === 'Single' && this.guests > 1) return false;

if (this.roomType === 'Double' && this.guests > 2) return false;

if (this.roomType === 'Suite' && this.guests > 4) return false;

return true;

}

getTotalCost() {

return this.getNumberOfNights() \* this.pricePerNight;

}

getReservationDetails() {

return `

Name: ${this.name}<br>

Email: ${this.email}<br>

Room Type: ${this.roomType}<br>

Check-in: ${this.checkInDate.toDateString()}<br>

Check-out: ${this.checkOutDate.toDateString()}<br>

Guests: ${this.guests}<br>

Total Cost: $${this.getTotalCost()}

`;

}

}

document.getElementById('reservationForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('name').value;

const email = document.getElementById('email').value;

const roomType = document.getElementById('roomType').value;

const checkInDate = document.getElementById('checkInDate').value;

const checkOutDate = document.getElementById('checkOutDate').value;

const guests = document.getElementById('guests').value;

const reservation = new Reservation(name, email, roomType, checkInDate, checkOutDate, guests);

if (!reservation.validateDates()) {

alert('Check-out date must be after check-in date.');

return;

}

if (!reservation.validateGuests()) {

alert('Invalid number of guests for the selected room type.');

return;

}

document.getElementById('reservationInfo').innerHTML = reservation.getReservationDetails();

});

**Explanation:**

* **HTML:** The form includes fields for name, email, room type selection, dates, and the number of guests. The form submission triggers the JavaScript validation and cost calculation.
* **CSS:** Basic styling for form inputs, buttons, and result display using flexbox and consistent colors.
* **JavaScript:**
  + The Reservation class encapsulates the logic for handling the reservation process. It includes methods for validating dates and guest numbers based on room type, calculating the total cost, and displaying reservation details.
  + Validation ensures that the check-out date is after the check-in date and that the number of guests is appropriate for the selected room type.
  + The cost is calculated based on the number of nights and the room type’s price per night.

**Exercise 14: Airline Booking System**

Create a form for booking airline tickets with the following fields:

* Full Name
* Email
* Destination (A list of options like New York, London, Tokyo)
* Departure Date
* Return Date
* Number of Passengers
* Ticket Class (Economy, Business, First Class)

Calculate the total cost based on the number of passengers, ticket class, and destination. The form should validate that:

* The return date is after the departure date.
* The number of passengers is between 1 and 6.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Airline Booking System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="bookingForm">

<h2>Book Your Flight</h2>

<label for="fullName">Full Name:</label>

<input type="text" id="fullName" name="fullName" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<label for="destination">Destination:</label>

<select id="destination" name="destination" required>

<option value="New York">New York</option>

<option value="London">London</option>

<option value="Tokyo">Tokyo</option>

</select>

<label for="departureDate">Departure Date:</label>

<input type="date" id="departureDate" name="departureDate" required>

<label for="returnDate">Return Date:</label>

<input type="date" id="returnDate" name="returnDate" required>

<label for="passengers">Number of Passengers:</label>

<input type="number" id="passengers" name="passengers" min="1" max="6" required>

<label for="ticketClass">Ticket Class:</label>

<select id="ticketClass" name="ticketClass" required>

<option value="Economy">Economy</option>

<option value="Business">Business</option>

<option value="First Class">First Class</option>

</select>

<button type="submit">Book Now</button>

</form>

<div id="bookingInfo"></div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 600px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #f0f8ff;

padding: 20px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

margin-bottom: 5px;

}

input, select {

margin-bottom: 15px;

padding: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 12px;

background-color: #3498db;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #2980b9;

}

#bookingInfo {

margin-top: 20px;

padding: 15px;

background-color: #e0f7fa;

border-radius: 5px;

}

**JavaScript (script.js):**

class FlightBooking {

constructor(fullName, email, destination, departureDate, returnDate, passengers, ticketClass) {

this.fullName = fullName;

this.email = email;

this.destination = destination;

this.departureDate = new Date(departureDate);

this.returnDate = new Date(returnDate);

this.passengers = parseInt(passengers);

this.ticketClass = ticketClass;

this.pricePerPassenger = this.getPricePerPassenger();

}

getPricePerPassenger() {

let basePrice;

switch (this.destination) {

case 'New York':

basePrice = 500;

break;

case 'London':

basePrice = 700;

break;

case 'Tokyo':

basePrice = 900;

break;

default:

basePrice = 0;

}

switch (this.ticketClass) {

case 'Economy':

return basePrice;

case 'Business':

return basePrice \* 1.5;

case 'First Class':

return basePrice \* 2.5;

default:

return 0;

}

}

getNumberOfDays() {

const oneDay = 24 \* 60 \* 60 \* 1000;

return Math.round((this.returnDate - this.departureDate) / oneDay);

}

validateDates() {

return this.returnDate > this.departureDate;

}

validatePassengers() {

return this.passengers >= 1 && this.passengers <= 6;

}

getTotalCost() {

return this.passengers \* this.pricePerPassenger;

}

getBookingDetails() {

return `

Full Name: ${this.fullName}<br>

Email: ${this.email}<br>

Destination: ${this.destination}<br>

Departure Date: ${this.departureDate.toDateString()}<br>

Return Date: ${this.returnDate.toDateString()}<br>

Passengers: ${this.passengers}<br>

Ticket Class: ${this.ticketClass}<br>

Total Cost: $${this.getTotalCost()}

`;

}

}

document.getElementById('bookingForm').addEventListener('submit', function (e) {

e.preventDefault();

const fullName = document.getElementById('fullName').value;

const email = document.getElementById('email').value;

const destination = document.getElementById('destination').value;

const departureDate = document.getElementById('departureDate').value;

const returnDate = document.getElementById('returnDate').value;

const passengers = document.getElementById('passengers').value;

const ticketClass = document.getElementById('ticketClass').value;

const booking = new FlightBooking(fullName, email, destination, departureDate, returnDate, passengers, ticketClass);

if (!booking.validateDates()) {

alert('Return date must be after departure date.');

return;

}

if (!booking.validatePassengers()) {

alert('Number of passengers must be between 1 and 6.');

return;

}

document.getElementById('bookingInfo').innerHTML = booking.getBookingDetails();

});

**Explanation:**

* **HTML:** The form collects user input for booking a flight, including full name, destination, dates, and ticket class.
* **CSS:** The form styling uses flexbox for alignment and provides a clean, readable layout.
* **JavaScript:**
  + The FlightBooking class manages the logic for booking flights. It includes methods for validating the dates and number of passengers, and calculates the total cost based on the destination and ticket class.
  + Validations ensure that the return date is after the departure date and the number of passengers is between 1 and 6.
  + The total cost is calculated and displayed based on the number of passengers and the selected ticket class.

**Exercise 15: Car Rental Management System**

Build a car rental management system where users can:

* Choose a car category (Economy, SUV, Luxury)
* Select the rental duration in days
* Choose optional add-ons such as a GPS or child seat
* Input their name, email, and driver's license number

The system should:

* Calculate the total rental cost based on the car category and duration
* Add the cost of any optional services
* Validate that all fields are filled, the duration is at least one day, and the driver's license is valid (using a regular expression)

After form submission, display the rental summary including the total cost and details about the selected car and services.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Car Rental Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="rentalForm">

<h2>Rent a Car</h2>

<label for="name">Full Name:</label>

<input type="text" id="name" name="name" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<label for="license">Driver's License:</label>

<input type="text" id="license" name="license" required>

<label for="category">Car Category:</label>

<select id="category" name="category" required>

<option value="Economy">Economy</option>

<option value="SUV">SUV</option>

<option value="Luxury">Luxury</option>

</select>

<label for="days">Rental Duration (days):</label>

<input type="number" id="days" name="days" min="1" required>

<label for="gps">Add GPS ($10/day):</label>

<input type="checkbox" id="gps" name="gps">

<label for="childSeat">Add Child Seat ($5/day):</label>

<input type="checkbox" id="childSeat" name="childSeat">

<button type="submit">Calculate Total</button>

</form>

<div id="rentalInfo"></div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 600px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #f9f9f9;

padding: 20px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

margin-bottom: 5px;

}

input, select {

margin-bottom: 15px;

padding: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 12px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

#rentalInfo {

margin-top: 20px;

padding: 15px;

background-color: #e0f7fa;

border-radius: 5px;

}

**JavaScript (script.js):**

class Rental {

constructor(name, email, license, category, days, gps, childSeat) {

this.name = name;

this.email = email;

this.license = license;

this.category = category;

this.days = parseInt(days);

this.gps = gps;

this.childSeat = childSeat;

this.pricePerDay = this.getPricePerDay();

}

getPricePerDay() {

switch (this.category) {

case 'Economy':

return 50;

case 'SUV':

return 80;

case 'Luxury':

return 120;

default:

return 0;

}

}

validateLicense() {

const licensePattern = /^[A-Z]{1}[0-9]{7}$/; // Example license pattern (e.g., A1234567)

return licensePattern.test(this.license);

}

getAdditionalServicesCost() {

let additionalCost = 0;

if (this.gps) {

additionalCost += 10 \* this.days;

}

if (this.childSeat) {

additionalCost += 5 \* this.days;

}

return additionalCost;

}

getTotalCost() {

const baseCost = this.days \* this.pricePerDay;

const additionalCost = this.getAdditionalServicesCost();

return baseCost + additionalCost;

}

getRentalDetails() {

return `

Full Name: ${this.name}<br>

Email: ${this.email}<br>

Driver's License: ${this.license}<br>

Car Category: ${this.category}<br>

Rental Duration: ${this.days} days<br>

GPS: ${this.gps ? 'Yes' : 'No'}<br>

Child Seat: ${this.childSeat ? 'Yes' : 'No'}<br>

Total Cost: $${this.getTotalCost()}

`;

}

}

document.getElementById('rentalForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('name').value;

const email = document.getElementById('email').value;

const license = document.getElementById('license').value;

const category = document.getElementById('category').value;

const days = document.getElementById('days').value;

const gps = document.getElementById('gps').checked;

const childSeat = document.getElementById('childSeat').checked;

const rental = new Rental(name, email, license, category, days, gps, childSeat);

if (!rental.validateLicense()) {

alert('Invalid driver\'s license format.');

return;

}

document.getElementById('rentalInfo').innerHTML = rental.getRentalDetails();

});

**Explanation:**

* **HTML:**
  + The form includes fields for user input like name, email, driver's license, car category, rental duration, and additional services such as GPS or child seat.
  + The form submission triggers the calculation and validation logic in the JavaScript file.
* **CSS:**
  + Provides a clean and simple layout for the form using flexbox, and the result display is highlighted for user clarity.
* **JavaScript:**
  + The Rental class handles the rental logic, calculating costs based on the selected car category and rental duration. It also checks if optional services (GPS, child seat) are selected.
  + Validation ensures that the driver's license follows a specific pattern, and additional services are appropriately charged based on the number of rental days.
  + After form submission, the total cost and rental details are displayed.

**Exercise 16: Event Management System**

Create an event management system where users can:

* Enter details for organizing an event (Event Name, Organizer Name, Date, Number of Guests)
* Choose from a list of event services (Catering, Photography, Music, Decorations)
* Enter their budget, and the system will:
  + Calculate the total cost based on selected services
  + Validate the guest number (minimum of 10) and that the budget covers the event cost
  + Display a warning if the budget is insufficient or display the final event summary if successful

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Event Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<form id="eventForm">

<h2>Organize Your Event</h2>

<label for="eventName">Event Name:</label>

<input type="text" id="eventName" name="eventName" required>

<label for="organizerName">Organizer Name:</label>

<input type="text" id="organizerName" name="organizerName" required>

<label for="eventDate">Event Date:</label>

<input type="date" id="eventDate" name="eventDate" required>

<label for="guests">Number of Guests:</label>

<input type="number" id="guests" name="guests" min="10" required>

<label for="budget">Budget ($):</label>

<input type="number" id="budget" name="budget" required>

<h3>Event Services</h3>

<label for="catering">Catering ($30/guest):</label>

<input type="checkbox" id="catering" name="catering">

<label for="photography">Photography ($500):</label>

<input type="checkbox" id="photography" name="photography">

<label for="music">Music ($300):</label>

<input type="checkbox" id="music" name="music">

<label for="decorations">Decorations ($200):</label>

<input type="checkbox" id="decorations" name="decorations">

<button type="submit">Calculate Total</button>

</form>

<div id="eventInfo"></div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 600px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #fff8dc;

padding: 20px;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

margin-bottom: 5px;

}

input {

margin-bottom: 15px;

padding: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 12px;

background-color: #FF4500;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #DC143C;

}

#eventInfo {

margin-top: 20px;

padding: 15px;

background-color: #f0fff0;

border-radius: 5px;

}

**JavaScript (script.js):**

class Event {

constructor(eventName, organizerName, eventDate, guests, budget, catering, photography, music, decorations) {

this.eventName = eventName;

this.organizerName = organizerName;

this.eventDate = new Date(eventDate);

this.guests = parseInt(guests);

this.budget = parseFloat(budget);

this.catering = catering;

this.photography = photography;

this.music = music;

this.decorations = decorations;

this.totalCost = this.calculateTotalCost();

}

calculateTotalCost() {

let total = 0;

if (this.catering) {

total += 30 \* this.guests;

}

if (this.photography) {

total += 500;

}

if (this.music) {

total += 300;

}

if (this.decorations) {

total += 200;

}

return total;

}

validateGuests() {

return this.guests >= 10;

}

isBudgetSufficient() {

return this.budget >= this.totalCost;

}

getEventDetails() {

return `

Event Name: ${this.eventName}<br>

Organizer: ${this.organizerName}<br>

Event Date: ${this.eventDate.toDateString()}<br>

Number of Guests: ${this.guests}<br>

Total Cost: $${this.totalCost}<br>

Budget: $${this.budget}<br>

${this.isBudgetSufficient() ? 'Budget is sufficient!' : 'Warning: Budget is insufficient!'}

`;

}

}

document.getElementById('eventForm').addEventListener('submit', function (e) {

e.preventDefault();

const eventName = document.getElementById('eventName').value;

const organizerName = document.getElementById('organizerName').value;

const eventDate = document.getElementById('eventDate').value;

const guests = document.getElementById('guests').value;

const budget = document.getElementById('budget').value;

const catering = document.getElementById('catering').checked;

const photography = document.getElementById('photography').checked;

const music = document.getElementById('music').checked;

const decorations = document.getElementById('decorations').checked;

const event = new Event(eventName, organizerName, eventDate, guests, budget, catering, photography, music, decorations);

if (!event.validateGuests()) {

alert('Minimum number of guests is 10.');

return;

}

document.getElementById('eventInfo').innerHTML = event.getEventDetails();

});

**Explanation:**

* **HTML:** The form collects event details, including the number of guests, budget, and optional services (Catering, Photography, Music, Decorations).
* **CSS:** Provides a visually appealing design with simple color contrasts for input fields and result display.
* **JavaScript:**
  + The Event class calculates the total cost based on the selected services and validates that the budget is sufficient and the guest count meets the minimum requirement.
  + The system displays either a success message or a warning about insufficient budget based on the total cost and user-provided budget.

**Exercise 17: Online Library Management System**

Create an online library system where users can:

* Register as a user by entering their details (name, email, and membership type).
* Browse available books with categories such as Fiction, Non-Fiction, Science, History, etc.
* Borrow or return books, with a restriction based on their membership type (e.g., a regular member can borrow up to 3 books, while a premium member can borrow up to 5).
* Track their borrowing history and the total number of books they have borrowed.

The system should consist of multiple classes to handle users, books, and the library system itself. Each class should interact to manage the borrowing and returning of books.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Library Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2>Library Management System</h2>

<form id="registerForm">

<h3>User Registration</h3>

<label for="userName">Name:</label>

<input type="text" id="userName" required>

<label for="userEmail">Email:</label>

<input type="email" id="userEmail" required>

<label for="membership">Membership Type:</label>

<select id="membership" required>

<option value="Regular">Regular (Max 3 books)</option>

<option value="Premium">Premium (Max 5 books)</option>

</select>

<button type="submit">Register</button>

</form>

<div id="librarySection">

<h3>Browse Books</h3>

<div id="bookList"></div>

<h3>Your Borrowed Books</h3>

<div id="borrowedBooks"></div>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 800px;

margin: 0 auto;

}

form {

display: flex;

flex-direction: column;

background-color: #f4f4f4;

padding: 20px;

margin-bottom: 20px;

border-radius: 5px;

}

label {

margin-bottom: 5px;

}

input, select {

margin-bottom: 15px;

padding: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

padding: 12px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

#librarySection {

margin-top: 20px;

}

#bookList, #borrowedBooks {

background-color: #e8f0fe;

padding: 15px;

border-radius: 5px;

}

.book-item, .borrow-item {

margin-bottom: 10px;

}

**JavaScript (script.js):**

class User {

constructor(name, email, membershipType) {

this.name = name;

this.email = email;

this.membershipType = membershipType;

this.borrowedBooks = [];

}

canBorrow() {

const limit = this.membershipType === 'Premium' ? 5 : 3;

return this.borrowedBooks.length < limit;

}

borrowBook(book) {

if (this.canBorrow()) {

this.borrowedBooks.push(book);

} else {

console.log(`${this.name} cannot borrow more books.`);

}

}

returnBook(book) {

this.borrowedBooks = this.borrowedBooks.filter(b => b !== book);

}

}

class Book {

constructor(title, category) {

this.title = title;

this.category = category;

this.isAvailable = true;

}

borrow() {

if (this.isAvailable) {

this.isAvailable = false;

} else {

console.log(`${this.title} is already borrowed.`);

}

}

returnBook() {

this.isAvailable = true;

}

}

class Library {

constructor() {

this.users = [];

this.books = [];

}

addUser(user) {

this.users.push(user);

}

addBook(book) {

this.books.push(book);

}

displayAvailableBooks() {

return this.books.filter(book => book.isAvailable);

}

}

// Initialize Library and some sample books

const library = new Library();

library.addBook(new Book("The Great Gatsby", "Fiction"));

library.addBook(new Book("A Brief History of Time", "Science"));

library.addBook(new Book("Sapiens", "History"));

library.addBook(new Book("1984", "Fiction"));

library.addBook(new Book("The Selfish Gene", "Science"));

// Function to update the book list UI

function updateBookList() {

const bookList = document.getElementById('bookList');

const availableBooks = library.displayAvailableBooks();

bookList.innerHTML = '';

availableBooks.forEach((book, index) => {

const bookItem = document.createElement('div');

bookItem.classList.add('book-item');

bookItem.innerHTML = `${book.title} - ${book.category} <button onclick="borrowBook(${index})">Borrow</button>`;

bookList.appendChild(bookItem);

});

}

// Function to update the borrowed books UI

function updateBorrowedBooks(user) {

const borrowedBooks = document.getElementById('borrowedBooks');

borrowedBooks.innerHTML = '';

user.borrowedBooks.forEach((book, index) => {

const borrowItem = document.createElement('div');

borrowItem.classList.add('borrow-item');

borrowItem.innerHTML = `${book.title} - ${book.category} <button onclick="returnBook(${index})">Return</button>`;

borrowedBooks.appendChild(borrowItem);

});

}

let currentUser;

document.getElementById('registerForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('userName').value;

const email = document.getElementById('userEmail').value;

const membershipType = document.getElementById('membership').value;

currentUser = new User(name, email, membershipType);

library.addUser(currentUser);

alert(`Welcome ${name}! You are registered as a ${membershipType} member.`);

updateBookList();

});

// Function to handle borrowing books

function borrowBook(index) {

const book = library.books[index];

if (book.isAvailable && currentUser.canBorrow()) {

book.borrow();

currentUser.borrowBook(book);

updateBookList();

updateBorrowedBooks(currentUser);

} else {

alert("You cannot borrow more books or this book is unavailable.");

}

}

// Function to handle returning books

function returnBook(index) {

const book = currentUser.borrowedBooks[index];

book.returnBook();

currentUser.returnBook(book);

updateBookList();

updateBorrowedBooks(currentUser);

}

**Explanation:**

* **Classes**:
  1. User: Represents a library user. The user can borrow and return books, with borrowing restrictions based on their membership type (either Regular or Premium).
  2. Book: Represents a book in the library. Each book has a title, category, and availability status.
  3. Library: The central class managing the users and books. It provides methods to add users and books and display the available books.
* **Interaction**:
  1. After registering, the user can borrow books. Their borrowing limit is determined by their membership type. Once the borrowing limit is reached, they are restricted from borrowing more books.
  2. Users can return books, which will make them available for other users to borrow.
* **UI Updates**:
  1. The available books are displayed dynamically on the page.
  2. Borrowed books are also displayed in a separate section with an option to return them.

**Exercise 18: E-Commerce Shopping Cart System**

In this exercise, you will create a multi-class shopping cart system for an e-commerce website. Users can browse products, add them to their cart, update quantities, remove products from the cart, and proceed to checkout. The system should track product inventory, calculate totals (including tax), and offer discounts.

**Features:**

* **Product Catalog**: Displays available products with their details and prices.
* **Shopping Cart**: Users can add/remove products, update quantities, and see real-time cart updates.
* **Inventory Management**: Tracks product availability and prevents users from purchasing more than is available.
* **Checkout**: Calculates the total cost, including discounts and taxes.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Shopping Cart System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2>E-Commerce Shopping Cart System</h2>

<div id="productList">

<h3>Product Catalog</h3>

<div id="products"></div>

</div>

<div id="cartSection">

<h3>Your Shopping Cart</h3>

<div id="cart"></div>

<div id="cartSummary"></div>

<button id="checkoutButton">Proceed to Checkout</button>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 900px;

margin: 0 auto;

}

#productList, #cartSection {

margin-bottom: 30px;

}

#products, #cart {

background-color: #f4f4f4;

padding: 20px;

border-radius: 5px;

}

.product-item, .cart-item {

margin-bottom: 10px;

}

button {

padding: 10px;

background-color: #28a745;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

#cartSummary {

margin-top: 20px;

font-weight: bold;

}

#checkoutButton {

margin-top: 20px;

background-color: #007bff;

}

#checkoutButton:hover {

background-color: #0056b3;

}

**JavaScript (script.js):**

class Product {

constructor(id, name, price, stock) {

this.id = id;

this.name = name;

this.price = price;

this.stock = stock;

}

updateStock(quantity) {

if (quantity <= this.stock) {

this.stock -= quantity;

} else {

console.log("Not enough stock available.");

}

}

isAvailable(quantity) {

return this.stock >= quantity;

}

}

class CartItem {

constructor(product, quantity) {

this.product = product;

this.quantity = quantity;

}

updateQuantity(newQuantity) {

this.quantity = newQuantity;

}

getTotalPrice() {

return this.product.price \* this.quantity;

}

}

class ShoppingCart {

constructor() {

this.items = [];

this.taxRate = 0.07; // 7% tax

this.discount = 0.1; // 10% discount for orders above $100

}

addItem(product, quantity) {

const existingItem = this.items.find(item => item.product.id === product.id);

if (existingItem) {

existingItem.updateQuantity(existingItem.quantity + quantity);

} else {

const cartItem = new CartItem(product, quantity);

this.items.push(cartItem);

}

product.updateStock(quantity);

}

removeItem(productId) {

this.items = this.items.filter(item => item.product.id !== productId);

}

updateItemQuantity(productId, newQuantity) {

const cartItem = this.items.find(item => item.product.id === productId);

if (cartItem) {

cartItem.updateQuantity(newQuantity);

}

}

calculateTotal() {

let subtotal = 0;

this.items.forEach(item => {

subtotal += item.getTotalPrice();

});

const discount = subtotal > 100 ? subtotal \* this.discount : 0;

const tax = subtotal \* this.taxRate;

return {

subtotal: subtotal.toFixed(2),

discount: discount.toFixed(2),

tax: tax.toFixed(2),

total: (subtotal - discount + tax).toFixed(2),

};

}

displayCart() {

const cartDiv = document.getElementById('cart');

cartDiv.innerHTML = '';

this.items.forEach(item => {

const cartItemDiv = document.createElement('div');

cartItemDiv.classList.add('cart-item');

cartItemDiv.innerHTML = `

${item.product.name} - $${item.product.price} x ${item.quantity} = $${item.getTotalPrice().toFixed(2)}

<button onclick="removeFromCart(${item.product.id})">Remove</button>

`;

cartDiv.appendChild(cartItemDiv);

});

const summary = this.calculateTotal();

document.getElementById('cartSummary').innerHTML = `

Subtotal: $${summary.subtotal}<br>

Discount: -$${summary.discount}<br>

Tax: $${summary.tax}<br>

<strong>Total: $${summary.total}</strong>

`;

}

}

const cart = new ShoppingCart();

const products = [

new Product(1, "Laptop", 1200, 5),

new Product(2, "Smartphone", 800, 10),

new Product(3, "Headphones", 150, 20),

new Product(4, "Monitor", 300, 8),

];

// Display products

function displayProducts() {

const productDiv = document.getElementById('products');

products.forEach(product => {

const productItemDiv = document.createElement('div');

productItemDiv.classList.add('product-item');

productItemDiv.innerHTML = `

${product.name} - $${product.price} (Stock: ${product.stock})

<button onclick="addToCart(${product.id}, 1)">Add to Cart</button>

`;

productDiv.appendChild(productItemDiv);

});

}

function addToCart(productId, quantity) {

const product = products.find(p => p.id === productId);

if (product && product.isAvailable(quantity)) {

cart.addItem(product, quantity);

cart.displayCart();

} else {

alert("Not enough stock available.");

}

}

function removeFromCart(productId) {

cart.removeItem(productId);

cart.displayCart();

}

document.getElementById('checkoutButton').addEventListener('click', function () {

const summary = cart.calculateTotal();

alert(`Your total is $${summary.total}. Thank you for your purchase!`);

});

// Initialize

displayProducts();

cart.displayCart();

**Explanation:**

* **Classes**:
  1. **Product**: Represents a product in the store. It tracks the stock, updates it when items are purchased, and checks if enough stock is available.
  2. **CartItem**: A product added to the shopping cart. It holds the product and the quantity and calculates the total price for that item.
  3. **ShoppingCart**: Manages the cart items, calculates the total, and applies discounts and taxes. It also handles adding/removing items and updating quantities.
* **Cart Management**:
  1. Users can add products to the cart as long as there is stock available.
  2. The cart automatically updates quantities for existing items.
  3. Users can remove products from the cart or update quantities.
  4. The total is updated in real time, applying a discount if the subtotal exceeds $100 and calculating taxes at 7%.
* **UI**:
  1. Products are dynamically displayed with options to add them to the cart.
  2. The cart section shows the items in the cart with the option to remove them.
  3. A checkout button calculates the final total (including tax and discounts) and alerts the user when they complete the purchase.

This exercise involves complex class interactions and mimics real-world e-commerce functionality. It handles cart management, inventory updates, discount application, and tax calculations.

**Exercise 19: Online Library Management System**

In this exercise, you will build a multi-class Online Library Management System that allows users to search for books, borrow them, and return them. The system will include functionalities for both users and library admins, such as managing books (adding, editing, deleting) and handling book borrowing and returns. It will also track the availability of books.

**Features:**

* **Admin Panel**: Admins can add, update, and remove books from the library.
* **User Interface**: Users can browse and search for books, borrow available books, and return borrowed books.
* **Book Availability**: Tracks the availability of each book and prevents borrowing if the book is already checked out.
* **Borrowing History**: Shows users the books they have borrowed and their due dates.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Online Library Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2>Online Library Management System</h2>

<div id="bookCatalog">

<h3>Available Books</h3>

<div id="books"></div>

</div>

<div id="userSection">

<h3>User Dashboard</h3>

<div id="borrowedBooks"></div>

<button id="returnAllButton">Return All Books</button>

</div>

<div id="adminSection">

<h3>Admin Panel</h3>

<form id="addBookForm">

<input type="text" id="bookTitle" placeholder="Book Title" required>

<input type="text" id="bookAuthor" placeholder="Author" required>

<input type="number" id="bookCopies" placeholder="Number of Copies" required>

<button type="submit">Add Book</button>

</form>

<div id="adminBooks"></div>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 1000px;

margin: 0 auto;

}

#bookCatalog, #userSection, #adminSection {

margin-bottom: 30px;

}

#books, #adminBooks, #borrowedBooks {

background-color: #f4f4f4;

padding: 20px;

border-radius: 5px;

}

.book-item, .borrowed-book, .admin-book {

margin-bottom: 10px;

}

button {

padding: 10px;

background-color: #28a745;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

form {

margin-bottom: 20px;

}

#returnAllButton {

margin-top: 20px;

background-color: #007bff;

}

#returnAllButton:hover {

background-color: #0056b3;

}

**JavaScript (script.js):**

class Book {

constructor(id, title, author, copies) {

this.id = id;

this.title = title;

this.author = author;

this.copies = copies;

this.borrowed = 0;

}

borrowBook() {

if (this.copies > this.borrowed) {

this.borrowed += 1;

} else {

console.log("No copies available to borrow.");

}

}

returnBook() {

if (this.borrowed > 0) {

this.borrowed -= 1;

}

}

isAvailable() {

return this.copies > this.borrowed;

}

getAvailability() {

return `${this.copies - this.borrowed}/${this.copies} available`;

}

}

class BorrowedBook {

constructor(book, borrowDate, dueDate) {

this.book = book;

this.borrowDate = borrowDate;

this.dueDate = dueDate;

}

}

class Library {

constructor() {

this.books = [];

this.borrowedBooks = [];

}

addBook(title, author, copies) {

const id = this.books.length + 1;

const book = new Book(id, title, author, copies);

this.books.push(book);

}

borrowBook(bookId) {

const book = this.books.find(b => b.id === bookId);

if (book && book.isAvailable()) {

book.borrowBook();

const today = new Date();

const dueDate = new Date(today);

dueDate.setDate(today.getDate() + 14); // 2 weeks borrow period

const borrowedBook = new BorrowedBook(book, today, dueDate);

this.borrowedBooks.push(borrowedBook);

} else {

alert("Book is not available for borrowing.");

}

}

returnAllBooks() {

this.borrowedBooks.forEach(borrowedBook => {

borrowedBook.book.returnBook();

});

this.borrowedBooks = [];

}

displayBooks() {

const booksDiv = document.getElementById('books');

booksDiv.innerHTML = '';

this.books.forEach(book => {

const bookItemDiv = document.createElement('div');

bookItemDiv.classList.add('book-item');

bookItemDiv.innerHTML = `

<strong>${book.title}</strong> by ${book.author} - ${book.getAvailability()}

<button onclick="borrowBook(${book.id})">Borrow</button>

`;

booksDiv.appendChild(bookItemDiv);

});

}

displayBorrowedBooks() {

const borrowedBooksDiv = document.getElementById('borrowedBooks');

borrowedBooksDiv.innerHTML = '';

this.borrowedBooks.forEach(borrowedBook => {

const borrowedBookDiv = document.createElement('div');

borrowedBookDiv.classList.add('borrowed-book');

borrowedBookDiv.innerHTML = `

<strong>${borrowedBook.book.title}</strong> - Borrowed on: ${borrowedBook.borrowDate.toDateString()},

Due: ${borrowedBook.dueDate.toDateString()}

`;

borrowedBooksDiv.appendChild(borrowedBookDiv);

});

}

displayAdminBooks() {

const adminBooksDiv = document.getElementById('adminBooks');

adminBooksDiv.innerHTML = '';

this.books.forEach(book => {

const adminBookDiv = document.createElement('div');

adminBookDiv.classList.add('admin-book');

adminBookDiv.innerHTML = `

<strong>${book.title}</strong> by ${book.author} - ${book.getAvailability()}

`;

adminBooksDiv.appendChild(adminBookDiv);

});

}

}

const library = new Library();

document.getElementById('addBookForm').addEventListener('submit', function (e) {

e.preventDefault();

const title = document.getElementById('bookTitle').value;

const author = document.getElementById('bookAuthor').value;

const copies = document.getElementById('bookCopies').value;

library.addBook(title, author, parseInt(copies));

library.displayBooks();

library.displayAdminBooks();

this.reset();

});

function borrowBook(bookId) {

library.borrowBook(bookId);

library.displayBooks();

library.displayBorrowedBooks();

}

document.getElementById('returnAllButton').addEventListener('click', function () {

library.returnAllBooks();

library.displayBorrowedBooks();

library.displayBooks();

});

// Initialize the library system

library.displayBooks();

library.displayAdminBooks();

**Explanation:**

* **Classes**:
  1. **Book**: Represents a book in the library. It has properties for id, title, author, copies, and borrowed. Methods allow borrowing and returning books, as well as checking availability.
  2. **BorrowedBook**: Represents a book borrowed by a user. It tracks the book, the borrow date, and the due date.
  3. **Library**: Manages all books and borrowed books. It allows for adding new books (admin functionality), borrowing and returning books, and displaying the list of books in the catalog and borrowed books.
* **Admin Functionality**:
  1. The admin can add books by filling in the form with the book title, author, and the number of copies available.
  2. Once added, books are displayed in both the user catalog and the admin panel.
* **User Functionality**:
  1. Users can borrow books by clicking the "Borrow" button, provided the book is available.
  2. Borrowed books are displayed in the user dashboard with their borrow date and due date.
  3. Users can return all borrowed books by clicking the "Return All Books" button.
* **UI**:
  1. The available books are displayed in the catalog for users to browse and borrow.
  2. Borrowed books are shown with their details, including the borrow and due dates.
  3. The admin section allows adding new books and viewing the current inventory.

This is a more complex system, involving multiple classes for handling books, borrowing, and library management. It incorporates admin features, user interactions, and real-time updates of book availability.

**Exercise 20: Event Management System**

In this exercise, you'll build an Event Management System that allows users to browse, register, and manage events. It includes functionality for both event organizers (admins) to create and manage events and users to register and view their upcoming events.

**Features:**

* **Admin Panel**: Admins can create, update, and delete events.
* **User Interface**: Users can view upcoming events, register for an event, and see their registered events.
* **Event Registration**: Tracks the capacity of each event and prevents overbooking.
* **Event Management**: Admins can add or remove events, and users can view details about upcoming events, including dates, location, and capacity.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Event Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2>Event Management System</h2>

<div id="eventCatalog">

<h3>Upcoming Events</h3>

<div id="events"></div>

</div>

<div id="userSection">

<h3>User Dashboard</h3>

<div id="registeredEvents"></div>

<button id="cancelAllButton">Cancel All Registrations</button>

</div>

<div id="adminSection">

<h3>Admin Panel</h3>

<form id="addEventForm">

<input type="text" id="eventName" placeholder="Event Name" required>

<input type="date" id="eventDate" required>

<input type="text" id="eventLocation" placeholder="Location" required>

<input type="number" id="eventCapacity" placeholder="Capacity" required>

<button type="submit">Add Event</button>

</form>

<div id="adminEvents"></div>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 1000px;

margin: 0 auto;

}

#eventCatalog, #userSection, #adminSection {

margin-bottom: 30px;

}

#events, #adminEvents, #registeredEvents {

background-color: #f9f9f9;

padding: 20px;

border-radius: 5px;

}

.event-item, .registered-event, .admin-event {

margin-bottom: 15px;

}

button {

padding: 10px;

background-color: #007bff;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

form {

margin-bottom: 20px;

}

#cancelAllButton {

margin-top: 20px;

background-color: #dc3545;

}

#cancelAllButton:hover {

background-color: #c82333;

}

**JavaScript (script.js):**

class Event {

constructor(id, name, date, location, capacity) {

this.id = id;

this.name = name;

this.date = new Date(date);

this.location = location;

this.capacity = capacity;

this.registrations = 0;

}

registerUser() {

if (this.registrations < this.capacity) {

this.registrations += 1;

} else {

console.log("No spots available.");

}

}

cancelRegistration() {

if (this.registrations > 0) {

this.registrations -= 1;

}

}

isAvailable() {

return this.registrations < this.capacity;

}

getAvailability() {

return `${this.capacity - this.registrations} spots left`;

}

}

class RegisteredEvent {

constructor(event, registrationDate) {

this.event = event;

this.registrationDate = registrationDate;

}

}

class EventManager {

constructor() {

this.events = [];

this.registeredEvents = [];

}

addEvent(name, date, location, capacity) {

const id = this.events.length + 1;

const event = new Event(id, name, date, location, capacity);

this.events.push(event);

}

registerForEvent(eventId) {

const event = this.events.find(e => e.id === eventId);

if (event && event.isAvailable()) {

event.registerUser();

const today = new Date();

const registeredEvent = new RegisteredEvent(event, today);

this.registeredEvents.push(registeredEvent);

} else {

alert("Event is fully booked.");

}

}

cancelAllRegistrations() {

this.registeredEvents.forEach(registeredEvent => {

registeredEvent.event.cancelRegistration();

});

this.registeredEvents = [];

}

displayEvents() {

const eventsDiv = document.getElementById('events');

eventsDiv.innerHTML = '';

this.events.forEach(event => {

const eventItemDiv = document.createElement('div');

eventItemDiv.classList.add('event-item');

eventItemDiv.innerHTML = `

<strong>${event.name}</strong> - ${event.date.toDateString()} at ${event.location}

<span>${event.getAvailability()}</span>

<button onclick="registerForEvent(${event.id})">Register</button>

`;

eventsDiv.appendChild(eventItemDiv);

});

}

displayRegisteredEvents() {

const registeredEventsDiv = document.getElementById('registeredEvents');

registeredEventsDiv.innerHTML = '';

this.registeredEvents.forEach(registeredEvent => {

const registeredEventDiv = document.createElement('div');

registeredEventDiv.classList.add('registered-event');

registeredEventDiv.innerHTML = `

<strong>${registeredEvent.event.name}</strong> - Registered on: ${registeredEvent.registrationDate.toDateString()}

`;

registeredEventsDiv.appendChild(registeredEventDiv);

});

}

displayAdminEvents() {

const adminEventsDiv = document.getElementById('adminEvents');

adminEventsDiv.innerHTML = '';

this.events.forEach(event => {

const adminEventDiv = document.createElement('div');

adminEventDiv.classList.add('admin-event');

adminEventDiv.innerHTML = `

<strong>${event.name}</strong> on ${event.date.toDateString()} at ${event.location}

<span>${event.getAvailability()}</span>

`;

adminEventsDiv.appendChild(adminEventDiv);

});

}

}

const eventManager = new EventManager();

document.getElementById('addEventForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('eventName').value;

const date = document.getElementById('eventDate').value;

const location = document.getElementById('eventLocation').value;

const capacity = document.getElementById('eventCapacity').value;

eventManager.addEvent(name, date, location, parseInt(capacity));

eventManager.displayEvents();

eventManager.displayAdminEvents();

this.reset();

});

function registerForEvent(eventId) {

eventManager.registerForEvent(eventId);

eventManager.displayEvents();

eventManager.displayRegisteredEvents();

}

document.getElementById('cancelAllButton').addEventListener('click', function () {

eventManager.cancelAllRegistrations();

eventManager.displayRegisteredEvents();

eventManager.displayEvents();

});

// Initialize the event system

eventManager.displayEvents();

eventManager.displayAdminEvents();

**Explanation:**

**Classes:**

1. **Event**:
   * Represents an event with properties like name, date, location, and capacity.
   * The registerUser method adds a registration if the event has available spots.
   * The cancelRegistration method removes a registration, ensuring spots are freed up.
   * It also has a method isAvailable to check if the event can accept more registrations and getAvailability to display remaining spots.
2. **RegisteredEvent**:
   * Represents a user's registration for an event, tracking both the event and the registration date.
3. **EventManager**:
   * Manages all events and registered events.
   * Admins can add new events, while users can register for them.
   * It also handles canceling all registrations for a user and displays all events for both the user and the admin.

**Functionality:**

* **Admin Features**:
  + Admins can create new events by filling out a form specifying the event's name, date, location, and capacity.
  + The admin section shows all events, allowing the admin to view their capacity and registrations.
* **User Features**:
  + Users can register for events by clicking the "Register" button, provided the event has available spots.
  + Users can view their registered events in the dashboard.
  + They can also cancel all registrations by clicking the "Cancel All Registrations" button.
* **UI**:
  + Displays all upcoming events with their details, such as date, location, and availability.
  + Users can see which events they have registered for and manage their registrations from the user dashboard.
  + Admins can manage events by viewing the event details and registration status.

This exercise expands on OOP principles by integrating more complex interactions between multiple classes, offering functionalities for both admins and users, and handling dynamic data such as event registrations and availability.

**Exercise 21: Library Management System**

This exercise involves creating a **Library Management System** where both librarians (admins) can add, update, and remove books, while users can borrow, return, and view borrowed books.

**Features:**

* **Admin Panel**: Librarians (admins) can add, update, and delete books in the library.
* **User Interface**: Users can browse books, borrow and return books, and view their borrowed books.
* **Book Availability**: The system tracks book availability and prevents borrowing if a book is already taken.
* **Borrow Limit**: Users can borrow a maximum of 3 books at a time.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Library Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1>Library Management System</h1>

<div id="bookCatalog">

<h2>Available Books</h2>

<div id="books"></div>

</div>

<div id="userSection">

<h2>User Dashboard</h2>

<div id="borrowedBooks"></div>

<button id="returnAllButton">Return All Books</button>

</div>

<div id="adminSection">

<h2>Admin Panel</h2>

<form id="addBookForm">

<input type="text" id="bookTitle" placeholder="Title" required>

<input type="text" id="bookAuthor" placeholder="Author" required>

<input type="number" id="bookCopies" placeholder="Number of Copies" required>

<button type="submit">Add Book</button>

</form>

<div id="adminBooks"></div>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 1000px;

margin: 0 auto;

}

#bookCatalog, #userSection, #adminSection {

margin-bottom: 30px;

}

#books, #adminBooks, #borrowedBooks {

background-color: #f4f4f4;

padding: 20px;

border-radius: 5px;

}

.book-item, .borrowed-book, .admin-book {

margin-bottom: 15px;

}

button {

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

form {

margin-bottom: 20px;

}

#returnAllButton {

margin-top: 20px;

background-color: #f44336;

}

#returnAllButton:hover {

background-color: #d32f2f;

}

**JavaScript (script.js):**

class Book {

constructor(id, title, author, totalCopies) {

this.id = id;

this.title = title;

this.author = author;

this.totalCopies = totalCopies;

this.borrowedCopies = 0;

}

borrowBook() {

if (this.isAvailable()) {

this.borrowedCopies += 1;

} else {

console.log("No copies available.");

}

}

returnBook() {

if (this.borrowedCopies > 0) {

this.borrowedCopies -= 1;

}

}

isAvailable() {

return this.borrowedCopies < this.totalCopies;

}

getAvailability() {

return `${this.totalCopies - this.borrowedCopies} copies available`;

}

}

class BorrowedBook {

constructor(book, borrowDate) {

this.book = book;

this.borrowDate = borrowDate;

}

}

class LibraryManager {

constructor() {

this.books = [];

this.borrowedBooks = [];

}

addBook(title, author, totalCopies) {

const id = this.books.length + 1;

const book = new Book(id, title, author, parseInt(totalCopies));

this.books.push(book);

}

borrowBook(bookId) {

const book = this.books.find(b => b.id === bookId);

if (book && book.isAvailable() && this.borrowedBooks.length < 3) {

book.borrowBook();

const today = new Date();

const borrowedBook = new BorrowedBook(book, today);

this.borrowedBooks.push(borrowedBook);

} else {

alert("Cannot borrow more books or book is not available.");

}

}

returnAllBooks() {

this.borrowedBooks.forEach(borrowedBook => {

borrowedBook.book.returnBook();

});

this.borrowedBooks = [];

}

displayBooks() {

const booksDiv = document.getElementById('books');

booksDiv.innerHTML = '';

this.books.forEach(book => {

const bookItemDiv = document.createElement('div');

bookItemDiv.classList.add('book-item');

bookItemDiv.innerHTML = `

<strong>${book.title}</strong> by ${book.author}

<span>${book.getAvailability()}</span>

<button onclick="borrowBook(${book.id})">Borrow</button>

`;

booksDiv.appendChild(bookItemDiv);

});

}

displayBorrowedBooks() {

const borrowedBooksDiv = document.getElementById('borrowedBooks');

borrowedBooksDiv.innerHTML = '';

this.borrowedBooks.forEach(borrowedBook => {

const borrowedBookDiv = document.createElement('div');

borrowedBookDiv.classList.add('borrowed-book');

borrowedBookDiv.innerHTML = `

<strong>${borrowedBook.book.title}</strong> - Borrowed on: ${borrowedBook.borrowDate.toDateString()}

`;

borrowedBooksDiv.appendChild(borrowedBookDiv);

});

}

displayAdminBooks() {

const adminBooksDiv = document.getElementById('adminBooks');

adminBooksDiv.innerHTML = '';

this.books.forEach(book => {

const adminBookDiv = document.createElement('div');

adminBookDiv.classList.add('admin-book');

adminBookDiv.innerHTML = `

<strong>${book.title}</strong> by ${book.author}

<span>${book.getAvailability()}</span>

`;

adminBooksDiv.appendChild(adminBookDiv);

});

}

}

const libraryManager = new LibraryManager();

document.getElementById('addBookForm').addEventListener('submit', function (e) {

e.preventDefault();

const title = document.getElementById('bookTitle').value;

const author = document.getElementById('bookAuthor').value;

const totalCopies = document.getElementById('bookCopies').value;

libraryManager.addBook(title, author, parseInt(totalCopies));

libraryManager.displayBooks();

libraryManager.displayAdminBooks();

this.reset();

});

function borrowBook(bookId) {

libraryManager.borrowBook(bookId);

libraryManager.displayBooks();

libraryManager.displayBorrowedBooks();

}

document.getElementById('returnAllButton').addEventListener('click', function () {

libraryManager.returnAllBooks();

libraryManager.displayBorrowedBooks();

libraryManager.displayBooks();

});

// Initialize the library system

libraryManager.displayBooks();

libraryManager.displayAdminBooks();

**Explanation:**

**Classes:**

1. **Book**:
   * Represents a book in the library with properties such as title, author, and totalCopies.
   * The borrowBook method allows users to borrow a book if there are available copies.
   * The returnBook method allows users to return a borrowed book.
   * The isAvailable method checks if there are any available copies, and getAvailability shows how many copies are left.
2. **BorrowedBook**:
   * Represents a user's borrowed book with information about the book and the date it was borrowed.
3. **LibraryManager**:
   * Manages the library's books and borrowed books.
   * Admins can add new books to the library using the form.
   * Users can borrow books and return them.
   * There is a limit of 3 borrowed books per user, and users cannot borrow a book if no copies are available.

**Functionality:**

* **Admin Features**:
  + Admins can add new books to the library by specifying the title, author, and number of copies.
  + The admin section displays all available books with their availability status (number of copies left).
* **User Features**:
  + Users can borrow books by clicking the "Borrow" button, as long as the book has available copies and the user hasn't exceeded the borrow limit.
  + Users can see which books they have borrowed and return all books at once using the "Return All Books" button.
* **UI**:
  + Displays available books with information about their title, author, and availability.
  + Users can see their borrowed books and manage their borrowing and returning.

This system expands on object-oriented concepts with multiple classes and tracks interactions between books and users. It introduces more complex user limitations, such as borrowing limits and availability checks, making the system more comprehensive.

**Exercise 22: Event Management System**

This exercise involves creating an **Event Management System** where admins can add, update, and remove events, and users can register for events. The system will track registration limits and prevent overbooking. It will also allow users to cancel registrations and view the events they are registered for.

**Features:**

* **Admin Panel**: Admins can add, update, and delete events.
* **User Registration**: Users can register for events if available and cancel their registrations.
* **Event Capacity**: The system tracks event capacity and prevents registrations if the event is full.
* **User Dashboard**: Users can view events they are registered for and manage their registrations.

**Solution:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Event Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1>Event Management System</h1>

<div id="eventCatalog">

<h2>Available Events</h2>

<div id="events"></div>

</div>

<div id="userSection">

<h2>User Dashboard</h2>

<div id="registeredEvents"></div>

<button id="cancelAllButton">Cancel All Registrations</button>

</div>

<div id="adminSection">

<h2>Admin Panel</h2>

<form id="addEventForm">

<input type="text" id="eventName" placeholder="Event Name" required>

<input type="text" id="eventDate" placeholder="Event Date" required>

<input type="number" id="eventCapacity" placeholder="Capacity" required>

<button type="submit">Add Event</button>

</form>

<div id="adminEvents"></div>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

.container {

width: 1000px;

margin: 0 auto;

}

#eventCatalog, #userSection, #adminSection {

margin-bottom: 30px;

}

#events, #adminEvents, #registeredEvents {

background-color: #f4f4f4;

padding: 20px;

border-radius: 5px;

}

.event-item, .registered-event, .admin-event {

margin-bottom: 15px;

}

button {

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

form {

margin-bottom: 20px;

}

#cancelAllButton {

margin-top: 20px;

background-color: #f44336;

}

#cancelAllButton:hover {

background-color: #d32f2f;

}

**JavaScript (script.js):**

class Event {

constructor(id, name, date, capacity) {

this.id = id;

this.name = name;

this.date = date;

this.capacity = capacity;

this.registeredCount = 0;

}

register() {

if (this.isAvailable()) {

this.registeredCount += 1;

} else {

console.log("Event is full.");

}

}

cancelRegistration() {

if (this.registeredCount > 0) {

this.registeredCount -= 1;

}

}

isAvailable() {

return this.registeredCount < this.capacity;

}

getAvailability() {

return `${this.capacity - this.registeredCount} spots available`;

}

}

class RegisteredEvent {

constructor(event, registrationDate) {

this.event = event;

this.registrationDate = registrationDate;

}

}

class EventManager {

constructor() {

this.events = [];

this.registeredEvents = [];

}

addEvent(name, date, capacity) {

const id = this.events.length + 1;

const event = new Event(id, name, date, parseInt(capacity));

this.events.push(event);

}

registerForEvent(eventId) {

const event = this.events.find(e => e.id === eventId);

if (event && event.isAvailable()) {

event.register();

const today = new Date();

const registeredEvent = new RegisteredEvent(event, today);

this.registeredEvents.push(registeredEvent);

} else {

alert("Event is full.");

}

}

cancelAllRegistrations() {

this.registeredEvents.forEach(registeredEvent => {

registeredEvent.event.cancelRegistration();

});

this.registeredEvents = [];

}

displayEvents() {

const eventsDiv = document.getElementById('events');

eventsDiv.innerHTML = '';

this.events.forEach(event => {

const eventItemDiv = document.createElement('div');

eventItemDiv.classList.add('event-item');

eventItemDiv.innerHTML = `

<strong>${event.name}</strong> on ${event.date}

<span>${event.getAvailability()}</span>

<button onclick="registerForEvent(${event.id})">Register</button>

`;

eventsDiv.appendChild(eventItemDiv);

});

}

displayRegisteredEvents() {

const registeredEventsDiv = document.getElementById('registeredEvents');

registeredEventsDiv.innerHTML = '';

this.registeredEvents.forEach(registeredEvent => {

const registeredEventDiv = document.createElement('div');

registeredEventDiv.classList.add('registered-event');

registeredEventDiv.innerHTML = `

<strong>${registeredEvent.event.name}</strong> - Registered on: ${registeredEvent.registrationDate.toDateString()}

`;

registeredEventsDiv.appendChild(registeredEventDiv);

});

}

displayAdminEvents() {

const adminEventsDiv = document.getElementById('adminEvents');

adminEventsDiv.innerHTML = '';

this.events.forEach(event => {

const adminEventDiv = document.createElement('div');

adminEventDiv.classList.add('admin-event');

adminEventDiv.innerHTML = `

<strong>${event.name}</strong> on ${event.date}

<span>${event.getAvailability()}</span>

`;

adminEventsDiv.appendChild(adminEventDiv);

});

}

}

const eventManager = new EventManager();

document.getElementById('addEventForm').addEventListener('submit', function (e) {

e.preventDefault();

const name = document.getElementById('eventName').value;

const date = document.getElementById('eventDate').value;

const capacity = document.getElementById('eventCapacity').value;

eventManager.addEvent(name, date, parseInt(capacity));

eventManager.displayEvents();

eventManager.displayAdminEvents();

this.reset();

});

function registerForEvent(eventId) {

eventManager.registerForEvent(eventId);

eventManager.displayEvents();

eventManager.displayRegisteredEvents();

}

document.getElementById('cancelAllButton').addEventListener('click', function () {

eventManager.cancelAllRegistrations();

eventManager.displayRegisteredEvents();

eventManager.displayEvents();

});

// Initialize the event system

eventManager.displayEvents();

eventManager.displayAdminEvents();

**Explanation:**

**Classes:**

1. **Event**:
   * Represents an event with properties such as name, date, and capacity.
   * The register method allows users to register for an event if spots are available.
   * The cancelRegistration method allows users to cancel their registration, freeing up spots.
   * The isAvailable method checks if spots are available, and getAvailability shows how many spots are left.
2. **RegisteredEvent**:
   * Represents a user's registration for an event with information about the event and the registration date.
3. **EventManager**:
   * Manages the events and user registrations.
   * Admins can add new events using the form.
   * Users can register for events and cancel all registrations at once.
   * Event registration is limited based on the event's capacity.

**Functionality:**

* **Admin Features**:
  + Admins can add new events to the system by specifying the name, date, and capacity of the event.
  + The admin section displays all events with their availability status (number of spots left).
* **User Features**:
  + Users can register for events by clicking the "Register" button, as long as the event has available spots.
  + Users can view the events they are registered for and cancel all registrations using the "Cancel All Registrations" button.
* **UI**:
  + Displays available events with information about their name, date, and availability.
  + Users can view their registered events and manage their registrations.

This system adds complexity with event management, user registration, and capacity tracking. Multiple classes handle the core functionality, providing a scalable event registration system with both admin and user interfaces.