1. Appointment Booking System with OOP

Problem Statement: Design a **hospital appointment booking system** where users can book, edit, and cancel appointments.

Key Features & Concepts:

1. **Object-Oriented Programming (OOP):**
    - Create a Patient class to store patient information and an Appointment class to handle booking details like date, time, and doctor.
    - Use methods to add, update, and delete appointments.
2. **DOM Manipulation:**
    - Dynamically create and update a table that lists all appointments.
    - Add form elements to book an appointment.
3. **Event Handling:**
    - Attach event listeners for form submissions, edit buttons, and delete buttons.
4. **Asynchronous Programming:**
    - Use setTimeout to simulate delayed responses for appointment confirmation.
5. **Storage Management:**
    - Save appointment data to localStorage so that it persists after page reloads.
6. **API Integration (Optional):**
    - Use a mock API to fetch available doctors or time slots.

2. Doctor Availability Scheduler

Problem Statement: Build a **Doctor Scheduler** where users can view available doctors and book slots based on their working hours.

Key Features & Concepts:

1. **Object-Oriented Programming (OOP):**
   - Create a Doctor class with properties like name, specialization, and availability.
   - Add methods to book, cancel, or check slot availability.
2. **DOM Manipulation:**
   - Render a weekly schedule as a table with available and booked slots highlighted.
3. **Event Handling:**
   - Add interactivity for booking slots via click events on calendar cells.
4. **Asynchronous Programming:**
   - Use setInterval to refresh availability in real-time (simulate changes with mock data).
5. **Storage Management:**
   - Store booked slots in localStorage to persist data.
6. **API Integration:**
   - Fetch a list of doctors and their schedules from a mock JSON file or API.

3. Patient Record Management

Problem Statement: Develop a **Patient Management System** to add, update, delete, and search for patient records.

Key Features & Concepts:

1. **Object-Oriented Programming (OOP):**
   - Define a Patient class with properties like id, name, age, and medicalHistory.
   - Implement methods to manage patient records (e.g., addPatient, updatePatient).
2. **DOM Manipulation:**
   - Dynamically render patient records in a table with search and edit options.
3. **Event Handling:**
   - Attach event listeners for add, update, delete, and search actions.
4. **Asynchronous Programming:**
   - Simulate fetching patient data using setTimeout or Promise.
5. **Storage Management:**
   - Store patient records in localStorage or sessionStorage.
6. **API Integration:**
   - Fetch mock patient data from a simulated REST API or JSON file.

4. Billing System with Discounts

Problem Statement: Create a **billing system** for hospital services that calculates the total cost dynamically, including taxes and discounts.

Key Features & Concepts:

1. **Object-Oriented Programming (OOP):**
    - Create a Service class to represent hospital services and a Bill class for total calculations.
    - Use methods to calculate taxes, apply discounts, and generate bill summaries.
2. **DOM Manipulation:**
    - Display a list of services with checkboxes for selection and dynamically update the bill summary.
3. **Event Handling:**
    - Handle events for service selection, applying discount codes, and generating a bill.
4. **Asynchronous Programming:**
    - Simulate fetching service data from a server using the Fetch API or Promise.
5. **Storage Management:**
    - Store selected services in sessionStorage to persist the bill during a session.
6. **API Integration:**
    - Fetch available services and their prices dynamically from a mock API.

5. Interactive Symptoms Checker with OOP

Problem Statement: Build a **symptoms checker** to recommend departments or specialists based on selected symptoms.

Key Features & Concepts:

1. **Object-Oriented Programming (OOP):**
   - Define a Symptom class and a Diagnosis class for recommendations.
   - Use methods to map symptoms to departments dynamically.
2. **DOM Manipulation:**
   - Create an interactive form with checkboxes for symptoms and display recommendations dynamically.
3. **Event Handling:**
   - Add interactivity for symptom selection and recommendations.
4. **Asynchronous Programming:**
   - Fetch symptom data from a mock JSON file or API.
5. **Storage Management:**
   - Cache previously selected symptoms using localStorage.
6. **API Integration:**
   - Fetch recommendations or symptom information from a simulated REST API.

6. Blood Bank Inventory Management

Problem Statement: Design a **Blood Bank Management System** to track blood units by type and availability.

Key Features & Concepts:

1. **Object-Oriented Programming (OOP):**
    - Create a BloodUnit class with attributes like type, quantity, and expiryDate.
    - Implement methods to add, update, or remove blood units.
2. **DOM Manipulation:**
    - Render the inventory dynamically as a table with real-time updates.
3. **Event Handling:**
    - Add interactivity for managing inventory via form submissions and table buttons.
4. **Asynchronous Programming:**
    - Simulate automatic alerts for low stock or expired units using setInterval.
5. **Storage Management:**
    - Save blood inventory in localStorage for persistence.
6. **API Integration:**
    - Fetch or post inventory data to a mock API.