**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.

**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.

**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.

**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.

**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.

**Exercise 6: Static Methods**

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

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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

**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.

**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.

**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.

**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.

**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.

**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.