"OOP in JavaScript
- What is OOP?
- Object Property and Method
- Class in JavaScript
- new keyword in JavaScript
1. What is Abstraction and how it works?
2. JavaScript Polymorphism
3. JavaScript Encapsulation
4. JavaScript Getter and Setter Method"

Task 1: Basic Object and Method

Objective: Create a simple object with methods to represent a person.

Task:

Create an object person with properties like name, age, and gender.
Implement a method introduce() that logs a message introducing the person.
Call the introduce() method to demonstrate the functionality.

Task 2: Class and Instantiation

Objective: Understand the basics of classes and instantiation.

Task:

Define a class Book with properties like title, author, and publishedYear.
Instantiate two instances of the Book class with different values.
Access and log the properties of each instance.

Task 3: Using new Keyword

Objective: Practice creating objects using the new keyword.
Task:
Define a class Dog with properties like name and breed.
Create an instance of Dog using the new keyword.
Set the properties of the dog and log them.

Task 4: Abstraction with Class Methods

Objective: Apply abstraction by defining methods within a class.
Task:
Create a class Calculator with methods like add(a, b) and subtract(a, b).
Implement these methods to perform addition and subtraction.
Demonstrate using the calculator to perform basic calculations.

Task 5: Understanding Polymorphism

Objective: Explore polymorphism with a common interface.
Task:
Define a class Shape with a method draw() that logs a message.
Create subclasses Circle and Square that extend Shape.
Implement the draw() method in each subclass to display shape-specific messages.
Create instances of both Circle and Square and call their draw() methods.

Task 6: Encapsulation with Private Properties

Objective: Understand encapsulation by defining private properties.

Task:

Create a class Person with private properties like _name and _age.
Implement getter and setter methods for name and age that validate inputs.
Use these getter and setter methods to set and retrieve the values.

Task 7: JavaScript Encapsulation with Bank Account

Objective:
Implement encapsulation by modeling a simple bank account.

Task:

Create a BankAccount class with private properties like _balance and _accountHolder.
Implement methods deposit(amount) and withdraw(amount) to modify the balance.
Ensure that withdrawal cannot exceed the current balance.
Use encapsulation to protect the internal state of the BankAccount class.

Task 8: JavaScript Getter and Setter with Student Information

Objective:
Practice using getter and setter methods with student information.

Task:

Define a Student class with private properties for name and grade.
Implement getter and setter methods for name and grade with basic validation.
Create instances of the Student class and use the getter and setter methods.

Task 9: JavaScript Inheritance with Animals

Objective:
Explore inheritance by creating a hierarchy of animal classes.

Task:

Define a base class Animal with properties like name and sound.
Create subclasses Dog and Cat that extend Animal.
Implement the makeSound() method for each subclass.
Create instances of both Dog and Cat and call their makeSound() methods.

Task 10: JavaScript Polymorphism with Shapes

Objective:
Understand polymorphism by working with a collection of shapes.

Task:

Create a base class Shape with a method calculateArea() that returns the area.
Create subclasses Circle and Rectangle that extend Shape.
Implement the calculateArea() method for each subclass.
Create an array of shapes (both circles and rectangles) and calculate the total area.

Task 11: JavaScript Abstraction with Vehicles

Objective:
Demonstrate abstraction by modeling different types of vehicles.

Task:

Define a base class Vehicle with properties like model and year.
Create subclasses Car and Motorcycle that extend Vehicle.
Implement methods for starting and stopping each type of vehicle.

Demonstrate abstraction by using these methods without exposing internal details.

Task 12: Real-life Application - Library Management System

Objective:

Apply OOP principles to model a library management system.

Task:

Identify entities like Book, Library, and Member.

Create classes for each entity with appropriate properties and methods.

Implement methods for checking out and returning books, tracking due dates, etc.

Demonstrate the interactions between different classes to model library operations.

Real-Life Example: Online Shopping Cart System

Objective:

Model an online shopping cart system using Object-Oriented Programming (OOP) principles in JavaScript.

1. Class Definitions:

Define the following classes with their properties and methods:

Product:

Properties: productId, productName, price, quantityAvailable

Methods: getDetails()

ShoppingCart:

Properties: cartItems (an array to store products in the cart)

Methods: addToCart(product, quantity), removeFromCart(productId), calculateTotal()

User:

Properties: userId, username, email

Methods: placeOrder(cart)

2. Abstraction:

Abstract the internal details of the classes. For example, the ShoppingCart class can internally manage the cart items without exposing the array directly.

3. Encapsulation:

Encapsulate the properties of the classes. For instance, use private properties in the User class and encapsulate methods within the ShoppingCart class to control access.

4. Inheritance:

Introduce inheritance by creating a specialized class, such as PremiumUser, which extends the User class. The PremiumUser class can have additional properties or methods.

5. Polymorphism:

Demonstrate polymorphism by having different types of products (e.g., physical products, digital downloads) that inherit from the Product class but may have specific behaviors.

6. Getter and Setter:

Use getter and setter methods to control and validate the values of certain properties. For example, ensure that the quantity of a product in the cart doesn't exceed the available quantity.

7. New Keyword:

Instantiate objects using the new keyword. For example, create instances of the Product, ShoppingCart, and User classes.

8. Real-life Scenario:

Product Listing:

Create instances of the Product class to represent various products available for purchase.

User Interaction:

Create instances of the User class to represent users interacting with the system.

Shopping Cart Operations:

Use the ShoppingCart class to simulate users adding products to their carts, removing items, and calculating the total amount.

Order Placement:

Use the User class to simulate a user placing an order with their shopping cart.

This example models the interactions between users, products, and a shopping cart in an online shopping environment, demonstrating how OOP principles can be applied to real-life scenarios.