**JavaScript OOP (Classes and Objects) Assignment**

**Part 1: Easy**

**1. Create a Simple Class**

* **Objective**: Define a class Car with properties and methods.
* **Instructions**:
  1. Create a class called Car that has:
     + A brand property.
     + A model property.
     + A method getCarInfo() that returns the car's brand and model as a string.
  2. Create two instances of the class Car and print their information using the method.

javascript

Copy code

class Car {

// Write the constructor and method here

}

const car1 = new Car('Toyota', 'Corolla');

const car2 = new Car('Honda', 'Civic');

console.log(car1.getCarInfo()); // Output: Toyota Corolla

console.log(car2.getCarInfo()); // Output: Honda Civic

**2. Class with Constructor**

* **Objective**: Create a class Person that initializes properties through the constructor.
* **Instructions**:
  1. Create a Person class with name and age properties.
  2. Use a constructor to initialize these properties when creating new objects.
  3. Create a method greet() that returns a greeting message like “Hello, my name is John, and I am 30 years old.”

javascript

Copy code

class Person {

// Write the constructor and method here

}

const person1 = new Person('John', 30);

console.log(person1.greet());

**3. Getters and Setters**

* **Objective**: Create a class with getter and setter methods.
* **Instructions**:
  1. Define a class Rectangle with properties width and height.
  2. Create getter methods to calculate and return the area and perimeter of the rectangle.
  3. Use setter methods to update the width and height.

javascript

Copy code

class Rectangle {

// Write the constructor, getter, and setter methods here

}

const rect = new Rectangle(5, 10);

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

console.log(rect.perimeter); // Output: 30

**Part 2: Medium**

**4. Inheritance**

* **Objective**: Implement inheritance between two classes.
* **Instructions**:
  1. Create a Animal class with properties name and type and a method describe() that returns “This is a [type] named [name].”
  2. Create a subclass Dog that inherits from Animal and adds a method bark() that returns “Woof! Woof!”
  3. Create an instance of Dog and demonstrate that it can access methods from both Animal and Dog.

javascript

Copy code

class Animal {

// Write the constructor and describe method here

}

class Dog extends Animal {

// Write the Dog class here

}

const myDog = new Dog('Buddy', 'Dog');

console.log(myDog.describe()); // Output: This is a Dog named Buddy.

console.log(myDog.bark()); // Output: Woof! Woof!

**5. Method Overriding**

* **Objective**: Override a method in a subclass.
* **Instructions**:
  1. Create a Vehicle class with a method move() that returns “The vehicle is moving.”
  2. Create a subclass Bike that overrides the move() method to return “The bike is moving.”
  3. Call the move() method from both Vehicle and Bike objects to demonstrate overriding.

javascript

Copy code

class Vehicle {

// Write the Vehicle class here

}

class Bike extends Vehicle {

// Write the Bike class with overridden method here

}

const myVehicle = new Vehicle();

const myBike = new Bike();

console.log(myVehicle.move()); // Output: The vehicle is moving.

console.log(myBike.move()); // Output: The bike is moving.

**6. Static Methods**

* **Objective**: Use static methods within a class.
* **Instructions**:
  1. Create a MathHelper class with a static method add() that returns the sum of two numbers.
  2. Demonstrate that you can call the add() method without creating an instance of the class.

javascript

Copy code

class MathHelper {

// Write the static add method here

}

console.log(MathHelper.add(5, 10)); // Output: 15

**Part 3: Hard**

**7. Private Fields (Encapsulation)**

* **Objective**: Use private fields to encapsulate data.
* **Instructions**:
  1. Create a class BankAccount with private fields #balance.
  2. The class should have methods to deposit(amount), withdraw(amount), and checkBalance() that interact with the private #balance.
  3. Ensure that balance cannot be accessed directly from outside the class.

javascript

Copy code

class BankAccount {

// Write the constructor and methods here

}

const myAccount = new BankAccount(1000);

myAccount.deposit(500);

myAccount.withdraw(200);

console.log(myAccount.checkBalance()); // Output: 1300

**8. Polymorphism (Different Forms of the Same Method)**

* **Objective**: Implement polymorphism using method overriding.
* **Instructions**:
  1. Create a Shape class with a method area() that returns 0.
  2. Create subclasses Circle, Rectangle, and Triangle, each with their own implementation of the area() method.
  3. Create instances of these shapes and demonstrate polymorphism by calling area() on each of them.

javascript

Copy code

class Shape {

// Write the Shape class here

}

class Circle extends Shape {

// Write the Circle class with area method

}

class Rectangle extends Shape {

// Write the Rectangle class with area method

}

class Triangle extends Shape {

// Write the Triangle class with area method

}

const myShapes = [new Circle(5), new Rectangle(4, 6), new Triangle(3, 4)];

myShapes.forEach(shape => console.log(shape.area()));

**9. Class Composition**

* **Objective**: Implement composition instead of inheritance.
* **Instructions**:
  1. Create a class Engine with a method start() that returns “Engine started.”
  2. Create a class Car that contains an instance of Engine and calls start() when its own start() method is called.
  3. Demonstrate how Car uses the Engine class to start the car.

javascript

Copy code

class Engine {

// Write the Engine class with start method

}

class Car {

// Write the Car class that uses Engine

}

const myCar = new Car();

console.log(myCar.start()); // Output: Engine started.

**Bonus Challenge (Optional)**

**10. Abstract Classes and Interfaces (Simulated in JavaScript)**

* **Objective**: Simulate abstract classes and interfaces in JavaScript.
* **Instructions**:
  1. Create a base class Animal that cannot be instantiated directly (simulate this by throwing an error in the constructor).
  2. Define a method speak() in Animal that must be overridden by subclasses.
  3. Create subclasses Dog and Cat that implement speak() with their own messages (“Bark” for Dog, “Meow” for Cat).

javascript

Copy code

class Animal {

// Write the Animal class here (simulate abstract)

}

class Dog extends Animal {

// Write the Dog class with speak method

}

class Cat extends Animal {

// Write the Cat class with speak method

}

const myDog = new Dog();

const myCat = new Cat();

console.log(myDog.speak()); // Output: Bark

console.log(myCat.speak()); // Output: Meow