OOPS Notes

I. Introduction

Object-Oriented Programming (OOP) is a programming paradigm that represents concepts as objects that have properties (attributes) and associated functions (methods). It emphasises the reusability of code, encapsulation, inheritance, and polymorphism.

II. Basic Concepts

Class: A blueprint for creating objects (a particular data structure). It defines a set of properties and methods that characterise any object of the class.

```
1.
2. class Dog {
3.     // Class properties and methods go here
4.     }
```

Object: An instance of a class. It has properties (data) and methods (functions) defined by the class.

```
1. const myDog = new Dog();
```

Properties: Attributes of an object. They store data values.

```
1. class Dog {
2.    constructor(name, age) {
3.        this.name = name;
4.        this.age = age;
5.    }
6. }
```

Methods: Functions associated with a class that define the behaviour of objects.

```
1.
2. class Dog {
3.     constructor(name, age) {
4.         this.name = name;
5.         this.age = age;
6.     }
7.
8.     bark() {
```

```
        console.log("Woof! My name is " + this.name);
    }
}
```

Encapsulation: The concept of bundling data and methods that act on that data within a single unit (class). It is used to hide the values or state of a structured data object inside a class, preventing unauthorised access.

Example:

```
class Dog {
    constructor(name, age) {
        this._name = name; // Private property
        this._age = age; // Private property
    }

    getName() {
        return this._name;
    }

    setName(name) {
        this._name = name;
    }
}
```

Inheritance: The process by which one class inherits properties and methods from another class. It allows creating a new class that is a modified version of an existing class.

Example:

```
class Animal {
    constructor(species) {
        this.species = species;
    }

    makeSound() {
        console.log("Some generic sound");
    }
}

class Dog extends Animal {
    constructor() {
        super("Dog");
    }
```

```
    makeSound() {
        console.log("Woof!");
    }
}
```

Polymorphism: The ability of a class to take on multiple forms. In OOP, polymorphism allows a subclass to override or extend methods from its superclass.

Example:

```
class Animal {
    makeSound() {
        console.log("Some generic sound");
    }
}

class Dog extends Animal {
    makeSound() {
        console.log("Woof!");
    }
}

class Cat extends Animal {
    makeSound() {
        console.log("Meow!");
    }
}

const myAnimal = new Animal();
const myDog = new Dog();
const myCat = new Cat();

myAnimal.makeSound(); // Some generic sound
myDog.makeSound(); // Woof!
myCat.makeSound(); // Meow!
```

Abstraction: The process of hiding complex implementation details and exposing only essential features to the user. It simplifies code by breaking it down into smaller, more manageable pieces.

Example:

```
class Shape {
    constructor(color) {
        this.color = color;
    }

    // Abstract method
    getArea() {
        throw new Error("getArea method must be implemented in subclasses");
    }
}

class Circle extends Shape {
    constructor(color, radius) {
        super(color);
        this.radius = radius;
    }

    // Implementing the abstract method
    getArea() {
        return Math.PI * this.radius * this.radius;
    }
}

class Rectangle extends Shape {
    constructor(color, width, height) {
        super(color);
        this.width = width;
        this.height = height;
    }

    // Implementing the abstract method
    getArea() {
        return this.width * this.height;
    }
}

```

Composition: The process of building complex objects by combining simpler objects. It allows creating more flexible and maintainable code by enabling a "has-a" relationship between objects.

Examples:

```
class Engine {
    start() {
        console.log("Engine starts");
    }
}

class Car {
    constructor() {
        this.engine = new Engine();
      }

      start() {
          this.engine.start();
          console.log("Car starts");
      }
  }

  const myCar = new Car();
  myCar.start(); // Output: "Engine starts" and "Car starts"

```

Static methods and properties: Methods and properties that belong to the class itself, not to the instances of the class. They are useful for creating utility functions or constants that don't depend on the state of an object.

```
class MathHelper {
    static PI = 3.141592653589793;

    static add(a, b) {
        return a + b;
    }

    static subtract(a, b) {
        return a - b;
      }
  }

  console.log(MathHelper.PI); // 3.141592653589793
  console.log(MathHelper.add(3, 4)); // 7
  console.log(MathHelper.subtract(10, 5)); // 5

```

These concepts form the foundation of Object-Oriented Programming in JavaScript. As a beginner, understanding and practising these concepts will help you create structured, reusable,

and maintainable code. As you gain more experience, you'll be able to explore more advanced OOP patterns and techniques.