**Topic: Oops – class, encapsulation**

Oops in js

OOP in JavaScript provides a way to structure and organize code in a more modular and reusable manner. By utilizing objects, classes, inheritance, encapsulation, and polymorphism.

        1.class

        2.object

        3.encapsulatiion

        4.inheritance

        5.polymorphism

        6.abstraction

Constructor class

Construtor class is a way to define a template for creating object

         \* class was introduced recently in ES6

         \* class is a collection of properties and methods(static/intance)

         \* syntax :

         \*          class Classname{

         \*           }

         \*

         \* obj syntax :

         \*        new Classname(arguments)

         \* constructor should be one

Class Definition

Class: A class is a template for creating objects. It encapsulates data and functions that operate on that data.

Properties (state): Variables that belong to the class.

Methods (behaviour): Functions that belong to the class.

Ex:-

Marker Class

The Marker class has three properties: color, lid, and type. It also has a constructor to initialize these properties.

      class Marker {

        color; // property to hold the color of the marker

        lid; // property to hold the type of lid of the marker

        type; // property to hold the type of the marker

        // Constructor to initialize the properties

        constructor(color,lid, type) {

          this.color = color; // initializes the color property

          this.lid = lid; // initializes the lid property

          this.type = type; // initializes the type property

        }

        // method initialisation - it is a behaviour of object

        write() {

            console.log("Writing with " + this.color + " marker");

        }

      }

Creating Objects

Objects are instances of a class. The new keyword is used to create an object from a class.

      var marker1 = new Marker("red", "square", "permanent");

      var marker2 = new Marker("black", "circle", "temporary");

Logging the object

     //logging the objects

     console.log(marker1);

     console.log(marker2);

Calling the method

    //calling the methods

    marker1.write();

Key Points

Class Definition: Defines the state and behavior of objects.

Properties: Variables within a class that hold data.

Methods : function that holds the behaviour

Constructor: Special method to initialize properties when an object is created.

this Keyword: Refers to the current instance of the class.

Creating Objects: Use new keyword followed by class name and arguments to create objects.

use the class keyword to create a JavaScript class.

// create a class

class Person{

    // body of class

};

To use the class constructor we must need to assign object to it

    class Person{

        constructor (name,age){

            this.name=name;

            this.age =age;

        }

        hi(a,b){

            return "my name is "+ this.name+ " age is " + this.age;

        }

     }

     var b=new Person("teja",25);

    console.log(b);//{name:teja; age:25}

    console.log(b.age);//25

    console.log(b.hi());//my name is teja age is 25

by using prototypes we can add properties directly

Person.prototype.fullName = “sai teja”

Javascript classes with private fields

Class Definition

Class: Employee

Private Fields: #name, #age, #salary (indicated by # prefix)

Introduced to provide encapsulation and data protection within a class.

Enhances security and prevents accidental modification of sensitive data.

    class Employee {

    #name;

    #age;

    #salary;

    constructor(name, age, salary) {

        this.#name = "John";    // Private field initialization

        this.#age = 30;         // Private field initialization

        this.#salary = 50000;   // Private field initialization

    }

//private fields can only accessed inside the class

  hello(){

        console.log("Hello, my name is " + this.# )

        }

   }

Private Fields

Definition: Private fields are declared using the # symbol before the field name (#name, #age, #salary).

Access Control: Private fields can only be accessed and modified from inside the class where they are defined.

Accessing object will thrown an error because we cannot able to access them in outside

    var person1=new Employee("john", 25, 50000);

    console.log(person1.name)// error - private property cannot be accessed directly

method will print the output because properties can be accessed inside a class

    person1.hello();

Encapsulation

Encapsulation in JavaScript refers to the bundling of data (attributes) and methods (functions) that operate on the data into a single unit, typically known as an object. This concept helps in hiding the internal state of an object and only exposing the necessary functionalities to interact with that state.

Access the values using getter

Syntax:

    // we can giving access

    get methodname(){

      return this.#name;

    }

    // we can giving access

    get accessname(){

      return this.#name;

    }

    class Employee {

    #name;

    #age;

    #salary;

    constructor(name, age, salary) {

        this.#name = name;    // Private field initialization

        this.#age = age;         // Private field initialization

        this.#salary = salary;   // Private field initialization

    }

    hello(){

        console.log("Hello, my name is " + this.#name + " and I am a " )

    }

    // Getter for accessing private field #name

    get accessname(){

      return this.#name;

    }

   }

    var person1=new Employee("john", 25, 50000);

    //we can able to access now because of encapsulation

    console.log(person1.accessname)// john - will not thrown an error

Modify the values using setter

    // Setter for modifying private field #salary

    set setsalary(sal){

      this.#salary=sal;

    }

    // Getter for accessing private field #salary

    get getsalary(){

      return this.#salary;

    }

   //creating a object

    var person1=new Employee("john", 25, 50000);

    //setting the values

    person1.setsalary=99500;

    //accessing the updated values

    console.log(person1.getsalary);

//complete program

    class Employee {

    #name;

    #age;

    #salary;

    constructor(name, age, salary) {

        this.#name = name;    // Private field initialization

        this.#age = age;         // Private field initialization

        this.#salary = salary;   // Private field initialization

    }

    hello(){

        console.log("Hello, my name is " + this.#name + " and I am a " )

    }

    // we can giving access

    get accessname(){

      return this.#name;

    }

    //giving the access to change the values

    set setsalary(sal){

      this.#salary=sal;

    }

    get getsalary(){

      return this.#salary;

    }

   }

   //creating a object

    var person1=new Employee("john", 25, 50000);

    //setting the values

    person1.setsalary=99500;

    //accessing the updated values

    console.log(person1.getsalary);

Certainly! Let's break down the concept of inheritance in JavaScript using the provided code as an example and create detailed notes.