CAPITULO NRO. 12 – POO – INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS.

Tipo de Clase: Teórico Práctica

Duración: 120 minutos

Objetivo de la clase:

El objetivo del presente capítulo es introducir al alumno en el amplio mundo de la programación orientada a objetos, el alumno debe comprender al finalizar la clase los conceptos de clase, objetos, atributos de la clase, función constructora, la instanciación de objetos, del poder de encapsular lógica, con atributos, métodos que engloban una determinada complejidad y resuelven un problema determinado. Luego poder convertir esos objetos a formato JSON que significa JavaScript Object Notation. Debe comprender que a lo largo de su carrera profesional seguramente interactuará en el mundo de la POO en algunos de los roles que definimos ya sea como consumidor de clases o constructor de clases que provean soluciones específicas.

Los objetos que derivan de una clase son una estructura de datos y vienen a sumarse a las que ya venimos viendo que son

Estructuras de datos: hasta este momento el alumno viene manejando estructuras de datos como ser:

- Variables
- Constantes
- Vectores

Ahora podremos incorporar los objetos como una nueva estructura de datos, una estructura que permite contener atributos, estos atributos pueden ser estructuras que ya conocemos como variables, como vectores (pero dentro de la POO se los conoce como atributos). Los objetos provienen de una clase que sería en el mundo real como un “MOLDE”.

En el mundo real por ejemplo si deseo construir block para la construcción de una casa necesito una blockera que sería el “MOLDE = CLASE”. Y los block que se construyeron a partir de ese MOLDE se denominarían objetos.

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT
MÓDULO 02 - JAVASCRIPT

CLASE

OBJETOS A PARTIR DE LA CLASE

Definición técnica de Clase en POO:

Una clase en Programación Orientada a Objetos (POO) es un modelo o una abstracción que representa un concepto o una entidad del mundo real. Esta clase encapsula datos (propiedades) y comportamientos (métodos) relacionados con ese concepto.

Definición alternativa (Profesor): una clase es como definir una estructura de datos nueva que permite incorporar y encapsular un conjunto de atributos, lógica, métodos (que son similares a las

funciones) y que en conjunto resuelven un problema determinado, encapsular significaría en este contexto englobar todo lo que sea necesario para resolver un problema.

Por ejemplo, se pueden construir clases que permiten a partir de series de números obtener todas las medidas estadísticas de dispersión como, por ejemplo, la media aritmética, la moda, la mediana, la desviación standard.

La idea de encapsular (atributos + métodos específicos) tiene la intención de englobar una problemática en particular y que en ella se encuentre toda la solución respecto a esa temática. Muchos programadores tienen experiencia y conocimiento respecto de temas particulares y que requieren mucha capacitación al respecto, como, por ejemplo.

Ejemplos de clases

- Ejemplos de clases o conjunto de clases nativas de JavaScript, que el lenguaje ya trae incorporadas y que uno utiliza normalmente y esto nos abstrae de cómo están implementados esos métodos.
    - String
    - Array
        - Push
        - Pop
        - forEach
        - map
        - filter
    - Promise
    - Fetch

- Existen Programadores ó empresas que construyen clases que encapsulan toda la lógica del Sistema Financiero y lógicamente venden y comercializan esas soluciones.
- Existen Programadores ó empresas que construyen clases que encapsulan toda la lógica de gráficos estadísticos.
- Existen Programadores ó empresas que construyen clases que encapsulan toda la lógica para manipular motores o componentes electro mecánicos. Esto para los programadores que se encargan de la robótica los abstrae completamente de cómo está implementado cada método que tiene ese conjunto de clases.

Es decir, nosotros como programadores podemos tener dos roles

Consumidor de Objetos: instanciados a partir clases que engloban comportamiento específico.

Constructor de Clases: que encapsulan complejidad y estarán provistos para que otros programadores puedan resolver problemas de esa temática sin necesidad de comprender a fondo los detalles de la implementación de las soluciones.

Rol de Consumidor de Clases

Los desarrolladores que asumen el rol de consumidores de clases utilizan las clases creadas por otros (terceros) para construir sus aplicaciones. Estos desarrolladores pueden no necesitar

entender completamente los detalles internos de cómo funcionan estas clases, siempre y cuando puedan interactuar con ellas de manera efectiva a través de sus interfaces públicas. Este enfoque promueve la reutilización de código, la abstracción, modularidad y estandarización.

Rol de Creador de Clases

Por otro lado, los desarrolladores que asumen el rol de creadores de clases son aquellos que tienen un conocimiento específico en un área o dominio y construyen clases que encapsulan esa lógica. Estos desarrolladores tienen la responsabilidad de diseñar clases que sean fáciles de usar y que oculten la complejidad interna detrás de una interfaz clara y bien definida. Su objetivo es proporcionar a otros desarrolladores una abstracción limpia y efectiva para trabajar con conceptos complejos o especializados.

Importancia de Ambos Roles

Ambos roles son igualmente importantes en el desarrollo de software orientado a objetos. Los consumidores de clases se benefician del trabajo de los creadores de clases al utilizar sus abstracciones para construir aplicaciones de manera eficiente y efectiva. Por otro lado, los creadores de clases juegan un papel fundamental al proporcionar soluciones especializadas y encapsular conocimiento específico del dominio, lo que facilita la construcción de sistemas complejos y la colaboración entre equipos multidisciplinarios.

Tipos de OBJETOS

En JavaScript tenemos dos tipos de objetos, los objetos literales que no necesitan una clase para ser instanciados, solamente con definirlos el lenguaje los interpreta como tal y aquellos que necesitan clases para darle la forma, con sus atributos, constructor, etc.

Objetos Literales:

```
const objetoLiteral = {
  nombre: 'Juan',
  edad: 30
};
```

Objetos Instanciados a partir de clase:

```
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
```

```
    }
}

const objetoInstanciado = new Persona('Juan', 30);
```

Podríamos definir que existen dos tipos de clases y por ende sus objetos

- Clases estructurales
  - Permiten ENCAPSULAR almacenar datos de cosas, hechos, entidades como por ejemplo datos de personas, datos de registración de eventos (compras, ventas, entradas y salidas de un depósito). Para este tipo de datos no es necesario construir una clase y luego instanciar la clase, simplemente podemos construir los objetos con sus atributos y valores.

    Es por ello que los denominamos “objetos literales”.

```
  const Persona1 = {Nombre:"MESSI LIONEL ANDRES",Domicilio:"MIAMI 824"};
    console.log(Persona1);

    /* (02) - Definimos otro objeto literal */

    const Persona2 = {Nombre:"RIQUELME JUAN ROMAN",Domicilio:"CIUDAD
AUTONOMA DE BS.AS."};
    console.log(Persona2);

    /* (03) - Definimos un objeto literal con un método a dentro */

    const Factura1 =
{FechaCompra:"21/03/2020",CantidadProductos:5,PrecioUnitario:2500.20,getTota
l(){return this.CantidadProductos * this.PrecioUnitario}};
    console.log(Factura1);
    console.log(Factura1.getTotal()); // invocamos el método que está dentro
del objeto

    /* (04) - Convertimos el OBJETO JAVASCRIPT A JSON => que quiere decir
JAVASCRIPT OBJET NOTATION  */

    console.log(JSON.stringify(Persona1));
    console.log(JSON.stringify(Persona2));

    /* cuando convierte un objeto a JSON no transforma la función interna ó
método */
    console.log(JSON.stringify(Factura1));
```

- Clases funcionales

- Permiten ENCAPSULAR almacenar atributos y lógica que tiene como objetivo realizar cálculos que son complejos o que requieren un conocimiento profundo del tema, es decir por ejemplo clases que se dedican a partir de números obtener medidas estadísticas de dispersión por dar un ejemplo.

A Continuación, realizaremos la construcción de una clase que tendrá como funcionalidad encapsular la lógica de sumar dos números. Es decir, a partir de esa clase se podrán obtener

Múltiples objetos que tendrán la lógica lista para encapsular:

```
class SumaDeDos
{
    // los atributos de la clase
    Numero1 = 0;
    Numero2 = 0;
    Resultado = 0; // un atributo de la clase

    /* metodo constructor de la clase permite
    inicializar los atributos de la clase */
    constructor(p1,p2)
    {
        this.Numero1 = p1;
        this.Numero2 = p2;
        this.Sumar();
    }

    /* Getters y Setters */

    // método que me permitirá retornar el Numero1 //
    getNumero1()
    {
        return this.Numero1;
    }

    // método que me permitirá retornar el Numero2 //
    getNumero2()
    {
        return this.Numero2;
    }

    // método que me permitirá modificar el Numero1 //
    setNumero1(parametro)
    {
```

```
        this.Numero1 = parametro; // actualizo el atributo y por ende
actualizo el resultado */
        this.Sumar(); // al cambiar el valor
    }

    // método que me permitirá modificar el Numero2 //
    setNumero2(parametro)
    {
        this.Numero2 = parametro; // actualizo el atributo y por ende tengo
que actualizar el resultado */
        this.Sumar();
    }

    Sumar()
    {
        this.Resultado = this.Numero1 + this.Numero2;
    }

    getResultado()
    {
        this.Sumar();
        return this.Resultado;
    }
}
```

Forma de instanciar los objetos a partir de la clase SumaDeDos

```
// creamos un objeto a partir de la clase SumaDeDos //

    let objeto1 = new SumaDeDos(10,20);
    console.log(objeto1.getResultado());

    let objeto2 = new SumaDeDos(22,34);
    console.log(objeto2.getResultado());

    objeto2.setNumero1(115);
    console.log(objeto2.getResultado());
```

Partes de una Clase

- Atributos de una clase: son lugares, estructuras de datos (“variables”) que pueden ser booleanos, numéricos, String u otros objetos que permiten almacenar datos que son importantes para el objetivo de la clase.
    - Públicos: pueden ser accedidos y modificados desde a fuera de la clase
    - Privados: pueden ser modificados y accedidos solo por métodos internos de la clase
    - Estáticos: son atributos que permanecerán estáticos en todas las instancias, son útiles para mantener información única para todas las instancias. Es decir, son atributos que pertenecen a la clase y no a la instancia. Si bien cada instancia puede acceder a dichos atributos, los mismos son únicos y si se modifican en una instancia, esa modificación se reflejará en todas las instancias.

- El Constructor: es un método (“función”) que tiene como objetivo asignarles valores a los atributos de la clase. Siempre y cuando no sean estáticos.
- Métodos de la clase: es comportamiento, procesos, (“funciones”) que tienen como objetivo realizar cálculos, operaciones con los atributos o hacer cálculos específicos y devolverlos al programa principal que instanció el objeto.
    - Públicos: en JavaScript todos los métodos por defecto son públicos.
    - Privados: no soportados en JavaScript
    - Estáticos: métodos que tienen acceso a atributos estáticos ya sea para asignar valores o para retornar los valores.
    - Getters y Setters: conjunto de métodos (“funciones”) que se encargan de asignarle valor a cada atributo y también a devolver valores de los atributos.
        - Cada atributo tendrá dos métodos, 1 método get y 1 método set.

Ejemplo de Clases y Bibliotecas de Clases de Terceros que ayudan a la tarea cotidiana del programador:

Express.Router (Express.js): Express.js es un marco web de Node.js que se utiliza para construir aplicaciones web y APIs. Express.Router es una clase que se utiliza para crear objetos de enrutador en una aplicación Express, lo que permite organizar las rutas de manera modular.

React.Component (React): React es una biblioteca de JavaScript para construir interfaces de usuario. React.Component es una clase base que se utiliza para crear componentes personalizados en aplicaciones React. Estos componentes encapsulan la lógica y la interfaz de usuario relacionada en una sola unidad.

Mongoose.Model (Mongoose): Mongoose es una biblioteca de modelado de datos para MongoDB en Node.js. Mongoose.Model es una clase que se utiliza para definir modelos de datos en una aplicación Node.js y proporciona métodos para interactuar con la base de datos MongoDB.

Chart.js: Chart.js es una biblioteca JavaScript simple pero poderosa para crear gráficos interactivos y responsivos en páginas web. Es fácil de usar y ofrece una amplia variedad de tipos de gráficos, como barras, líneas, radar, dispersión, etc.

D3.js (Data-Driven Documents): D3.js es una biblioteca de JavaScript para manipular documentos basados en datos. Si bien es más avanzada y flexible que otras bibliotecas, también puede ser más compleja de aprender. Se utiliza para crear visualizaciones de datos personalizadas y altamente interactivas.

Highcharts: Highcharts es una biblioteca de gráficos JavaScript que ofrece una amplia gama de gráficos y visualizaciones, incluidos histogramas, gráficos de dispersión, gráficos de líneas, gráficos de áreas y más. Es fácil de configurar y tiene una amplia documentación.

Google Charts: Google Charts es una biblioteca gratuita de visualización de datos desarrollada por Google. Ofrece una variedad de gráficos y visualizaciones, como gráficos de barras, líneas, áreas, histogramas, dispersión, entre otros. Es fácil de usar y se integra bien con otros servicios de Google.

Plotly.js: Plotly.js es una biblioteca de gráficos de código abierto que permite crear gráficos interactivos y personalizables en páginas web. Ofrece soporte para una variedad de tipos de gráficos, incluidos histogramas, gráficos de dispersión, gráficos de líneas, gráficos de barras y más.

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT
MÓDULO 02 - JAVASCRIPT

Conceptos Avanzados de POO

Herencia Extender las clases utilizando la palabra extends.

Herencia Método super();

Composición versus Herencia:

Definición de interfaces (no soportado nativamente 100% con JavaScript):

Conjunto de métodos que no tienen implementación, no tienen código, pero las clases que se adhieran a esa interfaz deberán implementar la solución a esos métodos.

Definición de clases abstractas:

No existe un concepto directo de "clase abstracta" como en otros lenguajes de programación orientada a objetos, como Java o C#. Sin embargo, es posible emular el comportamiento de una clase abstracta utilizando técnicas disponibles en JavaScript.

Una clase abstracta es una clase que no puede ser instanciada directamente y que generalmente contiene métodos abstractos, es decir, métodos que deben ser implementados por las clases que la heredan. En JavaScript.

LINK PROYECTO:

https://drive.google.com/file/d/16cymrqyYPWgokYLYiw3RZPxorsYjD9Xn/view?usp=sharing