APLICANDO LO APRENDIDO 3

Ejercicio 1

Considera el lenguaje JavaScript acotado al paradigma de programación orientada a objetos basado en prototipos y analízalo en términos de los cuatro componentes de un paradigma mencionados por Kuhn.

1. Generalización simbólica: ¿Cuáles son las reglas escritas del lenguaje?
2. Creencias de los profesionales: ¿Qué características particulares del lenguaje se cree que sean "mejores" que en otros lenguajes?
3. Valores: ¿Qué pensamiento o estilo de programación consideraron mejor los creadores.
4. Ejemplares: ¿Qué clase de problemas pueden resolverse más fácilmente en el lenguaje?

1) Las reglas estipuladas de este paradigma aplicado en JavaScript introduce el concepto de objeto como reemplazo de las clases en el orientado a clases, y cada objeto tiene una colección de pares clave-valor conocidos como propiedades, que pueden ser modificados en tiempo de ejecución, a diferencia de las instancias de clases, que no pueden modificar sus propiedades dinámicamente.
   Cada objeto puede heredar de otro a través de un prototipo, que es como un objeto padre, y a la vez ese prototipo tiene su propio prototipo, así sucesivamente se crea una cadena prototípica hasta llegar a un objeto que hereda null, lo cual termina esa cadena de herencias.
   La creación de "objetos tipo clase" se realizaba mediante funciones constructoras y la palabra clave `new`, o utilizando el método `object.create(nombrePrototipo)`.

2) Los profesionales creen en que la capacidad de modificar objetos en tiempo de ejecución y crear estructuras sin una jerarquía rígida permite resolver problemas de forma más rápida y creativa. Además, se cree que la herencia basada en prototipos es más sencilla y flexible que los

modelos basados en clases, sin la necesidad de definir clases complejas y estructurar sus herencias.

3) En vez de imponer una estructura rígida de clases, JavaScript enfatiza objetos dinámicos y relaciones prototípicas, lo que se alinea con un estilo ágil y práctico para programar.

4) Resuelve problemas más fácilmente cuando se requiere crear estructuras cambiantes en tiempo real (por ejemplo, modificar objetos JSON recibidos del servidor), la herencia prototípica permite hacerlo con naturalidad.

Ejercicio 4

Explica en un texto, con ejemplos y fundamentación, qué características de la OOP utilizaste para resolver los programas de los Ejercicios 2 y 3. Si hay alguna que no utilizaste o no implementaste, indica cuál y por qué crees que no fue necesario.

Respuesta

- He utilizado la característica de la abstracción en ambos ejercicios.

La abstracción es clave en el proceso de análisis y diseño orientado a objetos, pues nos permitirá armar un conjunto de objetos que permitan representar el problema que se busca resolver a través de la implementación de software.

En el caso del ejercicio 2, pude distinguir objetos como los operandos, las operaciones, la calculadora, y funciones auxiliares para crear una calculadora funcional, sin saber exactamente cómo funciona una calculadora real, evitando horas de estudio sobre cómo funciona internamente una calculadora y que circuitos posee por dentro, la idea es enfocarse en los elementos esenciales que serían ingresar números, realizar sumas, restas, multiplicaciones y divisiones, reduciendo la complejidad interna del ejercicio.

En el caso del ejercicio 3, una clienta nos pide realizar una app para registrar tareas diarias, donde pude distinguir objetos como las tareas que representan el elemento principal que la app necesita gestionar, los menús que se muestran en la app, las funcionalidades que representan las acciones que requiere registrar tareas y herramientas auxiliares para resolver el problema.
Con estos objetos limitados a un concepto clave se simplifica la idea de cómo llevar a cabo una app, preservando información relevante para resolver el problema.

- He utilizado la característica de la encapsulación en ambos ejercicios.

El encapsulamiento se refiere a la agrupación de datos con los métodos que operan en esos datos. Básicamente, implica entender a los objetos como cápsulas que agrupan datos y métodos estrechamente relacionados, también refiere a la restricción del acceso directo a algunos de los componentes de un objeto, como los datos, evitando exponer detalles propios de la implementación y protegiendo de cambios no deseados.

En el ejercicio 2, cree cápsulas que almacenan datos y operaciones sobre esos datos, como por ejemplo la cápsula operandos que guarda datos sobre números y acciones que se pueden realizar con esos números como mostrarlos, obtenerlos, inicializarlos a un valor nulo y darle valores a esos números.
Además, limité el acceso a esos números con la palabra clave protected para evitar que otras cápsulas no puedan acceder a este dato, exceptuando las cápsulas que heredan de los operandos.

En el ejercicio 3, hice lo mismo, agrupando los datos de una tarea como título y descripción, y las acciones o comportamientos que se pueden realizar con los datos de una tarea como ingresar un título o ingresar una descripción. Aunque no limité el acceso directo a las cápsulas, pues en este ejercicio se basa en OOP basada en prototipos, donde no hay un concepto formal de encapsulamiento como en la OPP basada en clases. Los atributos y métodos suelen ser públicos por defecto.

- He utilizado la característica de la herencia en ambos ejercicios.

La herencia es el mecanismo de la OOP que se utiliza para alcanzar la reutilización y la extensibilidad. Facilita la creación de objetos a partir de otros ya existentes e implica que un objeto puede obtener atributos y comportamiento desde otro objeto del que hereda.

En el ejercicio 2, cree una cadena de herencias: operandos, operaciones, calculadora. Esto permite que las operaciones (sumar, restar, multiplicar, dividir) también tengan asociados operandos con los que pueden realizarse,y a la vez la calculadora hereda ya las operaciones y los operandos para poder mostrarle al usuario los cálculos apropiadamente.

Esto ayuda a facilitar la estructura del código y reutilizar métodos y datos de superclases, promoviendo la prolijidad y escalabilidad. Por ejemplo, si luego el ejercicio pidiera que agreguemos la operación potencia bastaría agregar un método en operaciones para ello y la calculadora lo podría realizar, ya que es clase hija de operaciones.

En el ejercicio 3, hice que el objeto menú tenga un prototipo del objeto auxiliar, que tiene los métodos para ingresar por teclado y demás, facilitando la implementación de los métodos que posee menú, usando las funciones de auxiliar para usar los menús, extendiendo las propiedades, además si se agrega un método más al objeto auxiliar, menú lo podría utilizar ya que es su prototipo. Fomentando el manejo de propiedades extensibles y la reutilización.

- No he utilizado la característica del polimorfismo en ninguno de los ejercicios

El polimorfismo en OOP es un principio que permite que objetos de diferentes clases sean tratados como objetos de una clase común. Esencialmente, permite que una misma operación o método se ejecute de diferentes formas según el objeto que lo invoque.

En el ejercicio 2, no utilicé polimorfismo ya que si bien podría crear una clase abstracta que se llame operación que tenga un método abstracto llamado operación y crear una clase por cada operación que define su propio uso del método abstracto, esto aumentaría la cantidad de clases que tendría en el proyecto y considere que serian demasiados archivos y lo vi innecesario.

En el ejercicio 3, no utilicé polimorfismo porque los objetos que detecté en la abstracción para resolver el problema son muy específicos y no son tan parecidos entre ellos, pero juntos permiten que la app funcione, si hubieran distintos tipos de tareas, podría permitir una mayor flexibilidad en el código usando el mismo método de varias formas para cada tipo de tarea, pero en este caso la clienta nos dijo que las tareas eran todas iguales.

- He utilizado la característica de la modularidad en ambos ejercicios.

Se conoce como modularidad a la capacidad de un sistema de ser dividido en partes más pequeñas que deben comportarse como una caja negra y ser tan independientes como sea posible de la aplicación y de las otras partes.

Gracias a la modularidad un sistema puede ser visto o entendido como la unión de varias partes que interactúan entre sí y que trabajan solidariamente para alcanzar un objetivo común, realizando cada una de ellas una tarea necesaria para la consecución de dicho objetivo.

En el ejercicio 2, considere útil el uso de la modularidad creando módulos para los números, las operaciones, y el uso de ellos en la calculadora, todos esos módulos convergen en index donde se ejecuta el programa, permitiendo ver la resolución de crear una calculadora como la unión de varias partes, y facilitar la revisión del código por partes.

En el ejercicio 3, procedí de la misma manera, incluso modularice las funciones que usará index para que este archivo no sea tan grande.

- No utilicé la característica del principio de ocultación en ninguno de los ejercicios.

Según el principio de ocultación, cada objeto es un módulo natural aislado del exterior que expone una interfaz que detalla cómo pueden interactuar con él otros objetos. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas.

En el ejercicio 2, simulé el principio de ocultación creando métodos para acceder a la lista de números en los operandos, con getOperandos y setOperandos, la calculadora pueden ingresar números a Operandos a través de setOperandos(aunque sea un atributo que puede acceder directamente ya que es clase nieta de Operandos).

En el ejercicio 3, simulé el principio de ocultación al crear métodos que permitan el ingreso de los datos de una tarea, en vez de ingresar directamente el título accediendo a la propiedad, se debe ingresar al metodo ingresarTitulo, que seria la interfaz que muestra un

objeto creado con el prototipo de tarea para que las funciones externas puedan modificar el título.