TALLER TYPESCRIPT

PRESENTADO POR:

LUDWYNG JENSER ORDOÑEZ GARCIA

PRESENTADO PARA:

ING. BRAYAN ARCOS

INSTITUTO TECNOLOGICO DEL PUTUMAYO

MOCOA/PUTUMAYO

2024

INDICE

1. Declaración de Variables 3
2. Función para Calcular el Área de un Rectángulo 3
3. Implementación de la Clase Car 3
4. Interfaz y Clase Rectángulo 4
5. Filtrado de Libros y Gestión de la Biblioteca 5
6. Uso de Bucles para Procesar Números 5
Conclusión 5

Este informe presenta un análisis y desglose de la implementación de código en TypeScript proporcionada. El código abarca varios conceptos clave de programación como la declaración de variables, creación de funciones, estructuras de clases, implementación de interfaces y aplicaciones prácticas utilizando bucles y condicionales. Además, demuestra la organización del código siguiendo las mejores prácticas en TypeScript.

1. Declaración de Variables

El código comienza declarando variables utilizando diferentes tipos primitivos en TypeScript. Estos incluyen `string`, `number`, `undefined`, `null` y `boolean`. Cada variable se imprime en la consola para confirmación:

```
const Firstname: string = "Ludwyng";
const age: number = 19;
const toy: undefined = undefined;
const lol: null = null;
const state: boolean = true;
```

2. Función para Calcular el Área de un Rectángulo

Se define una función llamada `calculateRectangleArea` para calcular el área de un rectángulo. La función acepta dos parámetros, `base` y `height`, ambos de tipo `number`, y devuelve su producto. Esta función demuestra el uso adecuado del sistema de tipos de TypeScript:

```
function calculateRectangleArea(base: number, height: number): number {
    return base * height;
}
```

3. Implementación de la Clase Car

La clase `Car` se crea con cuatro propiedades privadas: `make`, `model`, `color`, y `brand`. Incluye un constructor para inicializar estas propiedades y un método `displayCarInfo` para imprimir la información del carro en un formato estructurado:

```
class Car {
    private make: string;
```

```
    private model: number;
    private color: string;
    private brand: string;

    constructor(make: string, model: number, color: string, brand: string) {
        this.make = make;
        this.model = model;
        this.color = color;
        this.brand = brand;
    }

    displayCarInfo(): void {
        console.log(`El año del carro es ${this.make}, el modelo es ${this.model}, su color es ${this.color}, la marca es ${this.brand}`);
    }
}
```

4. Interfaz y Clase Rectángulo

Se define una interfaz llamada `Shape` con un método `calculateShapeArea`. La clase `Rectangle` implementa esta interfaz, proporcionando la lógica para calcular el área de un rectángulo utilizando los parámetros `base` y `height`:

```
interface Shape {
    calculateShapeArea(): number;
}

class Rectangle implements Shape {
    base: number;
    height: number;

    constructor(base: number, height: number) {
        this.base = base;
        this.height = height;
    }

    calculateShapeArea(): number {
        return this.base * this.height;
```

```
    }
}
```

5. Filtrado de Libros y Gestión de la Biblioteca

El código introduce una colección de libros almacenados en un arreglo. El arreglo se filtra para obtener libros escritos por un autor específico (`J.K. Rowling`). Este proceso de filtrado utiliza el método `filter` de TypeScript para coincidir los libros según su propiedad `author`:

```
const books: { title: string, author: string, year: number }[] = [
    { title: "Harry Potter y la piedra filosofal", author: "J.K. Rowling", year: 1997 },
    { title: "Harry Potter y la cámara secreta", author: "J.K. Rowling", year: 1998 },
    { title: "Harry Potter y el prisionero de Azkaban", author: "J.K. Rowling", year: 1999 }
];

const filterBooksByAuthor = books.filter(book => book.author === "J.K. Rowling");
```

Además, se implementa una clase `Library` para gestionar una colección de libros. Proporciona dos métodos: `addNewBook` para agregar libros y `searchBooksByAuthor` para recuperar libros de un autor específico.

6. Uso de Bucles para Procesar Números

Se utilizan varias estructuras de bucles para realizar tareas aritméticas y lógicas básicas. Estas incluyen sumar todos los números pares del 1 al 100, contar los números impares entre 1 y 30 usando un bucle `while`, e imprimir números primos entre 1 y 100 usando bucles anidados y condicionales.

Conclusión

Este código en TypeScript demuestra una variedad de conceptos de programación, desde declaraciones de variables y funciones hasta programación orientada a objetos basada en clases e implementación de interfaces. Proporciona una base sólida en la sintaxis y aplicación de TypeScript, ilustrando

cómo estos conceptos trabajan juntos para resolver desafíos comunes de programación.

Link de repositorio

https://github.com/Ludwyng06/ProBacked.git

Link de Video

https://drive.google.com/drive/folders/15fOzxvuOWfhy8Gc7N_P5Rq38cbGNbJFJ?usp=drive_link