

Clase 4 y 5: Material Complementario

1. TypeScript

TypeScript

Definición:

TypeScript es un superset de JavaScript que introduce un sistema de tipos estáticos. Esto significa que puedes definir tipos de datos para variables, funciones y objetos, lo que ayuda a detectar errores durante la escritura del código en lugar de en tiempo de ejecución.

Beneficios:

- **Detección de Errores:** Gracias al tipado estático, TypeScript puede identificar errores comunes en el código antes de que se ejecute, lo que mejora la calidad del software.
- **Autocompletado y Herramientas:** Los editores de código pueden ofrecer autocompletado y mejores herramientas de refactorización gracias a la información de tipos.
- **Interoperabilidad:** TypeScript es completamente compatible con JavaScript, lo que permite a los desarrolladores adoptar gradualmente TypeScript en proyectos existentes.

Programación Orientada a Objetos (POO)

Definición: La POO es un paradigma de programación que se basa en la utilización de objetos.

Organiza el software en torno a estos objetos en lugar de funciones y lógica, facilitando la gestión del código y su mantenimiento.

Principios Fundamentales:

- **Modelado de la Realidad:** La POO permite representar entidades del mundo real en el software. Cada objeto tiene características (atributos) y comportamientos (métodos) que reflejan esas entidades.
- **Modularidad:** Las clases proporcionan una estructura modular al código, lo que facilita su organización y mantenimiento. Cada clase puede desarrollarse y probarse de forma independiente.

Conceptos Clave de POO

1. Clase:

- **Definición:** Una clase es un molde o plantilla que define un conjunto de atributos y métodos. Actúa como una especificación para crear objetos.
- **Instanciación:** Cuando se crea un objeto a partir de una clase, se dice que se está instanciando. Por ejemplo, al crear un objeto Persona, estamos generando una instancia de la clase Persona.

2. Instancia:

- **Definición:** Una instancia es un objeto que ha sido creado a partir de una clase. Cada instancia tiene sus propios valores para los atributos definidos en la clase, lo que permite la representación de múltiples entidades del mismo tipo.
- **Ejemplo:** Si Persona es una clase, entonces `const javier = new Persona("Javier", 32, new Date(1992, 9, 10));` crea una instancia llamada javier.

3. Método:

- **Definición:** Un método es una función definida dentro de una clase que describe los comportamientos de los objetos de esa clase.
- **Ejemplo:** En la clase Persona, el método `saludar()` permite al objeto saludar a otros. La sintaxis se asemeja a las funciones regulares, pero están asociadas a la clase y pueden acceder a los atributos de la misma.

4. Objeto:

- **Definición:** Un objeto es una instancia de una clase. Representa una entidad del mundo real y puede tener atributos y métodos asociados.
- **Características:** Cada objeto tiene sus propios valores para sus atributos. Por ejemplo, dos instancias de Persona pueden tener diferentes nombres y edades.

Ejemplo de Clase en TypeScript

```
class Persona {  
    // Atributo privado  
    private nombre: string; // Atributo privado  
    private edad: number; // Atributo privado  
    private fechaNac: Date; // Atributo privado  
    constructor(nombre: string, edad: number, fechaNac: Date) {  
        this.nombre = nombre; this.edad = edad; this.fechaNac = fechaNac;  
    }  
    public saludar(): void {  
        console.log("Hola " + this.nombre);  
    }  
}  
  
// Crear una instancia del objeto  
const javier = new Persona("Javier", 32, new Date(1992, 9, 10));  
javier.saludar(); // "Hola Javier"
```

Los Cuatro Conceptos Fundamentales de POO

1. Encapsulamiento:

- **Definición:** Es el principio de ocultar los detalles internos del objeto y exponer solo lo necesario a través de métodos públicos. Esto significa que los atributos de un objeto pueden estar protegidos de accesos directos, permitiendo que se modifiquen solo a través de métodos controlados.

Tipos de Encapsulamiento:

- **Public (por defecto):** Los atributos y métodos son accesibles desde cualquier parte del código. Este es el modificador de acceso por defecto.
- **Private:** Los atributos y métodos solo son accesibles dentro de la misma clase, protegiendo así los datos de accesos no deseados.
- **Protected:** Similar a private, pero permite que las clases que heredan de ella también puedan acceder a esos atributos y métodos. Esto es útil en la herencia.

Beneficios:

- **Protección de Datos:** Ayuda a mantener la integridad de los datos del objeto, evitando modificaciones indeseadas.
- **Control de la Lógica de Acceso:** Permite validar los datos antes de asignarlos a los atributos, asegurando que se mantengan dentro de un rango o formato válidos.
- **Facilidad de Mantenimiento:** Se pueden cambiar detalles internos de la implementación sin afectar el código externo que utiliza el objeto.

2 . Herencia:

- **Definición:** La herencia permite que una clase (subclase o clase hija) herede propiedades y métodos de otra clase (clase padre o superclase). Esto permite reutilizar código y crear una jerarquía de clases.

Beneficios:

- **Reutilización del Código:** Las subclases pueden utilizar y extender funcionalidades de las superclases, evitando la duplicación de código.
- **Relaciones Jerárquicas:** Facilita la organización del código en una estructura más lógica y jerárquica, donde las subclases pueden especializarse a partir de características comunes de las superclases.

3 . Polimorfismo:

- **Definición:** El polimorfismo permite que una variable, función o método adopte múltiples formas. Esto significa que el mismo método puede comportarse de diferentes maneras en diferentes contextos.
- **Ejemplo:** En una jerarquía de clases, una función puede tomar como argumento un objeto de una clase base y, dependiendo del tipo real del objeto, ejecutar un comportamiento específico.

- **Sobreescritura:** Esto se refiere a redefinir métodos heredados en las clases hijas para proporcionar implementaciones específicas. Por ejemplo, una clase Perro que hereda de Animal puede sobreescribir un método hacerSonido() para ladrar en lugar de emitir un sonido genérico.

4. Abstracción:

- **Definición:** La abstracción se refiere a representar solo los detalles esenciales de un objeto, ocultando su complejidad interna. Se centra en los aspectos relevantes que son importantes para el contexto en el que se utiliza.
- **Enfoque:** Permite que los desarrolladores interactúen con el objeto a un nivel más alto, sin necesidad de preocuparse por la implementación interna. Esto se logra a menudo a través de clases abstractas e interfaces, que definen métodos que deben ser implementados por las clases concretas.

Ejemplo de Herencia y Polimorfismo

```
class Animal {
    ... protected nombre: string;
    ... constructor(nombre: string) {
    ...     this.nombre = nombre;
    ... }

    ... public hacerSonido(): void {
    ...     console.log(`${this.nombre} hace un sonido.`);
    ... }
}

class Perro extends Animal {
    ... public hacerSonido(): void {
    ...     console.log(`${this.nombre} ladra.`);
    ... }
}

class Gato extends Animal {
    ... public hacerSonido(): void {
    ...     console.log(`${this.nombre} maulla.`);
    ... }
}

// Uso del polimorfismo
const animales: Animal[] = [new Perro("Fido"), new Gato("Miau")];
animales.forEach(animale => animale.hacerSonido());
// Salida:
// "Fido ladra."
// "Miau maulla."
```

Como manejar typescript en visual studio code

Hace falta aclarar...

Dependencies : Son piezas de software que nos van a acompañar en todos los ambientes ya sea productivo o desarrollo. Son necesarias para que la aplicación funcione en producción principalmente.

devDependencies : Nos acompañan solamente en el desarrollo. Solo se necesitan en el proceso de desarrollo. ejemplos: lint (revisa integridad del código), testing, compiladores y transpiladores.

Comandos

- **Iniciar proyecto con node** `npm init -y`

- **Instalar dependencies:**

Typescript se instala de forma global:

`npm install -g typescript`

`npm i ts-node` `npm i -D @types/node`

- **Inicializar tsconfig.json**

Este archivo permite generar el archivo de configuraciones para ts:

Hay dos opciones: `npx ts-config init`

ó `npx --package typescript tsc --init`

- **Al final para correr el archivo ts necesitamos ejecutar** `npx ts-node index`

- Como ver las dependencias globales que tenemos en nuestra computadora:

Las dependencias globales están disponibles para todos los proyectos que realices en tu computadora
: `npm ls -g`

Extra

Interfaces:

- Las interfaces definen los contratos que las clases deben seguir. Especifican la firma de los métodos y las propiedades que una clase concreta debe implementar, proporcionando una forma de lograr polimorfismo de inclusión.

Mensajería entre Objetos:

- En POO, los objetos interactúan entre sí enviándose mensajes. Estos mensajes consisten en la invocación de métodos en otros objetos. La mensajería es un componente clave para lograr la encapsulación y la interacción entre objetos.