**CONCEPTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS EN TYPESCRIPT**

**ALAN GABRIEL GUTIÉRREZ PRADA**

**CENTRO DE COMERCIO Y TURISMO**

**ANÁLISIS Y DESARROLLO DE SOFWARE**

**FICHA: 3144886**

**EDWING ROZO**

**ARMENIA-QUINDIO**

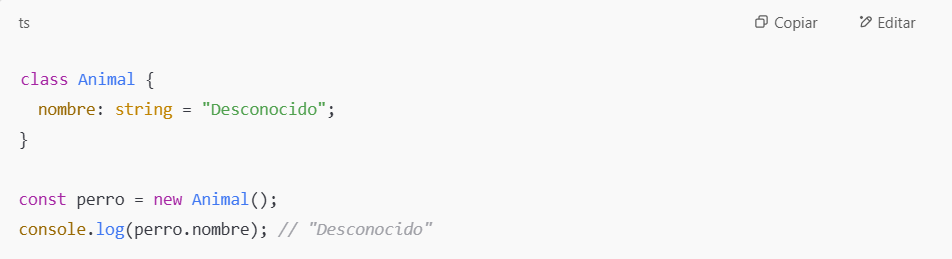
**MAYO DE 2025**

**CONCEPTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS EN TYPESCRIPT**

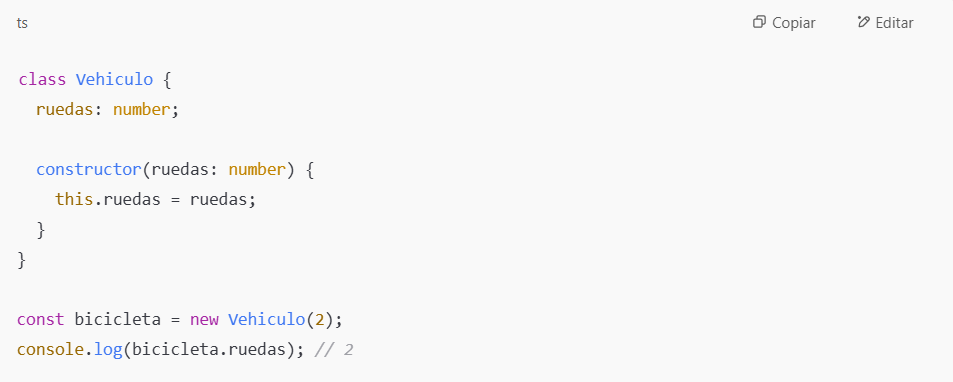
1. **CLASES**

Una clase en TypeScript es una plantilla para crear objetos. Representa entidades con características (propiedades) y comportamientos (métodos). Las clases organizan el código usando el paradigma de programación orientada a objetos (POO), permitiendo una estructura más clara, reutilizable y mantenible. Las clases también permiten herencia, encapsulamiento y otros principios de la POO.

**Ejemplo 1:**



**Ejemplo 2:**



1. **PROPIEDADES**

Las propiedades son variables que pertenecen a una clase y describen el estado o atributos del objeto. Se pueden declarar directamente dentro de la clase o ser asignadas dentro del constructor. Las propiedades pueden tener modificadores de acceso (como public o private) para controlar su visibilidad.

**Ejemplo 1:**



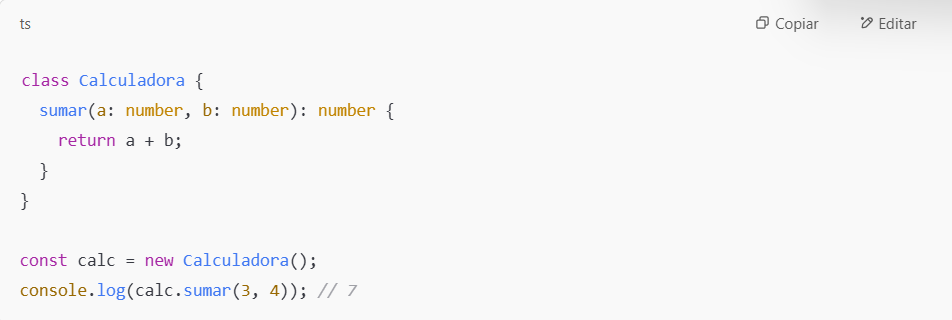
**Ejemplo 2:**



1. **MÉTODOS**

Los métodos son funciones definidas dentro de una clase y representan las acciones que un objeto puede realizar. Un método puede tener parámetros, devolver valores y acceder a las propiedades del objeto usando **“this”**. Son fundamentales para expresar el comportamiento de los objetos.

**Ejemplo 1:**



**Ejemplo 2:**



1. **MODIFICADORES DE ACCESO**

Los modificadores de acceso determinan la visibilidad y accesibilidad de propiedades y métodos dentro de una clase. **“public”** permite acceso desde cualquier parte del código, **“private”** restringe el acceso solo a la misma clase, y **“protected”** permite el acceso desde la clase y sus subclases. Esto favorece el encapsulamiento y protege los datos.

**Ejemplo 1:**



**Ejemplo 2:**



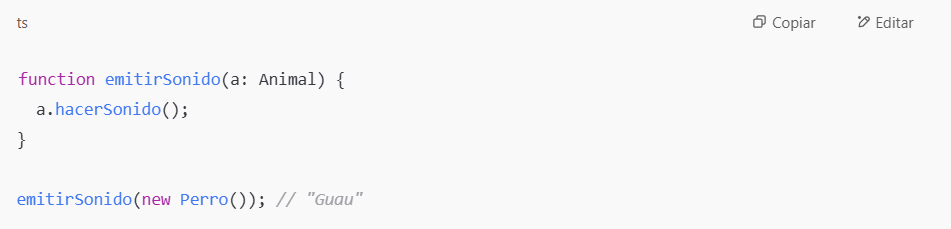
1. **POLIMORFISMO**

El polimorfismo permite que múltiples clases implementen métodos con el mismo nombre pero comportamientos diferentes. Esto facilita el uso de objetos diferentes de forma uniforme, lo cual es útil al escribir código genérico que puede adaptarse a distintos tipos de objetos.

**Ejemplo 1:**



**Ejemplo 2:**



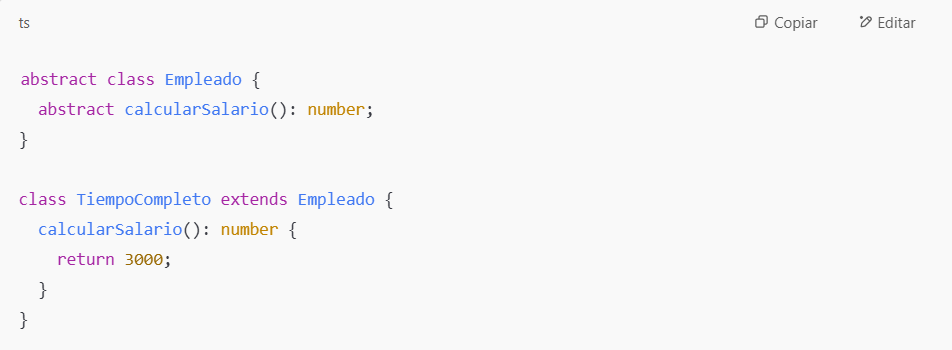
1. **ABSTRACCIÓN**

La abstracción permite ocultar los detalles internos de implementación de un objeto y exponer solo lo necesario. Esto se puede lograr con clases abstractas o interfaces. Es útil para diseñar sistemas complejos, enfocándose en el 'qué hace' un objeto y no en el 'cómo lo hace'.

**Ejemplo 1:**



**Ejemplo 2:**



1. **HERENCIA**

La herencia permite que una clase (subclase) herede propiedades y métodos de otra (superclase). Esto permite reutilizar código y establecer relaciones jerárquicas entre clases. La subclase puede extender o modificar el comportamiento de la clase base.

**Ejemplo 1:**



**Ejemplo 2:**



1. **MÉTODO CONSTRUCTOR**

El constructor es un método especial que se ejecuta automáticamente al crear una instancia de la clase. Se utiliza para inicializar propiedades o realizar configuraciones iniciales del objeto.

**Ejemplo 1:**



**Ejemplo 2:**



1. **INTERFACES**

Una interfaz en TypeScript define un contrato que una clase debe cumplir. Describe las propiedades y métodos que deben estar presentes en una clase o en un objeto. Las interfaces promueven una programación flexible y desacoplada, ya que una clase puede implementar múltiples interfaces.

**Ejemplo 1:**



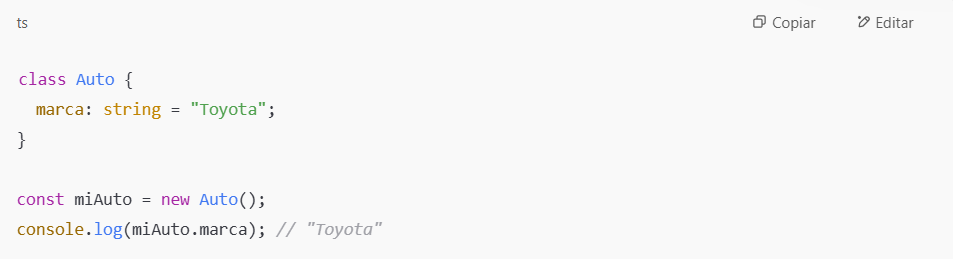
**Ejemplo 2:**



1. **INSTANCIA**

Una instancia es un objeto real creado a partir de una clase. Cada vez que usamos `new Clase()`, estamos creando una nueva instancia. Cada instancia tiene su propio espacio de memoria para almacenar valores de sus propiedades, aunque comparta la misma estructura de clase.

**Ejemplo 1:**



**Ejemplo 2:**

