Curso CFP

CFP Programador full-stack

Repaso General de POO

Agenda

- Organización de una Clase
- Composición
- Herencia
- Interfaces
- Patrones de Diseño
- Manejo de Errores
- Recomendaciones Generales
- Ejercicios

Organización de una Clase

Variables Internas

Constructor

Métodos Públicos

Métodos Privados

```
export default class Recta {
    private puntoA: Punto;
    private puntoB: Punto;

    public constructor(puntoA: Punto, puntoB: Punto) {
        this.asignarValores();
    }

    public getPuntoA(): Punto {
        return this.puntoA;
    }

    public getPuntoB(): Punto {
        return this.puntoB;
    }

    private asignarValores(puntoA: Punto, puntoB: Punto) {
        this.puntoA = puntoA;
        this.puntoB = puntoB;
    }
}
```

Este es el orden que siempre se tiene que respetar para que la clase sea legible

Composición

- Se usa cuando queremos que una clase compleja esté *compuesta* por clases más sencillas
  - Un Auto compuesto por Motor, Rueda, Puerta, etc.
  - Un Televisor compuesto por Botón, Pantalla, etc.
  - Un Telefono compuesto por Pantalla, Cámara, Altavoz, Botón, etc.
  - Una clase Biblioteca compuesta por las clases Libro, Cliente, etc.
- Básicamente una composición es que una clase sencilla figure como *variable interna* de otra clase

Herencia

- Se usa cuando queremos que una clase haga lo mismo que otra, pero *agregando/modificando* funcionalidad
  - Un AutoDeportivo hace lo mismo que Auto, pero modifica la forma en que acelera
  - Un SmartTV hace lo mismo que un televisor común, pero agrega funciones como por ejemplo la conexión a internet
- Tener en cuenta el modificador *protected*
  - Es como el *private*, pero haciendo que las subclases puedan verlo

Interfaces

- Funcionan como un “contrato” que debe cumplir una determinada clase
- Se especifican los métodos pero sin comportamiento, es decir sin codificar lo que hacen
- Las clases que *implementen* a las interfaces tienen la obligación de implementar todos los métodos definidos en la interfaz
- Son una muy buena práctica ya que ayudan a plantear lo que debería hacer una clase

Patrones de Diseño

- Muchos de los problemas que tenemos, son problemas conocidos
- Por lo tanto, problemas conocidos → soluciones conocidas
- Un *patrón de diseño* se le llama a una solución conocida
- Cada patrón tiene una situación específica en donde es *aplicable*
  - Muy importante → saber justificar la razón de por qué se aplica el patrón
  - Es muy común aplicar patrones en donde el problema no los requiere → el patrón queda forzado, y termina siendo más problemática la solución que el problema

Manejo de Errores

- Cuando estamos desarrollando, la mayoría de las veces llamamos a métodos con los valores equivocados
- TypeScript provee mecanismos específicos para gestionar valores inválidos → *errores*
- Usando bloques *try/catch* podemos capturar errores esperados, para darle un tratamiento específico → permite que nuestro programa se recupere de los errores
- Usando *throw* podemos lanzar un determinado error, en caso de haberlo descubierto
- También se pueden definir errores propios
  - Se definen como clases que *extienden* de Error

Recomendaciones Generales

- Hacer *siempre* un planteo del sistema que vamos a hacer → diagrama de clase, interfaces, etc.
  - Al momento del planteo, pensar en función de cómo se debiera usar una clase *desde afuera* → esto permite ayudarnos a saber qué cosas hacerlas públicas y qué cosas no
- Una clase → una responsabilidad
  - Ejemplo: la clase Auto no puede tener un método que se encargue de calcular el promedio de un arreglo
- Usar nombres descriptivos
- Respetar la organización de una clase
- Tener paciencia con los conceptos, es imposible hacer las cosas bien de entrada → madurez

Curso CFP

CFP Programador full-stack

Ejercicios

Ejercicios

Partiendo de la base de datos armada en las últimas clases

- Generar diagrama de clases usando draw.io
- Escribir un comentario de cada uno de los métodos implementados → el objetivo es ir ejercitando la justificación de las decisiones de diseño tomadas
- Con respecto a las clases, escribir un comentario para cada una, indicando la responsabilidad de cada clase, y la funcionalidad que provee