TP DE PdeP Nº2

EJERCICIO 1

Vamos a analizar **TypeScript** (restringido al **paradigma estructurado**) según el marco que propone **Thomas Kuhn** sobre los cuatro componentes de un paradigma científico. Nos centraremos en los dos primeros: **generalizaciones simbólicas** y **creencias de los profesionales**.

**1. Generalizaciones simbólicas (Reglas escritas del lenguaje)**

En el contexto de Kuhn, las *generalizaciones simbólicas* son los principios formales, las “leyes” que los practicantes aceptan como base del trabajo. En un lenguaje de programación, esto se traduce en sus **reglas sintácticas, semánticas y de uso**.

Para **TypeScript** en el paradigma estructurado, estas generalizaciones incluyen:

* **Estructuras de control secuenciales:** uso de instrucciones ejecutadas en orden (expresiones, asignaciones, llamadas a funciones).
* **Control de flujo mediante selección y repetición:**
  + if, else if, else
  + switch
  + bucles for, while, do...while
* **Funciones como bloques estructurales:**
  + Definición de funciones (function nombre(...) { ... })
  + Paso de parámetros por valor o referencia.
  + Retorno de valores.
* **Tipos estáticos y chequeo en tiempo de compilación:**
  + Declaración de tipos (number, string, boolean, interfaces, tipos personalizados, etc.)
  + Inferencia de tipos.
* **Ámbitos bien definidos:**
  + Variables con alcance de bloque (let, const) y funciones con ámbito local.
* **Organización jerárquica del código:**
  + Dividir el programa en funciones modulares y reutilizables.

Estas reglas promueven un estilo **estructurado**, sin recurrir a técnicas ajenas al paradigma (como programación orientada a objetos o programación funcional avanzada), y garantizan un código más predecible, mantenible y legible.

**2. Creencias de los profesionales (Valores y ventajas percibidas)**

En el sentido kuhniano, las *creencias de los profesionales* son aquellas ideas compartidas por la comunidad sobre **por qué el paradigma (o lenguaje) es valioso o superior** en ciertos aspectos.

En el caso de **TypeScript** dentro de la programación estructurada, los desarrolladores suelen creer que:

* ✅ **Tipado estático y seguridad:** la detección temprana de errores en tiempo de compilación reduce fallos en producción y facilita el mantenimiento del código.
* ✅ **Mejor legibilidad y mantenibilidad:** el uso de funciones, bloques bien definidos y estructuras de control claras ayuda a que el código sea más fácil de entender, probar y extender.
* ✅ **Compatibilidad con JavaScript:** al compilar a JavaScript estándar, TypeScript puede usarse en cualquier entorno web sin perder eficiencia.
* ✅ **Escalabilidad:** la combinación de estructuras claras y tipado fuerte facilita la construcción de proyectos grandes y equipos colaborativos.
* ✅ **Soporte para herramientas y análisis estático:** el ecosistema de TypeScript (VSCode, linters, etc.) ofrece autocompletado, refactorización segura y detección de errores antes de ejecutar el código.
* ✅ **Transición gradual:** es posible comenzar con JavaScript estructurado e ir añadiendo tipado progresivamente sin reescribir todo el proyecto.

En otras palabras, la comunidad ve a TypeScript como un lenguaje que **mantiene la simplicidad estructurada de JavaScript**, pero con las **ventajas del tipado fuerte y un mejor soporte para el desarrollo a gran escala**.

**✅ Resumen**

| **Componente de Kuhn** | **En TypeScript (paradigma estructurado)** |
| --- | --- |
| **Generalizaciones simbólicas** | Reglas de sintaxis, estructuras de control (if, for, etc.), funciones como unidades básicas, tipado estático, ámbitos bien definidos, modularidad. |
| **Creencias de los profesionales** | Seguridad por tipado, legibilidad, mantenibilidad, escalabilidad, compatibilidad con JavaScript, integración con herramientas, transición progresiva. |