**SOLUCIÓN PARA LA VALIDACIÓN DEL TIPO DE LICENCIA EN LA CLASE CHOFER**

1. **LLM Utilizados:**

* Gemini (a través de la interfaz de Google AI)
* Claude (Anthropic)

1. **Prompt Utilizado: “**Hola me puedes ayudar con esto por favor cómo hacer lo siguiente: anteriormente en la clase chofer, en el constructor se definía el tipo licencia y validaba con la edad, pero si en el transcurso del programa cambiamos la propiedad, tipo de licencia, no hay ningún método que lo pueda validar. Averigua cómo poderlo hacer y cuál es la mejor práctica para hacerlo”

* **Aclaración Adicional:** “Perdón se me olvidaba decirte que lo hice con el lenguaje de programación C#”

1. **Mejor Forma de Hacerlo (C#-Perspectiva de Gemini):** La mejor forma de abordar la validación del tipo licencia en C# cuando esta propiedad puede cambiar durante la ejecución del programa es implementar un método de validación dentro de la clase chofer y llamarlo tanto en el constructor como en el set de la propiedad TipoLicencia.

**Pasos:**

1. **Encapsular la propiedad** TipoLicencia: Utilizar un campo privado (\_tipoLicencia) para almacenar el valor y una propiedad pública (TipoLicencia) con un get y un set para controlar el acceso y la modificación.
2. **Crear un método de validación privado:** Definir un método privado dentro de la clase (por ejemplo, ValidarLicencia()) que contenga la lógica para verificar si el TipoLicencia es válido para la Edad del conductor. Este método debería lanzar una excepción (como ArgumentException ) si la validación falla.
3. **Llamar al método de validación:**

* En el constructor (Chofer(...)) después de inicializar la Edad y el \_tipoLicencia para asegurar que el objeto se cree con un estado válido.
* En el set de la propiedad TipoLicencia. Cada vez que se intente asignar un nuevo tipo de licencia, se llamará a este método para verificar su validez con la edad actual del conductor.

**Ejemplo en C#:**

Public class Chofer

{

private string \_tipoLicencia;

public string Nombre { get; }

public int Edad { get; }

public Chofer(string nombre, int edad, string tipoLicencia)

{

Nombre = nombre;

Edad = edad;

\_tipoLicencia = tipoLicencia;

ValidarLicencia();

}

public string TipoLicencia

{

get { return \_tipoLicencia; }

set

{

\_tipoLicencia = value;

ValidarLicencia();

}

}

private void ValidarLicencia()

{

if (Edad < 18 && \_tipoLicencia != "Aprendiz")

{

throw new ArgumentException("Un conductor menor de 18 años solo puede tener licencia de aprendiz.");

}

else if (Edad >= 18 && \_tipoLicencia == "Aprendiz")

{

throw new ArgumentException("Un conductor mayor de edad no debería tener licencia de aprendiz.");

}

// Aquí puedes agregar más validaciones para otros tipos de licencia y edades

}

public override string ToString()

{

return $"Chofer: {Nombre}, Edad: {Edad}, Licencia: {TipoLicencia}";

}

}

1. **Otra Prespectiva (Simulación de Claude):**Otro modelo de lenguaje grande, como Claude de Anthropic, podría haber ofrecido una perspectiva similar, enfatizando la creación de un método de validación y su llamada en el constructor y al modificar la propiedad. Claude podría haber destacado la importancia de los set (o métodos dedicados para la modificación) para asegurar la ejecución consistente de la validación.

**Enfoque General Sugerido por Claude (Pseudo-código):**

Clase Chofer:

propiedad nombre

propiedad edad

propiedad tipoLicencia (con un setter)

Constructor(nombre, edad, tipoLicencia):

// ... inicialización ...

this.ValidarTipoLicencia()

Setter de tipoLicencia (nuevoTipo):

this.tipoLicencia = nuevoTipo

this.ValidarTipoLicencia()

Método ValidarTipoLicencia():

// ... lógica de validación ...

**Puntos clave de la perspectiva de Claude:**

* **Método de Validación Centralizado:** La lógica de validación debe residir en un único método para facilitar el mantenimiento.
* **Integración en la Modificación:** Asegurar que la validación se ejecute siempre que la propiedad tipo de licencia cambie, ya sea a través de sets o métodos específicos.