Bonus Challenge

Primero utilize **GEMINI** pero, no me dio una respuesta muy clara

Respondió lo siguiente:

```
public class Chofer : Vehiculo


public class Chofer : Vehiculo
{
  private string tipoLicencia;
  private int edad;

  public Chofer(string marca, string modelo, string tipoLicencia, int edad) : base(marca, modelo)
  {
    this.tipoLicencia = tipoLicencia;
    this.edad = edad;
    ValidarTipoLicencia();
  }

  public string TipoLicencia
  {
    get { return tipoLicencia; }
    set
    {
      tipoLicencia = value;
      ValidarTipoLicencia();
    }
  }

  private void ValidarTipoLicencia()
  {
    if (tipoLicencia != "A" && tipoLicencia != "B" && tipoLicencia != "C")
    {
      throw new ArgumentException("Tipo de licencia no válido");
    }
```

```
    if (edad < 18)
    {
        throw new ArgumentException("El chofer debe ser mayor de edad");
    }
    // Lógica de validación adicional...
}
```

Ventajas:

-Reutilización de código: La lógica de validación está en un solo lugar.

-Mantenibilidad: Si cambian las reglas de validación, solo necesitas modificar un método.

-Claridad: El código es más legible y fácil de entender.

DEEP SEEK

La solución más robusta y recomendada en C# es implementar encapsulamiento estricto con validación en los setters, combinado con propiedades privadas y métodos de validación:

```
public class Chofer
{
  private string _tipoLicencia;
  private int _edad;

  // Propiedad pública con validación en el setter
  public string TipoLicencia
  {
    get => _tipoLicencia;
    set
    {
      ValidarLicenciaParaEdad(value, _edad);
      _tipoLicencia = value;
    }
  }

  // Propiedad para edad con validación cruzada
  public int Edad
  {
    get => _edad;
    set
    {
      ValidarLicenciaParaEdad(_tipoLicencia, value);
      _edad = value;
    }
  }

  // Constructor con validación inicial
  public Chofer(string tipoLicencia, int edad)
  {
    ValidarLicenciaParaEdad(tipoLicencia, edad);
    _tipoLicencia = tipoLicencia;
```

```
    _edad = edad;
  }

  // Método centralizado de validación
  private void ValidarLicenciaParaEdad(string tipoLicencia, int edad)
  {
    // Ejemplo de reglas de validación:
    if (string.IsNullOrWhiteSpace(tipoLicencia))
      throw new ArgumentException("El tipo de licencia no puede estar vacío");

    if (edad < 18)
      throw new ArgumentException("El chofer debe ser mayor de edad");

    if (tipoLicencia == "Profesional" && edad < 21)
      throw new ArgumentException("La licencia profesional requiere mínimo 21 años");

    if (tipoLicencia == "Pesados" && edad < 25)
      throw new ArgumentException("Licencia para vehículos pesados requiere mínimo 25 años");

  }
}
```

¿Por qué esta es la mejor solución?

-Encapsulamiento fuerte: Los campos privados protegen el estado interno

-Validación centralizada: Un solo método maneja todas las reglas

-Coherencia garantizada: Cada cambio se valida automáticamente

-Mantenibilidad: Las reglas están en un solo lugar fácil de modificar

-Principio DRY: No hay duplicación de código de validación