

Claude

Este es el nuevo código que me dio esta AI

```
using System;
```

```
using System.ComponentModel.DataAnnotations;
```

```
namespace p1bpoo.MisClases
```

```
{
```

```
    public class Chofer : IPiloto
```

```
    {
```

```
        // Enumeración para tipos de licencia
```

```
        public enum TipoLicenciaEnum
```

```
        {
```

```
            [Display(Name = "A", Description = "Licencia Profesional")]
```

```
            A,
```

```
            [Display(Name = "B", Description = "Licencia Semi-Profesional")]
```

```
            B,
```

```
            [Display(Name = "C", Description = "Licencia Básica")]
```

```
            C,
```

```
            [Display(Name = "M", Description = "Licencia Menor")]
```

```
            M
```

```
        }
```

```
        // Clase de validación de edad para licencias
```

```
        private class ValidacionLicencia
```

```
        {
```

```
            public static bool ValidarEdad(TipoLicenciaEnum licencia, int edad)
```

```
            {
```

```
                return licencia switch
```

```

    {
        TipoLicenciaEnum.A => edad >= 23,
        TipoLicenciaEnum.B => edad >= 21,
        TipoLicenciaEnum.C or TipoLicenciaEnum.M => edad >= 18,
        _ => false
    };
}
}

```

// Propiedades con validación

```
public string Nombre { get; }
```

```
private int _edad;
```

```
public int Edad
```

```

{
    get => _edad;
    set
    {
        ValidarCambioEdad(value);
        _edad = value;
    }
}

```

```
private TipoLicenciaEnum _tipoLicencia;
```

```
public TipoLicenciaEnum TipoLicencia
```

```

{
    get => _tipoLicencia;
    set
    {

```

```

        ValidarCambioLicencia(value);

        _tipoLicencia = value;
    }
}

// Constructor con validación
public Chofer(string nombre, int edad, TipoLicenciaEnum tipoLicencia)
{
    // Validación en construcción
    if (!ValidacionLicencia.ValidarEdad(tipoLicencia, edad))
    {
        throw new ArgumentException("La edad no es adecuada para la Licencia
seleccionada", nameof(edad));
    }

    Nombre = nombre;
    _edad = edad;
    _tipoLicencia = tipoLicencia;
}

// Métodos de validación separados
private void ValidarCambioEdad(int nuevaEdad)
{
    if (!ValidacionLicencia.ValidarEdad(_tipoLicencia, nuevaEdad))
    {
        throw new ArgumentException("La nueva edad no es compatible con el tipo de
licencia actual", nameof(nuevaEdad));
    }
}
}

```

```

private void ValidarCambioLicencia(TipoLicenciaEnum nuevaLicencia)
{
    if (!ValidacionLicencia.ValidarEdad(nuevaLicencia, _edad))
    {
        throw new ArgumentException("El tipo de licencia no es compatible con la edad actual", nameof(nuevaLicencia));
    }
}

// Método para mostrar información
public void MostrarInformacion()
{
    Console.WriteLine($"Nombre del piloto: {Nombre}");
    Console.WriteLine($"Edad: {Edad}");
    Console.WriteLine($"Tipo de Licencia: {TipoLicencia}");
}
}
}

```

🔍 **Análisis del Problema Original**

- *En el código inicial, la validación solo se hacía en el constructor*
- *Si se cambiaba directamente la propiedad de licencia o edad, no había validación*
- *Esto podía llevar a estados inconsistentes del objeto Chofer*

🔍 **Objetivos de Mejora**

- *Garantizar que siempre haya validación*
- *Hacer el código más robusto*
- *Prevenir la creación de objetos con combinaciones inválidas de edad y licencia*

🔍 **Decisiones de Diseño a) Reemplazo de String por Enum**

- *Los strings son propensos a errores (mayúsculas, tipesos)*
- *Un enum reduce los posibles valores de entrada*

- *Proporciona un conjunto limitado y controlado de licencias*

b) Validación Centralizada

- *Creé una clase interna ValidacionLicencia*
- *Método estático que contiene toda la lógica de validación*
- *Permite reutilizar la validación en diferentes partes del código*

c) Propiedades con Validación

- *Agregué lógica de validación en los setters*
- *Asegura que no se pueda cambiar edad o licencia sin validar*
- *Lanza excepciones con mensajes descriptivos*

? Mejoras Específicas

- *Uso de switch expression para validación (más moderno y conciso)*
- *Métodos de validación separados para edad y licencia*
- *Uso de nameof() para referencias de parámetros en excepciones*
- *Propiedades de solo lectura donde es posible (como Nombre)*

? Beneficios Adicionales

- *Código más seguro*
- *Más fácil de mantener*
- *Menos propenso a errores*
- *Validación consistente en todos los puntos de cambio*

? Consideraciones de Implementación

- *Mantuve la misma lógica de negocio del código original*
- *Agregué capas de seguridad sin cambiar el comportamiento fundamental*
- *Mejoré la estructura y legibilidad del código*

? Contexto de Buenas Prácticas

- *Sigue principios de programación defensiva*
- *Implementa validación en el punto de entrada*
- *Usa tipos fuertes para reducir errores*
- *Proporciona retroalimentación clara cuando algo va mal*

DeepAI

Este es el nuevo código que me dio esta AI

```
public class LicenciaInvalidaException : Exception
{
    public LicenciaInvalidaException(string message) : base(message) {}
}

public class Chofer : IPiloto
{
    private string tipoLicencia;

    public string Nombre { get; set; }
    public int Edad { get; set; }

    public string TipoLicencia
    {
        get => tipoLicencia;
        set
        {
            if (!EsEdadAdecuada(Edad, value))
            {
                throw new LicenciaInvalidaException("La edad no es adecuada para la Licencia
seleccionada");
            }
            tipoLicencia = value;
        }
    }

    public Chofer(string nombre, int edad, string licencia)
```

```

{
    if (!EsEdadAdecuada(edad, licencia))
    {
        throw new LicenciaInvalidaException("La edad no es adecuada para la Licencia
seleccionada");
    }

    Nombre = nombre;

    Edad = edad;

    TipoLicencia = licencia; // ahora llamará al setter
}

```

```

private bool EsEdadAdecuada(int edad, string licencia)
{
    return (licencia.ToUpper() == "A" && edad >= 23) ||
        (licencia.ToUpper() == "B" && edad >= 21) ||
        (licencia.ToUpper() == "C" && edad >= 18) ||
        (licencia.ToUpper() == "M" && edad >= 18);
}

```

```

public void mostrarInformacion()
{
    Console.WriteLine($"El piloto es: {Nombre}");
    Console.WriteLine($"La edad es: {Edad}");
    Console.WriteLine($"La licencia es: {TipoLicencia}");
}

```

```

public void CambiarLicencia(string nuevaLicencia)
{
    TipoLicencia = nuevaLicencia; // ahora usa el setter que valida
}

```

```
}  
  
}
```

1. Identificación del Problema Inicial

- **Descripción:** En el código original, el constructor validaba la edad del chofer en relación con el tipo de licencia. Sin embargo, no había validación al cambiar la licencia posteriormente, lo que podría llevar a que un Chofer tuviera una licencia no válida después de cambiarla.
- **Impacto:** Esto podría generar inconsistencias en los datos y errores en tiempo de ejecución, quizás dejando al programa en un estado inválido.

2. Creación de Excepción Personalizada

- **Cambio Realizado:** Se creó una clase `LicenciaInvalidaException` que hereda de `Exception`.
- **Razón:** Lanzar una excepción personalizada en lugar de una excepción genérica (`Exception`) ofrece varias ventajas:
 - **Claridad:** Proporciona un contexto más específico sobre el error ocurrido (es decir, que la licencia no es válida).
 - **Manejo de Errores:** Facilita la captura y manejo de excepciones más específicas en otros puntos del código, permitiendo decisiones más informadas sobre cómo proceder.

3. Implementación de Propiedad con Validador

- **Cambio Realizado:** La propiedad `TipoLicencia` fue transformada en una propiedad con un getter y un setter. El setter ahora incluye la lógica de validación que verifica si la edad del chofer es adecuada para el nuevo tipo de licencia.
- **Razón:**
 - **Encapsulamiento:** Al usar una propiedad, se encapsula la lógica de validación directamente en el setter, haciendo que cualquier cambio a la licencia automáticamente se valide sin necesitar explícitamente un método separado para esto.
 - **Facilidad de uso:** Esto simplifica la interfaz de la clase; los consumidores de la clase `Chofer` sólo necesitan establecer la propiedad para realizar la validación, en lugar de llamar a métodos separados.

4. Centralización de la Lógica de Validación

- **Cambio Realizado:** La lógica de la validación de la edad respecto al tipo de licencia se mantiene en el método `EsEdadAdecuada`, que es llamado tanto desde el constructor como desde el setter de `TipoLicencia`.

- **Razón:**
 - **Reutilización de Código:** Mantener la lógica en un solo lugar evita la duplicación y hace que sea más fácil de actualizar si las reglas cambian en el futuro.
 - **Mantenimiento:** Si los criterios de validación cambian (por ejemplo, un nuevo tipo de licencia o nueva edad mínima), solo tendrás que modificar una única función.

5. Simplicidad y Legibilidad

- **Cambio Realizado:** El código resultante es más limpio y más fácil de seguir.
- **Razón:**
 - **Legibilidad:** Los cambios que se hicieron simplifican el flujo de la clase Chofer, el cual es más fácil de leer y entender.
 - **Minimización de Errores:** La simplificación reduce la posibilidad de errores al interpretar la lógica.

Me pareció mas cómodo y mas entendible el código que me dio Claude