

# **Universidad Mariano Gálvez de Guatemala Campus Jutiapa**

Programación I

Ing. Ruldin Ayala

## **VALIDACIÓN DE CLASES PROMPT**



Jhony Abraham de León Pérez

0905-24-22282

El Progreso, Jutiapa 28 de marzo de 2025

### Prompt: Posibles soluciones a futuro en clase Chofer

En el constructor de la clase chofer se define el tipo de licencia y la verificación de la edad del chofer, pero por si en el transcurso de la realización de todo el proyecto cambiamos la propiedad TipoLicencia y no hay ningún método que la pueda validar, Averigua y dime como poder solucionar esto y cuál es la mejor practica para hacerlo. Te proporciono el código de la clase hija Chofer.

**Chatgpt** respondió:

#### Problema Detectado

Actualmente, la validación de la licencia solo se realiza en el constructor y en el método MostrarValidacion(). Sin embargo, si la propiedad TipoLicencia se cambia después de la creación del objeto Chofer, no hay ninguna validación que evite que tenga un valor incorrecto.

---

#### Solución: Usar una Propiedad con un Setter Validado

En lugar de permitir que TipoLicencia sea modificada sin validación, podemos convertirla en una **propiedad con un setter validado**. Así, cada vez que se asigne un nuevo valor, se validará automáticamente.

#### Código Mejorado para Chofer:

```
using p1bpoo.Interfaces;
using System;

namespace p1bpoo.MisClases
{
    public class Chofer : Ipiloto
    {
        private string tipoLicencia;

        public string TipoLicencia
        {
            get => tipoLicencia;
            set
            {
                if (EsLicenciaValida(value))
                {
                    tipoLicencia =
                        value.ToUpper();
                }
                else
                {
                    throw new
                        ArgumentException($"La licencia '{value}'

```

no es válida o la edad no cumple con los requisitos.");

```
    }  
  }  
}
```

public Chofer(string name, int edadPiloto, string tipo)

```
{  
    Nombre = name;  
    Edad = edadPiloto;  
    TipoLicencia = tipo; // Se validará automáticamente con el setter  
}
```

private bool EsLicenciaValida(string licencia)

```
{  
    switch (licencia.ToUpper())  
    {  
        case "A":  
            return Edad >= 23;  
        case "B":  
            return Edad >= 21;  
        case "C":  
            return Edad >= 16;
```

case "M":

return Edad >= 16;

default:

return false;

```
}
```

```
}
```

public void MostrarValidacion()

```
{
```

Console.WriteLine(\$"{Nombre} tiene la edad necesaria para obtener la licencia tipo {TipoLicencia} y conducir este vehículo.");

```
}
```

public void MostrarInformación()

```
{
```

Console.WriteLine("Nombre: {0}", Nombre);

Console.WriteLine("Edad: {0}", Edad);

Console.WriteLine("Licencia Tipo: {0}", TipoLicencia);

```
}
```

```
}
```

```
}
```

---

## ¿Cómo Funciona la Solución?

### 1. Uso de una variable privada tipoLicencia:

- Se usa una variable privada para almacenar el valor real de la licencia.

## 2. Setter validado en TipoLicencia:

- Cada vez que se asigne un nuevo valor a TipoLicencia, el setter llamará al método EsLicenciaValida() para verificar que la edad del chofer cumpla con los requisitos.
- Si la licencia no es válida, se lanza una excepción ArgumentException.

## 3. Validación automática al cambiar la licencia:

- Si se intenta asignar una licencia no válida, el código fallará de inmediato, evitando estados inconsistentes en el objeto.

---

### Ventajas de esta Solución

- ✓ **Protege la integridad de los datos:** No permite valores no válidos para TipoLicencia.
- ✓ **Evita errores futuros:** Se garantiza que siempre que la licencia cambie, la validación se ejecute.
- ✓ **Más seguro y fácil de mantener:** No se necesita llamar a ValidarLicencia() manualmente.

---

### Ejemplo de Uso y Pruebas:

```
try
```

```
{
```

```
    Chofer chofer1 = new Chofer("Carlos", 22, "A");
```

```
    chofer1.MostrarInformación();
```

```
    chofer1.TipoLicencia = "B"; // Esto funciona porque tiene 22 años y B requiere 21.
```

```
    chofer1.MostrarInformación();
```

```
    chofer1.TipoLicencia = "C"; // Esto genera un error porque C requiere 16, y él tiene 22 (pero es válido).
```

```
    chofer1.MostrarInformación();
```

```
chofer1.TipoLicencia = "Z"; // Esto generará una excepción porque "Z" no es una licencia válida.
```

```
}
```

```
catch (ArgumentException ex)
```

```
{
```

```
    Console.WriteLine("Error: " + ex.Message);
```

```
}
```

### Salida esperada:

yaml

CopiarEditar

Nombre: Carlos

Edad: 22

Licencia Tipo: A

Nombre: Carlos

Edad: 22

Licencia Tipo: B

Nombre: Carlos

Edad: 22

Licencia Tipo: C

Error: La licencia 'Z' no es válida o la edad no cumple con los requisitos.

---

Con esto, la propiedad TipoLicencia siempre será válida, sin importar cuándo se modifique.



**MICROSOFT COPILOT:** Respondió al mismo Prompt:

¡Gracias por compartir el código! Una excelente forma de resolver la posible inconsistencia con la validación de TipoLicencia sería introducir un setter (mutador) con lógica de validación. Esto asegura que cada vez que se intente cambiar el valor de TipoLicencia, pase por un filtro que compruebe su validez.

**Aquí está una propuesta para modificar tu clase Chofer con esta práctica:**

**Código actualizado con validación en el setter**

```
using p1bpoo.Interfaces;                                _tipoLicencia = value;
using System;                                           }
                                                       else
namespace p1bpoo.MisClases                             {
{                                                       throw new
public class Chofer : Ipiloto                          ArgumentException($"El tipo de licencia
{                                                       '{value}' no es válido. Los tipos
public string Nombre { get; set; }                     permitidos son: A, B, C, M.");
public int Edad { get; set; }                           }
                                                       }

private string _tipoLicencia; //
Campo privado para encapsular la                      // Constructor
propiedad
public string TipoLicencia
{
    get { return _tipoLicencia; }
    set
    {
        // Validación al asignar el valor
        if (EsTipoLicenciaValido(value))
        {
            // Método de validación para licencia
```

```
private bool
EsTipoLicenciaValido(string tipo)
{
    string[] tiposValidos = { "A", "B",
"C", "M" };

    return Array.Exists(tiposValidos, t
=> t.Equals(tipo,
StringComparison.OrdinalIgnoreCase));
}

// Método para validar la licencia
basado en la edad

public bool ValidarLicencia()
{
    switch (_tipoLicencia.ToUpper())
    {
        case "A":
            return Edad >= 23;

        case "B":
            return Edad >= 21;

        case "C":
        case "M":
            return Edad >= 16;

        default:
            return false;
    }
}
```

```
public void MostrarValidacion()
{
    if (ValidarLicencia())
    {
        Console.WriteLine($"{Nombre}
tiene la edad necesaria para obtener la
licencia tipo {_tipoLicencia} y conducir
este vehículo.");
    }
    else
    {
        Console.WriteLine($"{Nombre}
lamentablemente no tiene la edad
necesaria para obtener la licencia
{_tipoLicencia}.");
    }
}

public void MostrarInformación()
{
    Console.WriteLine("Nombre: {0}",
this.Nombre);

    Console.WriteLine("Edad: {0}",
this.Edad);

    Console.WriteLine("Licencia Tipo:
{0}", this._tipoLicencia);
}
}
```

**Cambios realizados:**

1. **Campo privado `_tipoLicencia`:** Se usa para encapsular la propiedad `TipoLicencia`.
2. **Validación en el setter de `TipoLicencia`:** La asignación ahora pasa por una validación que garantiza que el tipo de licencia sea válido.
3. **Método `EsTipoLicenciaValido`:** Define la lógica de validación de tipos permitidos (A, B, C, M).
4. **Excepción informativa:** Si el tipo de licencia no es válido, se lanza un error con un mensaje claro.

#### **Ventajas:**

- **Protección ante valores no válidos:** La validación es centralizada y automática al cambiar `TipoLicencia`.
- **Simplicidad en el uso:** El código cliente puede asignar valores sin preocuparse por la validación, ya que esta ocurre internamente.
- **Mantenibilidad:** Si en el futuro los tipos de licencia cambian, solo necesitas actualizar el método `EsTipoLicenciaValido`.

Este enfoque hace que tu clase sea más robusta y coherente. ¿Qué opinas? ¿Te gustaría discutir alguna parte con más detalle o hacer más mejoras? 😊

#### **Comentario personal:**

A mi parecer esta más fácil comprender el que chatgpt me brindo ya que no modifíco mucho la sintaxis con la que trabajé originalmente, pero analíticamente lo dejo a su consideración.