

Universidad Mariano Gálvez De Guatemala Ingeniería
En Sistemas De Información Y Ciencias De La
Computación
Campus Jutiapa

Programación I

Ing.

Ruldin Ayala



Estudiante:

José Mario Rosales Palma

Carnet:

0905-24-17488

2025

Primer LLM: ChatGPT

Mi Prompt fue: Ayúdame a solucionar un problema con un código que te voy a proporcionar, le tire el código de la clase chofer y le puse en el constructor de la clase chofer está definido el tipo de licencia y la verificación de edad, pero si en el transcurso del programa cambiamos la propiedad, TipoLicencia y no hay ningún método que lo pueda validar, dime cómo poder solucionar esto y cuál es la mejor práctica para hacerlo.

ChatGPT me dijo:

El problema que mencionas ocurre porque TipoLicencia es una propiedad pública sin validación, lo que permite que su valor sea cambiado después de la creación del objeto sin ninguna restricción.

Solución: Usar una propiedad con validación en el setter

Podemos modificar la propiedad TipoLicencia para que incluya validación dentro de su setter. De esta manera, cada vez que se asigne un nuevo valor, se verificará automáticamente si es válido. Aquí te muestro cómo hacerlo:

```
private string _tipoLicencia;
```

```
public string TipoLicencia
```

```
{
```

```
    get => _tipoLicencia;
```

```
    set
```

```
    {
```

```
        if (!EsLicenciaValida(value))
```

```
        {
```

```
        throw new ArgumentException($"El tipo de licencia '{value}' no  
        es válido.");  
    }  
    _tipoLicencia = value.ToUpper();  
}  
}
```

Y agregamos un método auxiliar para validar:

```
private bool EsLicenciaValida(string tipo)  
{  
    return tipo.ToUpper() == "A" || tipo.ToUpper() == "B" || tipo.ToUpper()  
    == "C" || tipo.ToUpper() == "D";  
}
```

Beneficios de esta solución

1. Evita asignaciones incorrectas: Si alguien intenta asignar una licencia inválida, se lanzará una excepción.
2. Asegura consistencia: No habrá valores incorrectos en el objeto Chofer.
3. Facilita el mantenimiento: Si se agregan nuevos tipos de licencia en el futuro, solo se modifica el método EsLicenciaValida.

Con esta solución, cada vez que intentes cambiar TipoLicencia, se validará automáticamente, evitando problemas en el resto del código.

Segundo LLM: Microsoft Copilot: Le puse el mismo Prompt

Microsoft Copilot me dijo:

¡Excelente pregunta! El problema que señalas surge porque se permite modificar la propiedad TipoLicencia fuera del constructor sin validar si ese cambio es correcto. Para solucionar este problema y mantener las mejores prácticas, puedes implementar una validación en el setter de la propiedad TipoLicencia. De esta forma, cada vez que se modifique esa propiedad, se llevará a cabo una verificación automática. Aquí te muestro cómo hacerlo:

```
public class Chofer : Ipiloto
{
    private string tipoLicencia;
    public string TipoLicencia
    {
        get { return tipoLicencia; }
        set
        {
            if (ValidarLicencia(value))
            {
                tipoLicencia = value;
            }
            else
            {
                throw new ArgumentException($"El tipo de licencia '{value}'
no es válido o no cumple con los requisitos de edad.");
            }
        }
    }
}
```

```
    }  
  }  
}
```

// Constructor

```
public Chofer(string name, int edadPiloto, string tipo)  
{  
    Nombre = name;  
    Edad = edadPiloto;  
    TipoLicencia = tipo; // Esto también pasará por la validación del  
setter  
}
```

// Método para validar licencia

```
private bool ValidarLicencia(string tipoLicencia)  
{  
    switch(tipoLicencia.ToUpper())  
    {  
        case "A":  
            return Edad >= 23;  
        case "B":  
            return Edad >= 21;  
        case "C":  
            return Edad >= 16;  
        case "D":
```

```
        return Edad >= 16;
    default:
        return false; // Si el tipo de licencia no es válido
    }
}
}
```

¿Por qué es una mejor práctica?

Encapsulación: El setter asegura que cualquier modificación de la propiedad TipoLicencia pase por una validación, evitando errores.

Consistencia: Mantiene la lógica de validación centralizada dentro de la clase, reduciendo el riesgo de inconsistencias.

Manejo de errores: Utiliza una excepción (ArgumentException) para informar al usuario o desarrollador cuando se intenta asignar un valor inválido.

Alternativa: Método de validación explícito

Otra opción sería crear un método dedicado para validar cambios en TipoLicencia en lugar de usar el setter, pero esto requiere que el desarrollador llame manualmente a dicho método, lo que puede ser propenso a errores si se olvida hacerlo.