Instituto CEI

Carrera: Analista Programador
Materia: Programación 2
Informe Trabajo Practico 1

Federico Goldaracena

Resumen

El programa es una simulación de un Parlamento que permite a los usuarios realizar diversas operaciones relacionadas con legisladores y sus actividades en una legislatura. Los legisladores representan a diferentes partidos políticos y departamentos, y tienen la capacidad de presentar propuestas legislativas, registrar votos, participar en debates y más. El Parlamento gestiona a estos legisladores y proporciona funcionalidades como contar votos y listar legisladores y propuestas legislativas.

Utiliza ampliamente conceptos de programación orientada a objetos, incluyendo herencia y polimorfismo, para modelar y gestionar los legisladores y sus actividades en el Parlamento de manera eficiente y estructurada.

Índice

1. Introducción

1.1. Lenguaje C#

C# (pronunciado "C sharp") es un lenguaje de programación de propósito general desarrollado por Microsoft. Fue creado a principios de la década de 2000 y se ha convertido en uno de los lenguajes de programación más populares y ampliamente utilizados en el mundo del desarrollo de software. C# Es conocido por su sintaxis limpia y estructurada, lo que lo hace relativamente fácil de aprender y leer. Este lenguaje se utiliza principalmente para desarrollar aplicaciones de escritorio, aplicaciones web y aplicaciones móviles en el entorno de desarrollo de Microsoft, incluyendo el uso de la plataforma .NET. Una de las características distintivas de C# Es su capacidad de programación orientada a objetos, que permite a los desarrolladores crear software modular y escalable. También ofrece soporte para la programación asincrónica, lo que facilita la creación de aplicaciones que pueden manejar múltiples tareas de manera eficiente.

Microsoft pone al alcance, de toda la comunidad planetaria de programadores, sus plataformas de desarrollo, como Visual Studio Code, de licencia gratuita (Freeware: no pagas por utilizarla), multiplataforma (para Windows, Linux y Mac OS), bajo el entorno .NET Core; y también Visual Studio (de pago), en sus versiones de 2017/2019, para Windows y Mac OS, bajo el entorno de .NET Framework. La diferencia entre ambos entornos es el destinatario final de las aplicaciones, siendo el último solo para Windows, mientras que el primero para las 3 plataformas mencionadas.

Para trabajar con C# es recomendable utilizar Microsoft Visual Studio, ya que ha sido desarrollado especialmente para ello y soporta la carga y trabajo con su mismo lenguaje.

Por ello, nos descargamos Microsoft Visual Studio y ejecutamos el programa. Puede ocurrir que te pida la instalación de algunas librerías esenciales para un correcto funcionamiento del programa, que nos la podemos descargar gratuitamente desde Microsoft.

Una vez ejecutado el programa, puedes crear un documento de tipo C# y ya puedes escribir en este código.[1]

1. POO

1.2. Programación Orientado a Objetos (POO)

La Programación Orientada a Objetos (POO) es un enfoque de programación que se fundamenta en la utilización de objetos para modelar entidades del mundo real y llevar a cabo tareas complejas. La POO se encuentra presente en numerosos lenguajes de programación populares, tales como Java, Python, C++, y muchos más. En este artículo, examinaremos los cuatro pilares fundamentales de la POO: el encapsulamiento, la herencia, el polimorfismo y la abstracción.[2]

1. POO

1.3. Herencia

La herencia es una de las tres características principales de la programación orientada a objetos, junto con la encapsulación y el polimorfismo. Permite crear clases que reutilizan y extienden el comportamiento de otras clases. En este proceso, una clase base proporciona la estructura inicial, mientras que una clase derivada hereda sus miembros. La clase derivada puede especializarse aún más, representando diferentes aspectos de la clase base. Por ejemplo, una clase base "Animal" puede tener clases derivadas como "Mammal" y "Reptile", que son especializaciones de "Animal". Esto facilita la organización y la reutilización del código en la programación orientada a objetos. [3]

2. Herencia

1.4. Polimorfismo

La palabra "polimorfismo" proviene del griego y significa "varias formas diferentes". En programación orientada a objetos, esta idea es fundamental. Al igual que la herencia se relaciona con las clases y su jerarquía, el polimorfismo se relaciona con los métodos. Existen tres tipos de polimorfismo: Polimorfismo de sobrecarga: Esto ocurre cuando funciones con el mismo nombre y funcionalidad similar existen en clases completamente independientes. Por ejemplo, clases como "complex", "image" y "link" pueden tener una función llamada "display". Esto significa que no es necesario preocuparse por el tipo de objeto cuando se quiere mostrar en pantalla. Polimorfismo paramétrico: Aquí, se pueden definir varias funciones con el mismo nombre, pero diferentes parámetros. El método correcto se selecciona automáticamente según el tipo de datos pasados como parámetros. Por ejemplo, se pueden tener métodos "addition" que sumen enteros, flotantes, caracteres, etc. Polimorfismo de subtipado: Este tipo de polimorfismo permite redefinir un método en clases derivadas de una clase base. Esto se llama especialización y permite llamar a un método de objeto sin necesidad de conocer su tipo intrínseco. Por ejemplo, en un juego de ajedrez, el método "movimiento" podría realizar el movimiento correspondiente según el tipo de pieza que se llama, sin preocuparse por los detalles de cada pieza en particular. El polimorfismo en programación orientada a objetos permite utilizar métodos de manera flexible, adaptándolos a diferentes situaciones y tipos de datos, lo que simplifica el diseño y la reutilización de código. [4]

3. Polimorfismo

1.5. UML

Un diagrama UML es una herramienta visual utilizada por ingenieros de software para comprender sistemas y software complejos. Ayuda a simplificar la comprensión de diseños, arquitectura de código y flujos de trabajo. Se basa en el Lenguaje Unificado de Modelado (UML) y se utiliza tanto en la programación de software como en la modelización de procesos empresariales. En lugar de enfrentarse a miles de líneas de código, un diagrama UML ofrece una representación visual más clara y fácil de

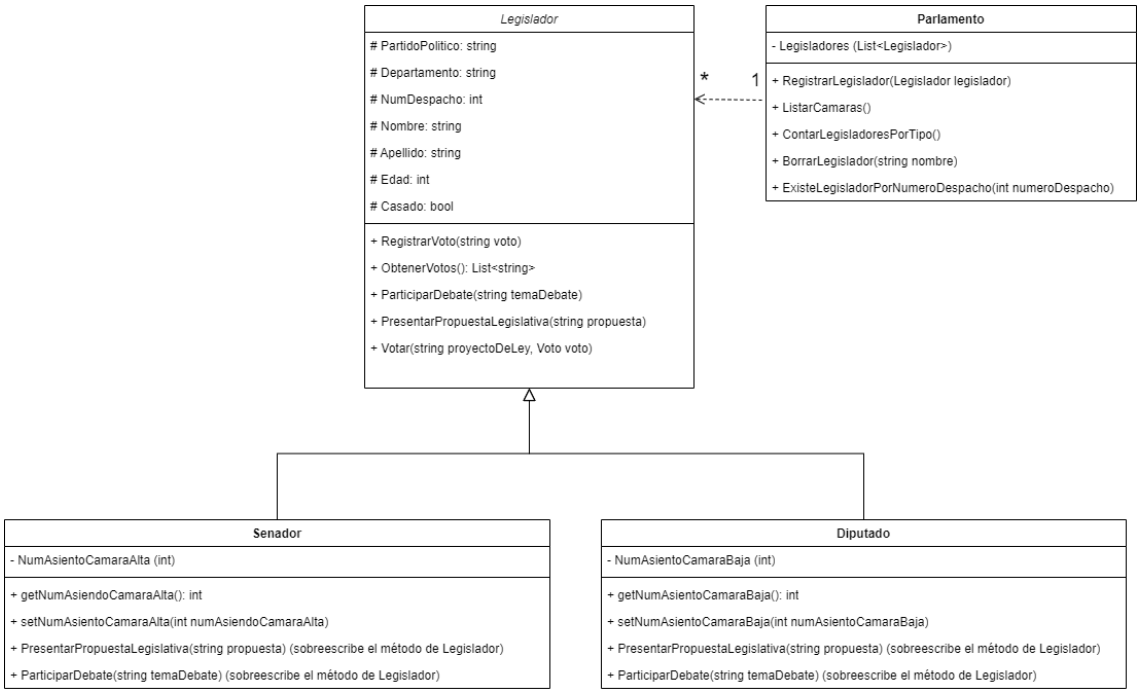
comprender. Facilita el seguimiento de relaciones y jerarquías importantes entre las partes del código. Aunque se asemejan a los diagramas de flujo, los diagramas UML tienen características específicas. Estos diagramas simplifican la complejidad del código, lo que beneficia tanto a los ingenieros como a las partes interesadas no técnicas. Permiten mantenerse al tanto de los proyectos y evitan perderse en las complejidades inherentes a la programación. Además, ayudan a descomponer los componentes esenciales de un programa informático en partes más manejables. [5]

4. UML

2. Metodología de estudio

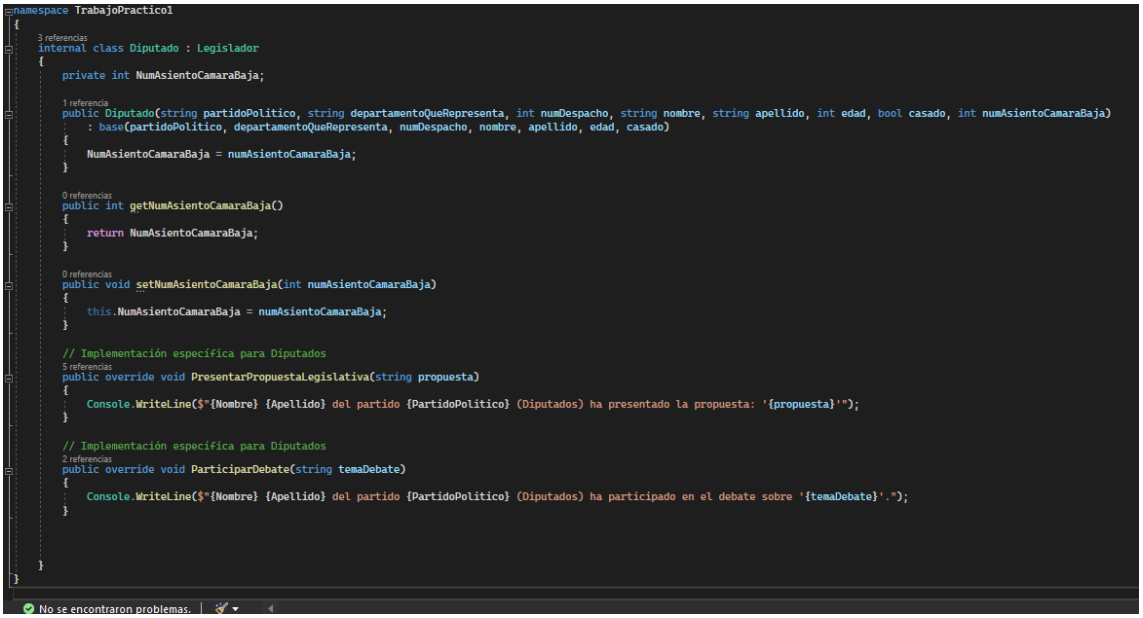
2.1. UML Trabajo Practico 1

El diagrama UML representa un sistema de legislatura en C# con tres clases principales: Legislador, Senador y Diputado. Legislador es la clase base y tiene atributos como PartidoPolítico, Departamento, Nombre, etc. Puede registrar votos, presentar propuestas, votar en proyectos y participar en debates legislativos. Senador hereda de Legislador y agrega un atributo NumAsientoCamaraAlta. Sobrescribe los métodos de presentar propuestas y participar en debates. Diputado también hereda de Legislador y tiene un atributo adicional, NumAsientoCamaraBaja. Al igual que Senador, sobrescribe los métodos de presentar propuestas y participar en debates. Parlamento es una clase que maneja una colección de Legisladores. Permite registrar legisladores, conectar senadores y diputados, listar las cámaras y verificar legisladores por número de despacho. Este diagrama UML modela la estructura y relaciones entre legisladores y su gestión en el contexto parlamentario.



2.2. Clase Diputado

Clase DiputadoNamespace (namespace TrabajoPractico1): Un namespace en C# es una forma de organizar el código en un contexto específico. En este caso, el código está organizado dentro del namespace TrabajoPractico1. Clase Diputado (internal class Diputado: Legislador): Esta clase representa a un diputado y hereda de la clase Legislador. Los diputados son legisladores específicos de una cámara baja en un sistema parlamentario. Atributos: private int NumAsientoCamaraBaja;: Es un atributo privado que almacena el número de asiento del diputado en la cámara baja. Constructor (public Diputado(...)): Es un constructor que inicializa un objeto Diputado con los parámetros proporcionados. Este constructor toma el partido político, departamento que representa, número de despacho, nombre, apellido, edad, estado civil, y número de asiento en la cámara baja como argumentos y luego inicializa los atributos correspondientes de la clase base (Legislador) y el atributo NumAsientoCamaraBaja de la clase Diputado. Métodos: public int getNumAsientoCamaraBaja(): Un método público que devuelve el número de asiento en la cámara baja del diputado. public void setNumAsientoCamaraBaja(int numAsientoCamaraBaja): Un método público que permite establecer el número de asiento en la cámara baja del diputado. public override void PresentarPropuestaLegislativa(string propuesta): Un método público que sobrescribe el método PresentarPropuestaLegislativa de la clase base. Muestra un mensaje indicando que el diputado ha presentado una propuesta legislativa específica. public override void ParticiparDebate(string temaDebate): Un método público que sobrescribe el método ParticiparDebate de la clase base. Muestra un mensaje indicando que el diputado ha participado en un debate sobre un tema específico. Sobreescritura de Métodos: La clase Diputado sobrescribe los métodos PresentarPropuestaLegislativa y ParticiparDebate de la clase base Legislador. Esto significa que cuando un objeto de tipo Diputado llama a estos métodos, se utilizarán las implementaciones específicas proporcionadas en la clase Diputado en lugar de las implementaciones en la clase base.



```
namespace TrabajoPractico1
{
    3 referencias
    internal class Diputado : Legislador
    {
        private int NumAsientoCamaraBaja;

        1 referencia
        public Diputado(string partidoPolitico, string departamentoQueRepresenta, int numDespacho, string nombre, string apellido, int edad, bool casado, int numAsientoCamaraBaja)
            : base(partidoPolitico, departamentoQueRepresenta, numDespacho, nombre, apellido, edad, casado)
        {
            NumAsientoCamaraBaja = numAsientoCamaraBaja;
        }

        0 referencias
        public int getNumAsientoCamaraBaja()
        {
            return NumAsientoCamaraBaja;
        }

        0 referencias
        public void setNumAsientoCamaraBaja(int numAsientoCamaraBaja)
        {
            this.NumAsientoCamaraBaja = numAsientoCamaraBaja;
        }

        // Implementación específica para Diputados
        5 referencias
        public override void PresentarPropuestaLegislativa(string propuesta)
        {
            Console.WriteLine($"{Nombre} {Apellido} del partido {PartidoPolitico} (Diputados) ha presentado la propuesta: '{propuesta}'");
        }

        // Implementación específica para Diputados
        2 referencias
        public override void ParticiparDebate(string temaDebate)
        {
            Console.WriteLine($"{Nombre} {Apellido} del partido {PartidoPolitico} (Diputados) ha participado en el debate sobre '{temaDebate}'.");
        }



    }
}
```

No se encontraron problemas.

2.2.1 Clase Diputado con su atributo, herencia y los métodos con polimorfismo.

2.3. Clase Senador

Clase Senador (internal class Senador: Legislador): Esta clase representa a un senador y hereda de la clase Legislador. Atributos: private int NumAsientoCamaraAlta;: Es un atributo privado que almacena el número de asiento del senador en la cámara alta. Constructor (public Senador(...)): Es un constructor que inicializa un objeto Senador con los mismos parámetros que la clase Diputado. Toma el partido político, departamento que representa, número de despacho, nombre, apellido, edad, estado civil y número de asiento en la cámara alta como argumentos y luego inicializa los atributos correspondientes de la clase base (Legislador) y el atributo NumAsientoCamaraAlta de la clase Senador. Métodos: public int getNumAsiendoCamaraAlta(): Un método público que devuelve el número de asiento en la cámara alta del senador. public void setNumAsientoCamaraAlta(int numAsiendoCamaraAlta): Un método público que permite establecer el número de asiento en la cámara alta del senador. public override void PresentarPropuestaLegislativa(string propuesta): Un método público que sobrescribe el método PresentarPropuestaLegislativa de la clase base. Muestra un mensaje indicando que el senador ha presentado una propuesta legislativa específica. public override void ParticiparDebate(string temaDebate): Un método público que sobrescribe el método ParticiparDebate de la clase base. Muestra un mensaje indicando que el senador ha participado en un debate sobre un tema específico. Similar a la clase Diputado, la clase Senador también proporciona implementaciones específicas para los métodos PresentarPropuestaLegislativa y ParticiparDebate, lo que permite personalizar el comportamiento de estos métodos para los senadores en particular.

```
namespace TrabajoPractico1
{
    1 referencia
    internal class Senador : Legislador
    {
        private int NumAsientoCamaraAlta;

        0 referencias
        public Senador(string partidoPolitico, string departamentoQueRepresenta, int numDespacho, string nombre, string apellido, int edad, bool casado, int numAsientoCamaraAlta)
            : base(partidoPolitico, departamentoQueRepresenta, numDespacho, nombre, apellido, edad, casado)
        {
            NumAsientoCamaraAlta = numAsientoCamaraAlta;
        }

        0 referencias
        public int getNumAsiendoCamaraAlta()
        {
            return NumAsientoCamaraAlta;
        }

        0 referencias
        public void setNumAsientoCamaraAlta(int numAsiendoCamaraAlta)
        {
            this.NumAsientoCamaraAlta = numAsiendoCamaraAlta;
        }

        // Implementación específica para Senadores
        5 referencias
        public override void PresentarPropuestaLegislativa(string propuesta)
        {
            Console.WriteLine($"{Nombre} {Apellido} del partido {PartidoPolitico} (Senado) ha presentado la propuesta: '{propuesta}'");
        }

        // Implementación específica para Senadores
        2 referencias
        public override void ParticiparDebate(string temaDebate)
        {
            Console.WriteLine($"{Nombre} {Apellido} del partido {PartidoPolitico} (Senado) ha participado en el debate sobre '{temaDebate}'.");
        }
    }
}
```

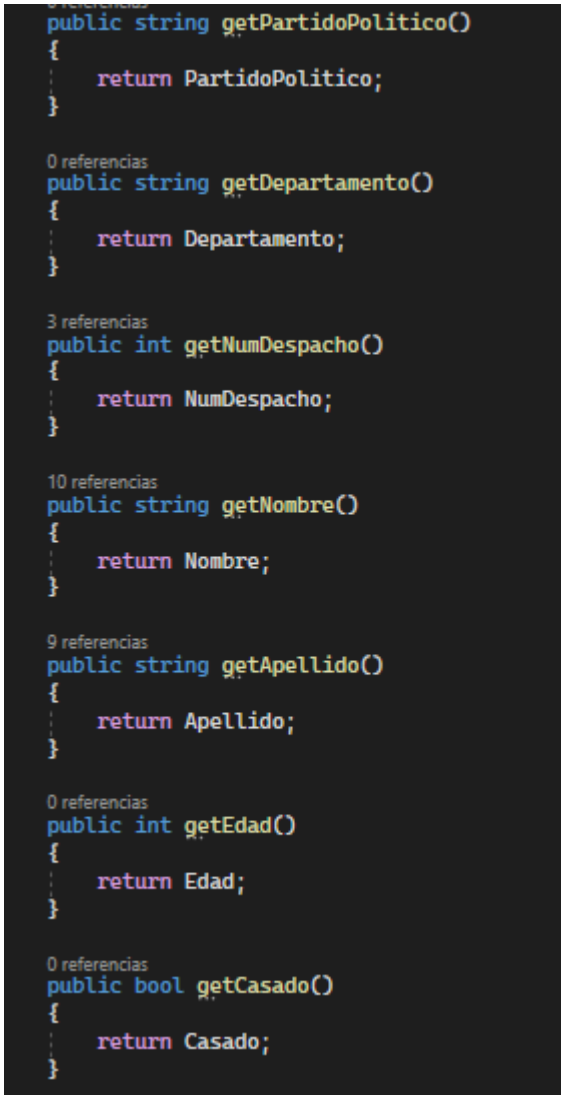
2.2.2 Clase Senador con su atributo, herencia y los métodos con polimorfismo.

2.4. Clase Legislador

La clase representa a un legislador genérico en un sistema parlamentario. Aquí está una explicación detallada del código: Enum Voto: Define un enumerador Voto con tres valores posibles: Aprobado, Rechazado y Abstencion. Estos valores representan los posibles resultados de un voto en el parlamento. Clase Legislador (internal class Legislador): Esta clase representa a un legislador y contiene varios atributos y métodos relacionados con los legisladores en general. Atributos: protected string PartidoPolitico;: Almacena el nombre del partido político al que pertenece el legislador. protected string Departamento;: Representa el departamento que el legislador representa. protected int NumDespacho;: Es el número de despacho del legislador. protected string Nombre;: Almacena el nombre del legislador. protected string Apellido;: Almacena el apellido del legislador. protected int Edad;: Almacena la edad del legislador. protected bool Casado;: Indica si el legislador está casado o no. private List<string> votos = new List<string>();: Es una lista privada que se utiliza para rastrear los votos emitidos por el legislador. public List<string> PropuestasLegislativas { get; private set; }: Es una lista pública que almacena las propuestas legislativas presentadas por el legislador. Constructor (public Legislador(...)): Es el constructor de la clase. Inicializa los atributos del legislador con los valores proporcionados y crea una nueva lista para PropuestasLegislativas. Métodos: Métodos get y set para acceder y modificar los atributos del legislador, como getPartidoPolitico(), setPartidoPolitico(string partidoPolitico), getEdad(), setEdad(int edad), y así sucesivamente. public void RegistrarVoto(string voto): Un método para registrar un voto en la lista de votos del legislador. public List<string> ObtenerVotos(): Un método para obtener la lista de votos del legislador. public void MostrarVotos(): Un método para mostrar la lista de votos del legislador en la consola. public string getCamara(): Un método que devuelve "Senador" si el legislador tiene más de 30 años (supuestamente, la edad para ser senador) y "Diputados" en caso contrario. public virtual void PresentarPropuestaLegislativa(string propuesta): Un método virtual que muestra un mensaje indicando que el legislador ha presentado una propuesta legislativa y agrega la propuesta a la lista de propuestas. public virtual void Votar(string proyectoDeLey, Voto voto): Un método virtual que muestra un mensaje indicando el voto del legislador en un proyecto de ley específico. public virtual void ParticiparDebate(string temaDebate): Un método virtual que muestra un mensaje indicando que el legislador ha participado en un debate sobre un tema específico. Este código proporciona una estructura básica para representar a los legisladores en un sistema parlamentario, permitiendo registrar votos, presentar propuestas legislativas, participar en debates y realizar otras acciones relacionadas con su papel en el parlamento.

```
internal class Legislador
{
    protected string PartidoPolitico;
    protected string Departamento;
    protected int NumDespacho;
    protected string Nombre;
    protected string Apellido;
    protected int Edad;
    protected bool Casado;


    // Lista de votos del legislador

    3 referencias
    public List<string> PropuestasLegislativas { get; private set; }

    private List<string> votos = new List<string>(); // Agregar una lista para rastrear los votos


    // Constructor
    4 referencias
    public Legislador(string partidoPolitico, string departamentoQueRepresenta, int numDespacho, string nombre, string apellido, int edad, bool casado)
    {
        PartidoPolitico = partidoPolitico;
        Departamento = departamentoQueRepresenta;
        NumDespacho = numDespacho;
        Nombre = nombre;
        Apellido = apellido;
        Edad = edad;
        Casado = casado;
        PropuestasLegislativas = new List<string>();
    }
```

2.2.3 Clase Legislador con sus atributos y constructor

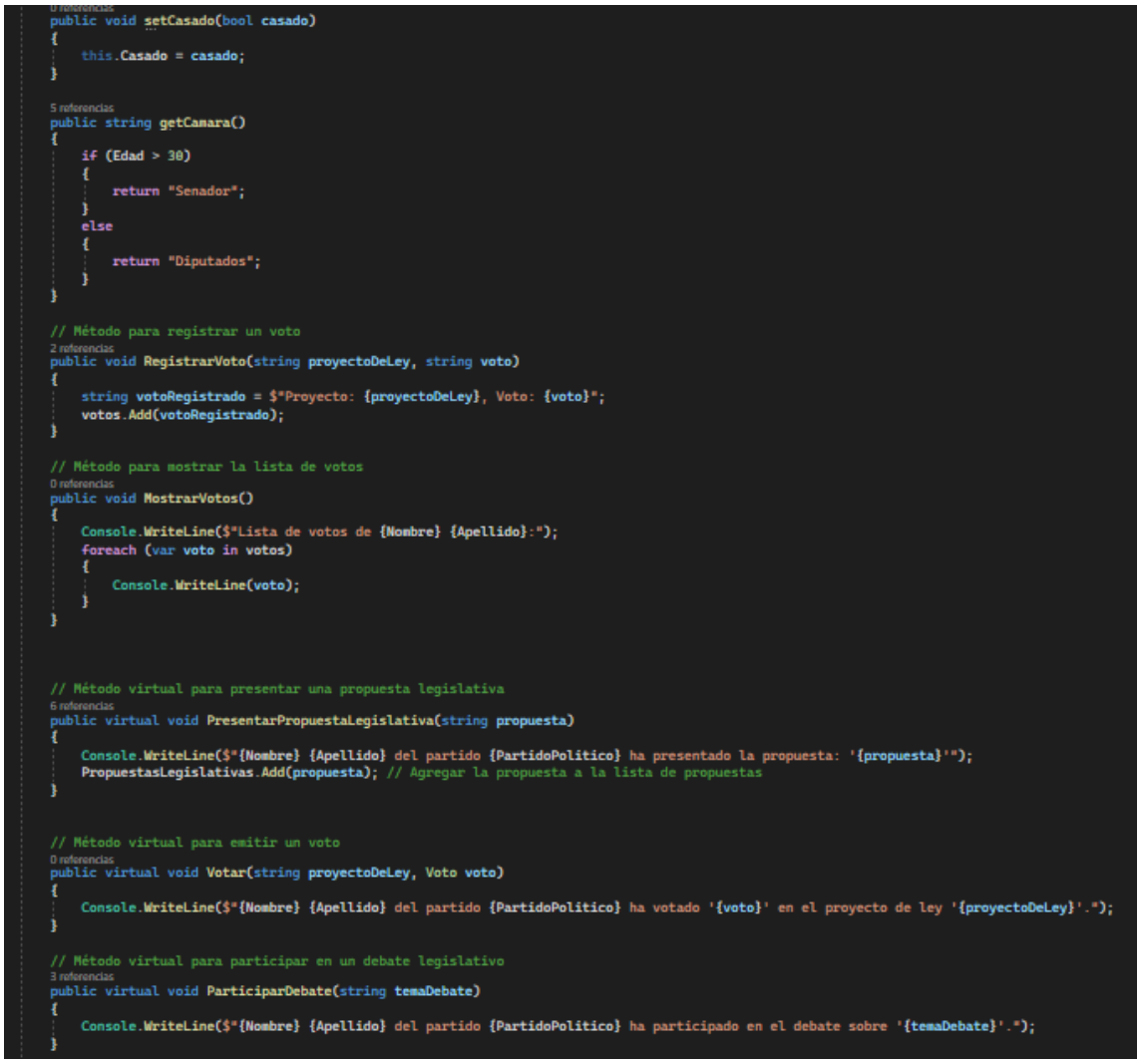


```
public string getPartidoPolitico()
{
    return PartidoPolitico;
}

0 referencias
public string getDepartamento()
{
    return Departamento;
}

3 referencias
public int getNumDespacho()
{
    return NumDespacho;
}

10 referencias
public string getNombre()
{
    return Nombre;
}

9 referencias
public string getApellido()
{
    return Apellido;
}

0 referencias
public int getEdad()
{
    return Edad;
}

0 referencias
public bool getCasado()
{
    return Casado;
}
```

2.2.4 Getters

```
// Método para registrar un voto
1 referencia
public void RegistrarVoto(string voto)
{
    votos.Add(voto);
}

// Método para obtener la lista de votos
1 referencia
public List<string> ObtenerVotos()
{
    return votos;
}

0 referencias
public void setPartidoPolitico(string partidoPolitico)
{
    this.PartidoPolitico = partidoPolitico;
}

0 referencias
public void setDepartamento(string departamentoQueRepresenta)
{
    this.Departamento = departamentoQueRepresenta;
}

1 referencia
public void setNumDespacho(int numDespacho)
{
    this.NumDespacho = numDespacho;
}

0 referencias
public void setNombre(string nombre)
{
    this.Nombre = nombre;
}

0 referencias
public void setApellido(string apellido)
{
    this.Apellido = apellido;
}

0 referencias
public void setEdad(int edad)
{
    this.Edad = edad;
}
```

2.2.5 Setters y Metodos



```
public void setCasado(bool casado)
{
    this.Casado = casado;
}

5 referencias
public string getCamara()
{
    if (Edad > 30)
    {
        return "Senador";
    }
    else
    {
        return "Diputados";
    }
}

// Método para registrar un voto
2 referencias
public void RegistrarVoto(string proyectoDeLey, string voto)
{
    string votoRegistrado = $"Proyecto: {proyectoDeLey}, Voto: {voto}";
    votos.Add(votoRegistrado);
}

// Método para mostrar la lista de votos
0 referencias
public void MostrarVotos()
{
    Console.WriteLine($"Lista de votos de {Nombre} {Apellido}:");
    foreach (var voto in votos)
    {
        Console.WriteLine(voto);
    }
}



// Método virtual para presentar una propuesta legislativa
6 referencias
public virtual void PresentarPropuestaLegislativa(string propuesta)
{
    Console.WriteLine($"{Nombre} {Apellido} del partido {PartidoPolitico} ha presentado la propuesta: '{propuesta}'");
    PropuestasLegislativas.Add(propuesta); // Agregar la propuesta a la lista de propuestas
}


// Método virtual para emitir un voto
0 referencias
public virtual void Votar(string proyectoDeLey, Voto voto)
{
    Console.WriteLine($"{Nombre} {Apellido} del partido {PartidoPolitico} ha votado '{voto}' en el proyecto de ley '{proyectoDeLey}'.");
}

// Método virtual para participar en un debate legislativo
3 referencias
public virtual void ParticiparDebate(string temaDebate)
{
    Console.WriteLine($"{Nombre} {Apellido} del partido {PartidoPolitico} ha participado en el debate sobre '{temaDebate}'.");
}
```

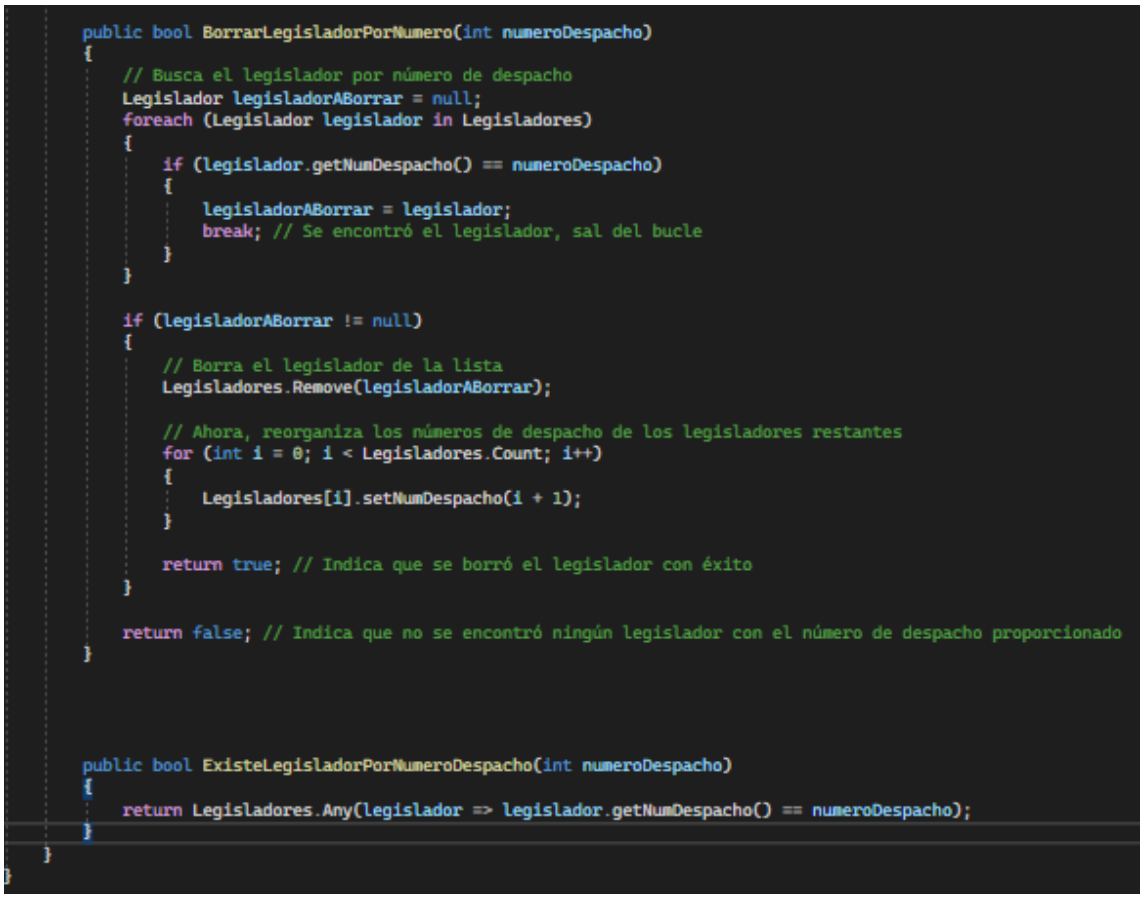
2.2.6 Mas setters y métodos polimórficos

2.5. Clase Parlamento

Clase Parlamento (internal class Parlamento): Esta clase representa un parlamento. Atributos: private List<Legislador> Legisladores;: Es una lista privada de objetos Legislador que almacena los legisladores en el parlamento. Constructor (public Parlamento()): Es el constructor de la clase Parlamento. Inicializa la lista de legisladores al crear un objeto de tipo Parlamento. Métodos: public List<Legislador> GetLegisladores(): Un método público que devuelve la lista de legisladores del parlamento. public void SetLegisladores(List<Legislador> legisladores): Un método público que permite establecer la lista de legisladores del parlamento. public void RegistrarLegislador(Legislador legislador): Un método público que agrega un legislador a la lista de legisladores del parlamento. public void ListarCamaras(): Un método público que lista el nombre, el número de despacho y la cámara (Senado o Diputados) de cada legislador en el parlamento. public void ContarLegisladoresPorTipo(): Un método público que cuenta y muestra la cantidad de senadores y diputados en el parlamento. public bool BorrarLegisladorPorNumero(int numeroDespacho): Un método público que busca un legislador por su número de despacho y lo borra de la lista. Luego reorganiza los números de despacho de los legisladores restantes. Devuelve true si el legislador fue borrado exitosamente y false si no se encontró ningún legislador con el número de despacho proporcionado. public bool ExisteLegisladorPorNumeroDespacho(int numeroDespacho): Un método público que verifica si existe un legislador con un número de despacho específico en la lista. Devuelve true si el legislador existe y false si no. Esta clase Parlamento proporciona métodos para gestionar una lista de legisladores, incluyendo la capacidad de agregar, listar, contar, borrar y verificar la existencia de legisladores en el parlamento.

```
internal class Parlamento
{
    private List<Legislador> Legisladores;

    public Parlamento()
    {
        Legisladores = new List<Legislador>();
    }

    public List<Legislador> GetLegisladores()
    {
        return Legisladores;
    }

    public void SetLegisladores(List<Legislador> legisladores)
    {
        this.Legisladores = legisladores;
    }

    public void RegistrarLegislador(Legislador legislador)
    {
        Legisladores.Add(legislador);
    }

    public void ListarCamaras()
    {
        foreach (var legislador in Legisladores)
        {
            Console.WriteLine($"\nNombre: {legislador.getNombre()}");
            Console.WriteLine($"Número de Despacho: {legislador.getNumDespacho()}");
            Console.WriteLine($"Cámara: {legislador.getCamara()}");
            Console.WriteLine();
        }
    }

    public void ContarLegisladoresPorTipo()
    {
        int cantidadSenadores = 0;
        int cantidadDiputados = 0;

        foreach (var legislador in Legisladores)
        {
            if (legislador.getCamara() == "Senado")
            {
                cantidadSenadores++;
            }
            else if (legislador.getCamara() == "Diputados")
            {
                cantidadDiputados++;
            }
        }

        Console.WriteLine($"\nCantidad de Senadores: {cantidadSenadores}");
        Console.WriteLine($"Cantidad de Diputados: {cantidadDiputados}");
    }
```

2.2.7 Clase Parlamento con Lista Legisladores, setter y métodos



```
        public bool BorrarLegisladorPorNumero(int numeroDespacho)
        {
            // Busca el legislador por número de despacho
            Legislador legisladorABorrar = null;
            foreach (Legislador legislador in Legisladores)
            {
                if (legislador.getNumDespacho() == numeroDespacho)
                {
                    legisladorABorrar = legislador;
                    break; // Se encontró el legislador, sal del bucle
                }
            }

            if (legisladorABorrar != null)
            {
                // Borra el legislador de la lista
                Legisladores.Remove(legisladorABorrar);

                // Ahora, reorganiza los números de despacho de los legisladores restantes
                for (int i = 0; i < Legisladores.Count; i++)
                {
                    Legisladores[i].setNumDespacho(i + 1);
                }

                return true; // Indica que se borró el legislador con éxito
            }

            return false; // Indica que no se encontró ningún legislador con el número de despacho proporcionado
        }



        public bool ExisteLegisladorPorNumeroDespacho(int numeroDespacho)
        {
            return Legisladores.Any(legislador => legislador.getNumDespacho() == numeroDespacho);
        }
    }
}
```

2.2.8 Mas métodos

3. Resultados

3.1 Menú

Este código representa el punto de entrada de un programa de consola. En este programa, los usuarios pueden interactuar con un sistema parlamentario simulado a través de varias opciones proporcionadas en un menú. A continuación, se detalla qué hace cada parte del código: Inicialización del Parlamento y Legisladores: Se crea una instancia de la clase Parlamento llamada miParlamento. Se crean dos instancias de la clase Legislador llamadas legislador1 y legislador2 con información predefinida (partido político, departamento, número de despacho, nombre, apellido, edad, estado civil). Estos legisladores también presentan propuestas y registran votos. Menú de Interacción: El programa entra en un bucle (do-while) que muestra un menú de opciones al usuario. Cada opción del menú realiza una acción específica: Opción '1': Muestra el total de legisladores en el parlamento y cuenta la cantidad de senadores y diputados. Opción '2': Lista los nombres, números de despacho y cámaras (Senado o Diputados) de todos los legisladores en el parlamento. Opción '3': Permite al usuario agregar votos a una propuesta presentada por un legislador seleccionado. Opción '4': Muestra el total de votos aprobados, rechazados y abstenciones de todas las propuestas presentadas por los legisladores en el parlamento. Opción '5': Permite al usuario presentar una propuesta legislativa en nombre de un legislador seleccionado. Opción '6': Permite al usuario agregar un nuevo legislador al parlamento (ya sea senador o diputado) con información proporcionada por el usuario. Opción '7': Permite al usuario borrar un legislador del parlamento por su número de despacho. Opción '8': Lista las propuestas legislativas presentadas por un legislador seleccionado. Opción '9': Permite al usuario seleccionar un legislador para participar en un debate sobre un tema proporcionado por el usuario. Opción '0': Sale del programa. Manejo de las Opciones del Usuario: Para cada opción del menú, el programa solicita la entrada del usuario y realiza acciones basadas en esa entrada. Se utilizan estructuras de control (switch, if) para manejar las diferentes opciones y ejecutar el código correspondiente a la opción seleccionada. Este código crea un programa de consola interactivo que simula un parlamento donde los usuarios pueden agregar legisladores, presentar propuestas, registrar votos, participar en debates y realizar otras acciones relacionadas con el proceso legislativo. El programa proporciona un menú intuitivo para que los usuarios realicen estas acciones.

```
0 referencias
static void Main(string[] args)
{
    char opcion;

    // Crear una instancia de Parlamento
    Parlamento miParlamento = new Parlamento();

    // Crear un legislador en "modo hardcore"
    Legislador legislador1 = new Legislador("Colorados", "Defensa", 2, "Guido Manini", "Rios", 65, true);
    Legislador legislador2 = new Legislador("Blancos", "Vicepresidente", 1, "Beatriz Argimón", "Cedeira", 62, true);

    // Agregar el legislador a la lista de legisladores del Parlamento
    miParlamento.RegistrarLegislador(legislador1);
    miParlamento.RegistrarLegislador(legislador2);
    legislador1.RegistrarVoto("Si a la baja", "Aprobado");
    legislador1.PresentarPropuestaLegislativa("Si a la baja");
    legislador2.RegistrarVoto("Si a la baja", "Aprobado");
    legislador2.PresentarPropuestaLegislativa("Aumento salario");

    do
    {
        Console.Clear(); // Limpia la consola

        // Mostrar el encabezado del menú
        Console.WriteLine("\t         ADMINISTRACION          ");
        Console.WriteLine("\n\t***********************************");
        Console.WriteLine("\t*             MENU                *");
        Console.WriteLine("\t***********************************");
        Console.WriteLine("\t* 1.    TOTAL LEGISLADORES        *");
        Console.WriteLine("\t* 2.    LISTAR CAMARA             *");
        Console.WriteLine("\t* 3.     AGREGAR VOTOS            *");
        Console.WriteLine("\t* 4.     TOTAL VOTOS              *");
        Console.WriteLine("\t* 5.    PROPUESTAS LEGISLATIVA    *");
        Console.WriteLine("\t* 6.    AGREGAR LEGISLADOR        *");
        Console.WriteLine("\t* 7.    BORRAR LEGISLADOR         *");
        Console.WriteLine("\t* 8.    LISTAR PROPUESTAS         *");
        Console.WriteLine("\t* 9.    PARTICIPAR DEBATE         *");
        Console.WriteLine("\t* 0.        SALIR                 *");
        Console.WriteLine("\t***********************************");
        Console.Write("\tSeleccione una opción: ");

        opcion = Console.ReadKey().KeyChar;
```

3.1.1 Parte del código del Menú (se muestra una parte del código porque no cabe la captura de pantalla con el código completo).

```
         ADMINISTRACION

***************************************
*              MENU                   *
***************************************
* 1.    TOTAL LEGISLADORES            *
* 2.    LISTAR CAMARA                 *
* 3.     AGREGAR VOTOS                *
* 4.     TOTAL VOTOS                  *
* 5.    PROPUESTAS LEGISLATIVA        *
* 6.    AGREGAR LEGISLADOR            *
* 7.    BORRAR LEGISLADOR             *
* 8.    LISTAR PROPUESTAS             *
* 9.    PARTICIPAR DEBATE             *
* 0.        SALIR                     *
***************************************
Seleccione una opción:
```

3.1.2 Menú mostrado por consola

4. Conclusiones

Aprendimos que es la programación orientada a objetos y como aplicarlo en la practica y facilitar a preparar en como ordenar, escribir el código y reutilizar métodos.

Conclusión personal: Entiendo que faltan cosas en el informe y que no este explicado mas detalladamente, pero sinceramente no me dio tiempo en prepararlo como se debe el informe por enfocarme en escribir el código e intentar arreglar algunos métodos que no funcionaban correctamente en el menú a la hora de invocar.

5. Referencias

[1] https://bsw.es/que-es-c/

[2] https://webdesigncusco.com/conceptos-basicos-de-la-programacion-orientada-a-objetos-poo/#:~:text=La%20programaci%C3%B3n%20orientada%20a%20objetos%20(POO)%20es%20un%20paradigma%20de,%2C%20C%2B%2B%2C%20entre%20otros.

[3] https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/object-oriented/inheritance

[4] https://algonzalezpoo.wordpress.com/polimorfismo/

[5] https://miro.com/es/diagrama/que-es-diagrama-uml/