TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería Informática

MATERIA:
Patrones de Diseño

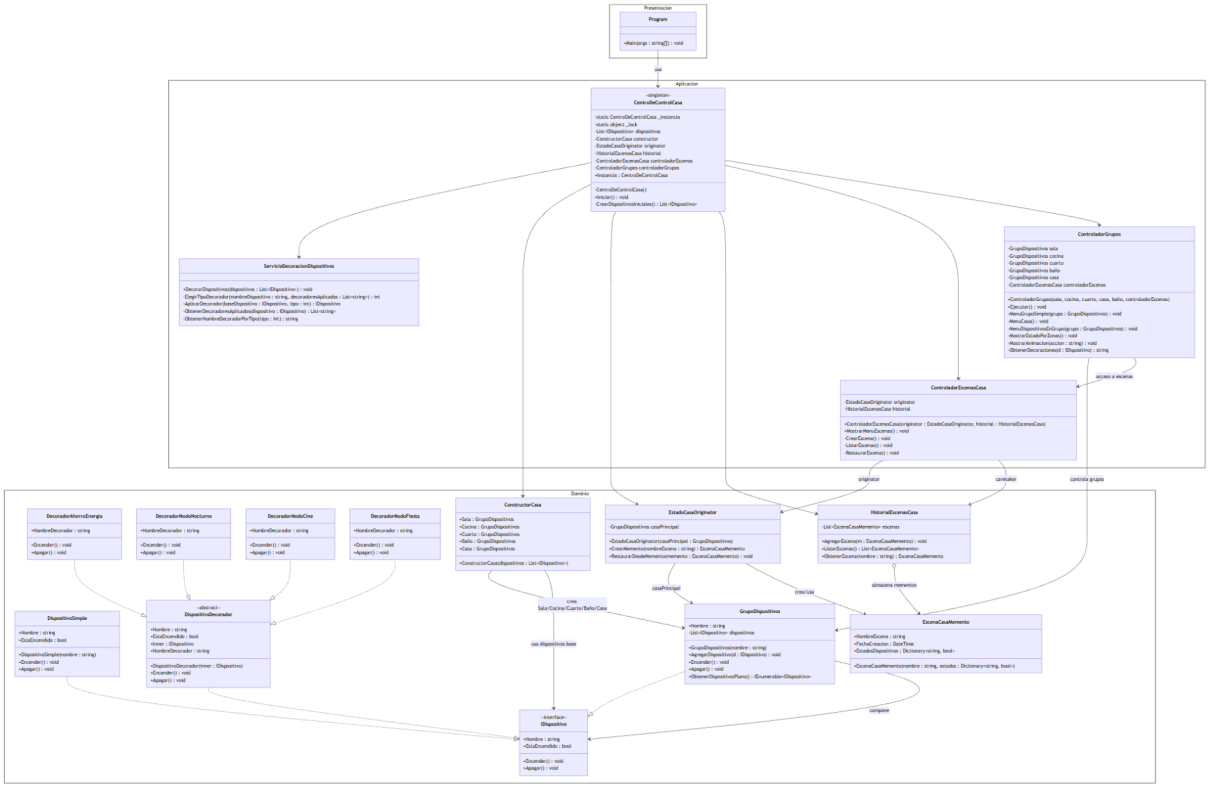
TÍTULO ACTIVIDAD:
Examen unidad 5

UNIDAD A EVALUAR:
5

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:
Gonzalez Gutierrez Daniel Alejandro 21212341

NOMBRE DEL MAESTRO (A):
Maribel Guerrero Luis

Diagrama UML



Código
Aplicación
CentroDeControlCasa

```
namespace Aplicacion
{
    5 referencias
    public sealed class CentroDeControlCasa
    {


        private static CentroDeControlCasa _instancia = null;
        private static readonly object _lock = new object();

        1 referencia
        public static CentroDeControlCasa Instancia
        {
            get
            {
                if (_instancia == null)
                {
                    lock (_lock)
                    {
                        if (_instancia == null)
                        {
                            _instancia = new CentroDeControlCasa();
                        }
                    }
                }

                return _instancia;
            }
        }


        private List<IDispositivo> dispositivos;
        private ConstructorCasa constructor;
        private EstadoCasaOriginator originator;
        private HistorialEscenasCasa historial;
        private ControladorEscenasCasa controladorEscenas;
        private ControladorGrupos controladorGrupos;

        1 referencia
        private CentroDeControlCasa() { }




        1 referencia
        public void Iniciar()
        {
            dispositivos = CrearDispositivosIniciales();

            var servicioDecoracion = new ServicioDecoracionDispositivos();
            servicioDecoracion.DecorarDispositivos(dispositivos);

            constructor = new ConstructorCasa(dispositivos);

            originator = new EstadoCasaOriginator(constructor.Casa);
            historial = new HistorialEscenasCasa();

            controladorEscenas = new ControladorEscenasCasa(originator, historial);

            controladorGrupos = new ControladorGrupos(
                constructor.Sala,
                constructor.Cocina,
                constructor.Cuarto,
                constructor.Casa,
                constructor.Baño,
                controladorEscenas
            );

            controladorGrupos.Ejecutar();
        }

        1 referencia
        private List<IDispositivo> CrearDispositivosIniciales()
        {
            return new List<IDispositivo>
            {
                new DispositivoSimple("Televisor"),
                new DispositivoSimple("Bocinas"),
                new DispositivoSimple("Foco sala"),
                new DispositivoSimple("Microondas"),
                new DispositivoSimple("Refrigerador"),
                new DispositivoSimple("Luces LED cuarto"),
                new DispositivoSimple("PC escritorio"),
                new DispositivoSimple("Foco Sanitario")
            };
        }
    }
}
```

ControladorEscenasCasa

```
5 referencias
public class ControladorEscenasCasa
{
    private readonly EstadoCasaOriginator originatorCasa;
    private readonly HistorialEscenasCasa historialEscenas;

    1 referencia
    public ControladorEscenasCasa(EstadoCasaOriginator originatorCasa, HistorialEscenasCasa historialEscen
    {
        this.originatorCasa = originatorCasa;
        this.historialEscenas = historialEscenas;
    }

    1 referencia
    public void EjecutarMenuEscenas()
    {
        bool regresar = false;

        while (!regresar)
        {
            Console.Clear();
            Console.WriteLine("====   Gestor de Guardado de casa    ====================\n");
            Console.WriteLine("1. Guardar configuración actual como nueva escena");
            Console.WriteLine("2. Listar escenas guardadas");
            Console.WriteLine("3. Restaurar una escena");
            Console.WriteLine("4. Eliminar una escena");
            Console.WriteLine("5. Limpiar todas las escenas");
            Console.WriteLine("6. Regresar");

            string opcion = Console.ReadLine();

            if (opcion == "1")
            {
                Console.Write("Nombre para la nueva escena: ");
                string nombre = Console.ReadLine();

                if (string.IsNullOrWhiteSpace(nombre))
                {
                    nombre = "Escena " + (historialEscenas.Escenas.Count + 1);
                }

                var memento = originatorCasa.CrearMemento(nombre);
                historialEscenas.AgregarEscena(memento);

                Console.WriteLine("\nEscena guardada correctamente.");
                Console.WriteLine("Presione una tecla para continuar...");
                Console.ReadKey();
            }
            else if (opcion == "2")
            {
                Console.Clear();
                Console.WriteLine("====   Escenas guardadas   ====================\n");

                if (!historialEscenas.TieneEscenas)
                {
                    Console.WriteLine("No hay escenas guardadas.");
                }
                else
                {
                    for (int i = 0; i < historialEscenas.Escenas.Count; i++)
                    {
                        var e = historialEscenas.Escenas[i];
                        Console.WriteLine((i + 1) + ". " + e.Nombre + " - " + e.FechaCreacion);
                    }
                }

                Console.WriteLine("\nPresione una tecla para continuar...");
                Console.ReadKey();
```

```
        else if (opcion == "3")
        {
            if (!historialEscenas.TieneEscenas)
            {
                Console.WriteLine("\nNo hay escenas para restaurar.");
                Console.WriteLine("Presione una tecla para continuar...");
                Console.ReadKey();
                continue;
            }

            Console.Clear();
            Console.WriteLine("====   Restaurar escena   ====================\n");
            for (int i = 0; i < historialEscenas.Escenas.Count; i++)
            {
                var e = historialEscenas.Escenas[i];
                Console.WriteLine((i + 1) + ". " + e.Nombre + " - " + e.FechaCreacion);
            }

            Console.Write("\nSeleccione el número de la escena a restaurar: ");
            string entrada = Console.ReadLine();
            if (int.TryParse(entrada, out int indice))
            {
                indice -= 1;
                var escena = historialEscenas.ObtenerEscena(indice);
                if (escena != null)
                {
                    Console.Clear();
                    Console.WriteLine("Restaurando escena: " + escena.Nombre);
                    MostrarAnimacion("Restaurando");
                    originatorCasa.RestaurarDesdeMemento(escena);
                    Console.WriteLine("\nEscena restaurada correctamente.");
                }
                else
                {
                    Console.WriteLine("Índice de escena inválido.");
                }
            }
            else
            {
                Console.WriteLine("Entrada inválida.");
            }

            Console.WriteLine("\nPresione una tecla para continuar...");
            Console.ReadKey();
        }
        else if (opcion == "4")
        {
            if (!historialEscenas.TieneEscenas)
            {
                Console.WriteLine("\nNo hay escenas para eliminar.");
                Console.WriteLine("Presione una tecla para continuar...");
                Console.ReadKey();
                continue;
            }

            Console.Clear();
            Console.WriteLine("====   Eliminar escena   ====================\n");
            for (int i = 0; i < historialEscenas.Escenas.Count; i++)
            {
                var e = historialEscenas.Escenas[i];
                Console.WriteLine((i + 1) + ". " + e.Nombre + " - " + e.FechaCreacion);
            }

            Console.Write("\nSeleccione el número de la escena a eliminar: ");
            string entrada = Console.ReadLine();
            if (int.TryParse(entrada, out int indice))
            {
                indice -= 1;
                var escena = historialEscenas.ObtenerEscena(indice);
                if (escena != null)
                {
                    historialEscenas.EliminarEscena(indice);
                    Console.WriteLine("\nEscena eliminada correctamente.");
                }
                else
                {
                    Console.WriteLine("Índice de escena inválido.");
                }
            }
            else
            {
                Console.WriteLine("Entrada inválida.");
            }

            Console.WriteLine("\nPresione una tecla para continuar...");
            Console.ReadKey();
        }
        else if (opcion == "5")
        {
            if (!historialEscenas.TieneEscenas)
            {
                Console.WriteLine("\nNo hay escenas que limpiar.");
            }
            else
            {
                historialEscenas.Limpiar();
                Console.WriteLine("\nTodas las escenas han sido eliminadas.");
            }
```

```
                }

                Console.WriteLine("Presione una tecla para continuar...");
                Console.ReadKey();
            }
            else if (opcion == "6")
            {
                regresar = true;
            }
            else
            {
                Console.WriteLine("Opción no válida.");
                Console.WriteLine("Presione una tecla para continuar...");
                Console.ReadKey();
            }
        }
    }

    1 referencia
    private static void MostrarAnimacion(string texto)
    {
        Console.Write("Cargando");
        int dotCount = 10;
        int dotsStartLeft = Console.CursorLeft;
        int dotsStartTop = Console.CursorTop;
        string prefix = "Cargando";

        for (int i = 0; i < dotCount; i++)
        {
            System.Threading.Thread.Sleep(200);
            Console.Write(".");
        }
        int startLeftFull = Math.Max(0, dotsStartLeft - prefix.Length);
        Console.SetCursorPosition(startLeftFull, dotsStartTop);
        Console.Write(new string(' ', prefix.Length + dotCount));
        Console.SetCursorPosition(0, dotsStartTop + 1);
    }
}
```

ControladorGrupos

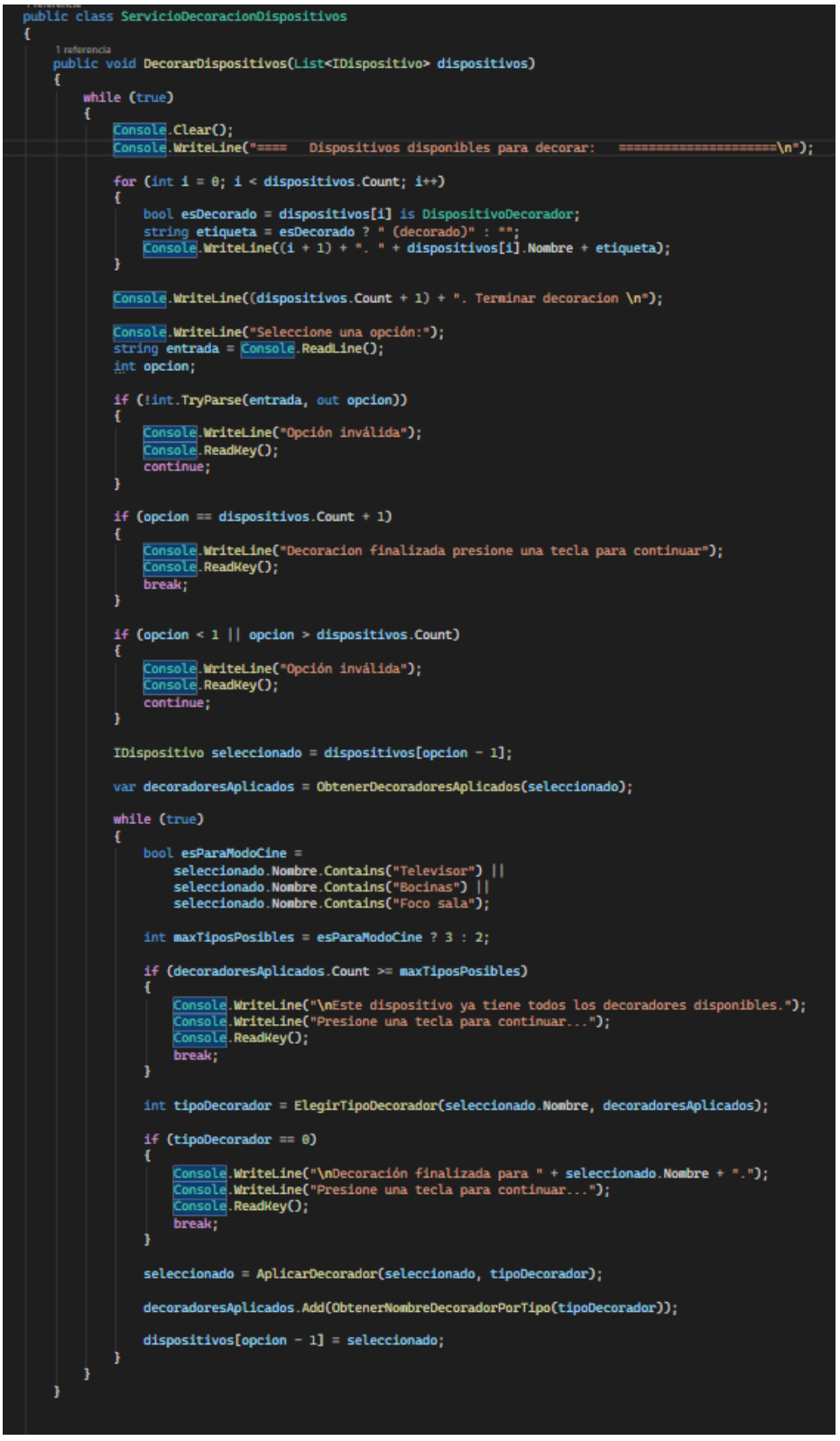
```
public class ControladorGrupos
{
    private readonly GrupoDispositivos sala;
    private readonly GrupoDispositivos cocina;
    private readonly GrupoDispositivos cuarto;
    private readonly GrupoDispositivos casa;
    private readonly GrupoDispositivos baño;
    private readonly ControladorEscenasCasa controladorEscenas;

    1 referencia
    public ControladorGrupos(
        GrupoDispositivos sala,
        GrupoDispositivos cocina,
        GrupoDispositivos cuarto,
        GrupoDispositivos casa,
        GrupoDispositivos baño,
        ControladorEscenasCasa controladorEscenas)
    {
        this.sala = sala;
        this.cocina = cocina;
        this.cuarto = cuarto;
        this.casa = casa;
        this.baño = baño;
        this.controladorEscenas = controladorEscenas;
    }

    1 referencia
    public void Ejecutar()
    {
        bool salir = false;

        while (!salir)
        {
            Console.Clear();
            Console.WriteLine("====   Control de dispositivos inteligentes   ==================
            Console.WriteLine("Seleccione un grupo:");
            Console.WriteLine("1. Sala");
            Console.WriteLine("2. Cocina");
            Console.WriteLine("3. Cuarto");
            Console.WriteLine("4. Baño");
            Console.WriteLine("6. Casa");
            Console.WriteLine("7. Salir");

            string opcionGrupo = Console.ReadLine();

            GrupoDispositivos grupoSeleccionado = null;

            if (opcionGrupo == "1") grupoSeleccionado = sala;
            else if (opcionGrupo == "2") grupoSeleccionado = cocina;
            else if (opcionGrupo == "3") grupoSeleccionado = cuarto;
            else if (opcionGrupo == "4") grupoSeleccionado = baño;
            else if (opcionGrupo == "6") grupoSeleccionado = casa;
            else if (opcionGrupo == "7")
            {
                salir = true;
                Console.WriteLine("Saliendo de control de dispositivos.");
                Console.ReadKey();
            }

            if (grupoSeleccionado == null)
            {
                if (!salir)
                {
                    Console.WriteLine("Opción no válida");
                    Console.ReadKey();
                }
                continue;
            }

            if (grupoSeleccionado == casa)
            {
                MenuCasa();
            }
            else
            {
                MenuGrupoSimple(grupoSeleccionado);
            }
        }
    }

    1 referencia
    private void MenuGrupoSimple(GrupoDispositivos grupo)
    {
        bool regresar = false;

        while (!regresar)
        {
            Console.Clear();
            Console.WriteLine("Seleccione acción para el grupo " + grupo.Nombre + "\n");
            Console.WriteLine("1. Encender todo el grupo");
            Console.WriteLine("2. Apagar todo el grupo");
            Console.WriteLine("3. Encender/Apagar dispositivo específico");
            Console.WriteLine("4. Regresar");

            string opcion = Console.ReadLine();
```

```
    var hojasGrupo = grupo.ObtenerDispositivosPlano().ToList();

    if (opcion == "1")
    {
        bool hayApagados = hojasGrupo.Any(d => !d.EstaEncendido);
        if (!hayApagados)
        {
            Console.Clear();
            Console.WriteLine("====   Grupo " + grupo.Nombre + "   ====================\n");
            Console.WriteLine("No hay dispositivos para encender");
            Console.WriteLine("\n=====================================\n");
            Console.WriteLine("\nPresione una tecla para regresar");
            Console.ReadKey();
        }
        else
        {
            Console.Clear();
            Console.WriteLine("====   Grupo " + grupo.Nombre + "   ====================");
            MostrarAnimacion("Encendiendo");
            grupo.Encender();
            Console.WriteLine("\n=====================================\n");
            Console.WriteLine("\nPresione una tecla para regresar");
            Console.ReadKey();
        }
    }
    else if (opcion == "2")
    {
        bool hayEncendidos = hojasGrupo.Any(d => d.EstaEncendido);
        if (!hayEncendidos)
        {
            Console.Clear();
            Console.WriteLine("====   Grupo " + grupo.Nombre + "   ====================\n");
            Console.WriteLine("No hay dispositivos para apagar");
            Console.WriteLine("\n=====================================\n");
            Console.WriteLine("\nPresione una tecla para regresar");
            Console.ReadKey();
        }
        else
        {
            Console.Clear();
            Console.WriteLine("====   Grupo " + grupo.Nombre + "   ====================");
            MostrarAnimacion("Apagando");
            grupo.Apagar();
            Console.WriteLine("\n=====================================\n");
            Console.WriteLine("\nPresione una tecla para regresar");
            Console.ReadKey();
        }
    }
    else if (opcion == "3")
    {
        MenuDispositivosEnGrupo(grupo);
    }
    else if (opcion == "4")
    {
        regresar = true;
    }
    else
    {
        Console.WriteLine("Opción no válida");
        Console.ReadKey();
    }
}

referencia
rivate void MenuDispositivosEnGrupo(GrupoDispositivos grupo)

  bool regresar = false;

  while (!regresar)
  {
      var dispositivos = grupo.ObtenerDispositivosPlano().ToList();

      Console.Clear();
      Console.WriteLine("====   Dispositivos en " + grupo.Nombre + "   ====================\n");

      if (dispositivos.Count == 0)
      {
          Console.WriteLine("No hay dispositivos en este grupo.");
          Console.WriteLine("\nPresione una tecla para regresar...");
          Console.ReadKey();
          return;
      }

      for (int i = 0; i < dispositivos.Count; i++)
      {
          var d = dispositivos[i];
          Console.WriteLine(
              (i + 1) + ". " + d.Nombre + " - " + (d.EstaEncendido ? "Encendido" : "Apagado"));
      }

      Console.WriteLine((dispositivos.Count + 1) + ". Regresar\n");
      Console.Write("Seleccione un dispositivo para alternar su estado (ON/OFF): ");

      string entrada = Console.ReadLine();
      int opcion;

      if (!int.TryParse(entrada, out opcion))
      {
```

```
    if (!int.TryParse(entrada, out opcion))
    {
        Console.WriteLine("Opción inválida.");
        Console.ReadKey();
        continue;
    }

    if (opcion == dispositivos.Count + 1)
    {
        regresar = true;
        continue;
    }

    if (opcion < 1 || opcion > dispositivos.Count)
    {
        Console.WriteLine("Opción fuera de rango.");
        Console.ReadKey();
        continue;
    }

    var seleccionado = dispositivos[opcion - 1];

    Console.Clear();
    Console.WriteLine("====    " + seleccionado.Nombre + "    ====================\n");

    if (seleccionado.EstaEncendido)
    {
        MostrarAnimacion("Apagando");
        seleccionado.Apagar();
    }
    else
    {
        MostrarAnimacion("Encendiendo");
        seleccionado.Encender();
    }

    Console.WriteLine("\n=====================================\n");
    Console.WriteLine("Presione una tecla para continuar...");
    Console.ReadKey();
}


referencia
rivate void MenuCasa()

bool regresar = false;

while (!regresar)
{
    Console.Clear();
    Console.WriteLine("Seleccione acción para el grupo " + casa.Nombre + "\n");
    Console.WriteLine("1. Encender");
    Console.WriteLine("2. Apagar");
    Console.WriteLine("3. Mostrar estado de dispositivos");
    Console.WriteLine("4. Gestionar  guardado dee escenas");
    Console.WriteLine("5. Regresar");

    string opcion = Console.ReadLine();

    var hojasCasa = casa.ObtenerDispositivosPlano().ToList();

    if (opcion == "1")
    {
        bool hayApagados = hojasCasa.Any(d => !d.EstaEncendido);

        Console.Clear();
        Console.WriteLine("====   Grupo " + casa.Nombre + "   ====================");

        if (!hayApagados)
        {
            Console.WriteLine();
            Console.WriteLine("No hay dispositivos para encender");
        }
        else
        {
            MostrarAnimacion("Encendiendo");
            casa.Encender();
        }

        Console.WriteLine("\n=====================================");
        Console.WriteLine("\nPresione una tecla para regresar");
        Console.ReadKey();
    }
    else if (opcion == "2")
    {
        bool hayEncendidos = hojasCasa.Any(d => d.EstaEncendido);

        Console.Clear();
        Console.WriteLine("====   Grupo " + casa.Nombre + "   ====================");

        if (!hayEncendidos)
        {
            Console.WriteLine();
            Console.WriteLine("No hay dispositivos para apagar");
        }
        else
        {
            MostrarAnimacion("Apagando");
            casa.Apagar();
        }
```

```
                }

                Console.WriteLine("\n=====================================");
                Console.WriteLine("\nPresione una tecla para regresar");
                Console.ReadKey();
            }
            else if (opcion == "3")
            {
                Console.Clear();
                Console.WriteLine("=====================================");
                Console.WriteLine("Estado de dispositivos en " + casa.Nombre + ":");
                Console.WriteLine("=====================================\n");
                MostrarEstadoPorZonas();
                Console.WriteLine("\n=====================================");
                Console.WriteLine("\nPresione una tecla para regresar");
                Console.ReadKey();
            }
            else if (opcion == "4")
            {
                controladorEscenas.EjecutarMenuEscenas();
            }
            else if (opcion == "5")
            {
                regresar = true;
            }
            else
            {
                Console.WriteLine("Opción no válida");
                Console.ReadKey();
            }
        }
    }

    1 referencia
    private void MostrarEstadoPorZonas()
    {
        foreach (var sub in new[] { sala, cocina, cuarto, baño })
        {
            Console.WriteLine();
            Console.WriteLine(sub.Nombre + ":");
            foreach (var d in sub.ObtenerDispositivosPlano())
            {
                string decor = ObtenerDecoraciones(d);
                Console.WriteLine("  - " + d.Nombre + " : " + (d.EstaEncendido ? "Encendido" : "Apagado") + decor);
            }
        }
    }

    6 referencias
    private static void MostrarAnimacion(string texto)
    {
        Console.Write("Cargando");
        int dotCount = 10;
        int dotsStartLeft = Console.CursorLeft;
        int dotsStartTop = Console.CursorTop;
        string prefix = "Cargando";

        for (int i = 0; i < dotCount; i++)
        {
            Thread.Sleep(200);
            Console.Write(".");
        }
        int startLeftFull = Math.Max(0, dotsStartLeft - prefix.Length);
        Console.SetCursorPosition(startLeftFull, dotsStartTop);
        Console.Write(new string(' ', prefix.Length + dotCount));
        Console.SetCursorPosition(0, dotsStartTop + 1);
    }

    1 referencia
    private static string ObtenerDecoraciones(IDispositivo d)
    {
        var decoradores = new List<string>();
        while (d is DispositivoDecorador dec)
        {
            decoradores.Add(dec.NombreDecorador);
            d = dec.Inner;
        }

        if (decoradores.Count == 0) return string.Empty;

        decoradores.Reverse();
        return " ( " + string.Join(", ", decoradores) + " )";
    }
}
}
```

ServicioDecoracionDispositivos



```
public class ServicioDecoracionDispositivos
{
    1 referencia
    public void DecorarDispositivos(List<IDispositivo> dispositivos)
    {
        while (true)
        {
            Console.Clear();
            Console.WriteLine("====   Dispositivos disponibles para decorar:   ====================\n");

            for (int i = 0; i < dispositivos.Count; i++)
            {
                bool esDecorado = dispositivos[i] is DispositivoDecorador;
                string etiqueta = esDecorado ? " (decorado)" : "";
                Console.WriteLine((i + 1) + ". " + dispositivos[i].Nombre + etiqueta);
            }

            Console.WriteLine((dispositivos.Count + 1) + ". Terminar decoracion \n");

            Console.WriteLine("Seleccione una opción:");
            string entrada = Console.ReadLine();
            int opcion;

            if (!int.TryParse(entrada, out opcion))
            {
                Console.WriteLine("Opción inválida");
                Console.ReadKey();
                continue;
            }

            if (opcion == dispositivos.Count + 1)
            {
                Console.WriteLine("Decoracion finalizada presione una tecla para continuar");
                Console.ReadKey();
                break;
            }

            if (opcion < 1 || opcion > dispositivos.Count)
            {
                Console.WriteLine("Opción inválida");
                Console.ReadKey();
                continue;
            }

            IDispositivo seleccionado = dispositivos[opcion - 1];

            var decoradoresAplicados = ObtenerDecoradoresAplicados(seleccionado);

            while (true)
            {
                bool esParaModoCine =
                    seleccionado.Nombre.Contains("Televisor") ||
                    seleccionado.Nombre.Contains("Bocinas") ||
                    seleccionado.Nombre.Contains("Foco sala");

                int maxTiposPosibles = esParaModoCine ? 3 : 2;

                if (decoradoresAplicados.Count >= maxTiposPosibles)
                {
                    Console.WriteLine("\nEste dispositivo ya tiene todos los decoradores disponibles.");
                    Console.WriteLine("Presione una tecla para continuar...");
                    Console.ReadKey();
                    break;
                }

                int tipoDecorador = ElegirTipoDecorador(seleccionado.Nombre, decoradoresAplicados);

                if (tipoDecorador == 0)
                {
                    Console.WriteLine("\nDecoración finalizada para " + seleccionado.Nombre + ".");
                    Console.WriteLine("Presione una tecla para continuar...");
                    Console.ReadKey();
                    break;
                }

                seleccionado = AplicarDecorador(seleccionado, tipoDecorador);

                decoradoresAplicados.Add(ObtenerNombreDecoradorPorTipo(tipoDecorador));

                dispositivos[opcion - 1] = seleccionado;
            }
        }
    }
```

```
    1 referencia
    private int ElegirTipoDecorador(string nombreDispositivo, List<string> decoradoresAplicados)
    {
        while (true)
        {
            Console.Clear();

            Console.WriteLine();
            Console.WriteLine("=======================================");
            Console.WriteLine("Seleccione decorador para " + nombreDispositivo);
            Console.WriteLine("=======================================");
            Console.WriteLine("Elija decorador:\n");

            bool esParaModoCine =
                nombreDispositivo.Contains("Televisor") ||
                nombreDispositivo.Contains("Bocinas") ||
                nombreDispositivo.Contains("Foco sala");

            bool tieneAhorro = decoradoresAplicados.Contains("Ahorro de energía");
            bool tieneNocturno = decoradoresAplicados.Contains("Modo nocturno");
            bool tieneCine = decoradoresAplicados.Contains("Modo Cine");

            if (!tieneAhorro)
                Console.WriteLine("1. Decorador ahorro de energía");

            if (!tieneNocturno)
                Console.WriteLine("2. Decorador modo nocturno");

            if (!tieneCine && esParaModoCine)
                Console.WriteLine("3. Decorador modo cine");

            Console.WriteLine("4. Terminar decoración");

            string opcion = Console.ReadLine();

            if (opcion == "1" && !tieneAhorro) return 1;
            if (opcion == "2" && !tieneNocturno) return 2;
            if (opcion == "3" && esParaModoCine && !tieneCine) return 3;
            if (opcion == "4") return 0;

            Console.WriteLine("Opción no válida, intente de nuevo.");
            Console.ReadKey();
        }
    }

    1 referencia
    private IDispositivo AplicarDecorador(IDispositivo baseDispositivo, int tipo)
    {
        if (tipo == 1) return new DecoradorAhorroEnergia(baseDispositivo);
        if (tipo == 2) return new DecoradorModoNocturno(baseDispositivo);
        if (tipo == 3) return new DecoradorModoCine(baseDispositivo);
        return baseDispositivo;
    }

    1 referencia
    private List<string> ObtenerDecoradoresAplicados(IDispositivo dispositivo)
    {
        var decoradores = new List<string>();
        IDispositivo actual = dispositivo;

        while (actual is DispositivoDecorador dec)
        {
            decoradores.Add(dec.NombreDecorador);
            actual = dec.Inner;
        }

        return decoradores;
    }

    1 referencia
    private string ObtenerNombreDecoradorPorTipo(int tipo)
    {
        if (tipo == 1) return "Ahorro de energía";
        if (tipo == 2) return "Modo nocturno";
        if (tipo == 3) return "Modo Cine";
        return "";
    }
}
```

Dominio

ConstructorCasa

```
namespace Dominio
{
    3 referencias
    public class ConstructorCasa
    {
        6 referencias
        public GrupoDispositivos Sala { get; }
        5 referencias
        public GrupoDispositivos Cocina { get; }
        5 referencias
        public GrupoDispositivos Cuarto { get; }
        7 referencias
        public GrupoDispositivos Casa { get; }
        4 referencias
        public GrupoDispositivos Baño { get; }


        1 referencia
        public ConstructorCasa(List<IDispositivo> dispositivos)
        {
            if (dispositivos == null || dispositivos.Count < 8)
                throw new ArgumentException("");

            Sala = new GrupoDispositivos("Sala");
            Sala.AgregarDispositivo(dispositivos[0]);
            Sala.AgregarDispositivo(dispositivos[1]);
            Sala.AgregarDispositivo(dispositivos[2]);


            Cocina = new GrupoDispositivos("Cocina");
            Cocina.AgregarDispositivo(dispositivos[3]);
            Cocina.AgregarDispositivo(dispositivos[4]);

            Cuarto = new GrupoDispositivos("Cuarto");
            Cuarto.AgregarDispositivo(dispositivos[5]);
            Cuarto.AgregarDispositivo(dispositivos[6]);


            Baño = new GrupoDispositivos("Baño");
            Baño.AgregarDispositivo(dispositivos[7]);



            Casa = new GrupoDispositivos("Casa");
            Casa.AgregarDispositivo(Sala);
            Casa.AgregarDispositivo(Cocina);
            Casa.AgregarDispositivo(Cuarto);
            Casa.AgregarDispositivo(Baño);
        }
    }
}
```

DecoradorAhorroEnergia

```
    public class DecoradorAhorroEnergia : DispositivoDecorador
    {
        1 referencia
        public DecoradorAhorroEnergia(IDispositivo dispositivo) : base(dispositivo)
        {
        }

        3 referencias
        public override string NombreDecorador => "Ahorro de energía";

        10 referencias
        public override void Encender()
        {
            base.Encender();
            Console.WriteLine("(Activando modo ahorro de energía en " + Nombre + ")");
        }

        10 referencias
        public override void Apagar()
        {
            Console.WriteLine("(Guardando consumo de energía de " + Nombre + ")");
            base.Apagar();
        }
    }
}
```

DecoradorModoCine

```
public class DecoradorModoCine : DispositivoDecorador
{
    1 referencia
    public DecoradorModoCine(IDispositivo dispositivo) : base(dispositivo)
    {
    }

    3 referencias
    public override string NombreDecorador => "Modo Cine";

    10 referencias
    public override void Encender()
    {
        base.Encender();
        Console.WriteLine("(Activando configuración de Modo Cine en " + Nombre + ": brillo optimizado, sonido envolvente)");
    }

    10 referencias
    public override void Apagar()
    {
        Console.WriteLine("(Saliendo de Modo Cine en " + Nombre + ")");
        base.Apagar();
    }
}
```

DecoradorModoFiesta

```
public class DecoradorModoFiesta : DispositivoDecorador
{
    0 referencias
    public DecoradorModoFiesta(IDispositivo dispositivo) : base(dispositivo)
    {
    }

    3 referencias
    public override string NombreDecorador => "Modo Fiesta";

    10 referencias
    public override void Encender()
    {
        base.Encender();
        Console.WriteLine("(Activando Modo Fiesta en " + Nombre + ": luces dinámicas, música intensa)");
    }

    10 referencias
    public override void Apagar()
    {
        Console.WriteLine("(Desactivando Modo Fiesta en " + Nombre + ")");
        base.Apagar();
    }
}
```

DecoradorModoNocturno

```
public class DecoradorModoNocturno : DispositivoDecorador
{
    1 referencia
    public DecoradorModoNocturno(IDispositivo dispositivo) : base(dispositivo)
    {
    }

    3 referencias
    public override string NombreDecorador => "Modo nocturno";

    10 referencias
    public override void Encender()
    {
        base.Encender();
        Console.WriteLine("(Ajustando brillo y volumen para modo nocturno en " + Nombre + ")");
    }

    10 referencias
    public override void Apagar()
    {
        Console.WriteLine("(Restaurando configuraciones normales de " + Nombre + ")");
        base.Apagar();
    }
}
```

DispositivoDecorador

```
public abstract class DispositivoDecorador : IDispositivo
{
    protected IDispositivo dispositivo;

    4 referencias
    protected DispositivoDecorador(IDispositivo dispositivo)
    {
        this.dispositivo = dispositivo;
    }

    2 referencias
    public IDispositivo Inner => dispositivo;

    22 referencias
    public virtual string Nombre => dispositivo.Nombre;

    13 referencias
    public virtual bool EstaEncendido => dispositivo.EstaEncendido;

    6 referencias
    public virtual string NombreDecorador => "Decorador";

    13 referencias
    public virtual void Encender()
    {
        dispositivo.Encender();
    }

    13 referencias
    public virtual void Apagar()
    {
        dispositivo.Apagar();
    }
}
```

DispositivoSimple

```
9 referencias
public class DispositivoSimple : IDispositivo
{
    17 referencias
    public string Nombre { get; }

    private bool encendido;

    13 referencias
    public bool EstaEncendido => encendido;

    8 referencias
    public DispositivoSimple(string nombre)
    {
        Nombre = nombre;
        encendido = false;
    }

    5 referencias
    public virtual void Encender()
    {
        if (!encendido)
        {
            encendido = true;
            Console.WriteLine("--- " + Nombre + " Encendido");
        }
    }

    5 referencias
    public virtual void Apagar()
    {
        if (encendido)
        {
            encendido = false;
            Console.WriteLine("--- " + Nombre + " Apagado");
        }
    }
}
```

EscenaCasaMemento

```
namespace Dominio
{
    9 referencias
    public class EscenaCasaMemento
    {
        5 referencias
        public string Nombre { get; }
        4 referencias
        public DateTime FechaCreacion { get; }
        2 referencias
        public Dictionary<string, bool> EstadosDispositivos { get; }

        1 referencia
        public EscenaCasaMemento(string nombre, DateTime fechaCreacion, Dictionary<string, bool> estadosDispositivos)
        {
            Nombre = nombre;
            FechaCreacion = fechaCreacion;
            EstadosDispositivos = estadosDispositivos ?? new Dictionary<string, bool>();
        }
    }
}
```

EstadoCasaOriginator

```
5 referencias
public class EstadoCasaOriginator
{
    private readonly GrupoDispositivos casa;

    1 referencia
    public EstadoCasaOriginator(GrupoDispositivos casa)
    {
        this.casa = casa ?? throw new ArgumentNullException(nameof(casa));
    }


    1 referencia
    public EscenaCasaMemento CrearMemento(string nombreEscena)
    {
        var estados = new Dictionary<string, bool>();

        foreach (var dispositivo in casa.ObtenerDispositivosPlano())
        {
            if (!estados.ContainsKey(dispositivo.Nombre))
            {
                estados.Add(dispositivo.Nombre, dispositivo.EstaEncendido);
            }
        }

        return new EscenaCasaMemento(nombreEscena, DateTime.Now, estados);
    }


    1 referencia
    public void RestaurarDesdeMemento(EscenaCasaMemento memento)
    {
        if (memento == null) return;

        foreach (var dispositivo in casa.ObtenerDispositivosPlano())
        {
            if (memento.EstadosDispositivos.TryGetValue(dispositivo.Nombre, out bool encender))
            {
                if (encender && !dispositivo.EstaEncendido)
                {
                    dispositivo.Encender();
                }
                else if (!encender && dispositivo.EstaEncendido)
                {
                    dispositivo.Apagar();
                }
            }
        }
    }
}
```

GrupoDispositivos

```
27 referencias
public class GrupoDispositivos : IDispositivo
{
    private readonly List<IDispositivo> dispositivos = new List<IDispositivo>();

    28 referencias
    public string Nombre { get; }

    5 referencias
    public GrupoDispositivos(string nombre)
    {
        Nombre = nombre;
    }

    13 referencias
    public bool EstaEncendido
    {
        get
        {
            foreach (var d in dispositivos)
            {
                if (d.EstaEncendido) return true;
            }
            return false;
        }
    }

    12 referencias
    public void AgregarDispositivo(IDispositivo dispositivo)
    {
        dispositivos.Add(dispositivo);
    }

    7 referencias
    public void Encender()
    {
        Console.WriteLine();
        Console.WriteLine("Encendido grupo " + Nombre);
        foreach (var d in dispositivos)
        {
            d.Encender();
        }
    }

    7 referencias
    public void Apagar()
    {
        Console.WriteLine();
        Console.WriteLine("Apagado grupo " + Nombre);
        foreach (var d in dispositivos)
        {
            d.Apagar();
        }
    }

    7 referencias
    public IEnumerable<IDispositivo> ObtenerDispositivosPlano()
    {
        foreach (var d in dispositivos)
        {
            if (d is GrupoDispositivos g)
            {
                foreach (var sub in g.ObtenerDispositivosPlano())
                    yield return sub;
            }
            else
            {
                yield return d;
            }
        }
    }
}
```

HistorialEscenasCasa

```
public class HistorialEscenasCasa
{
    private readonly List<EscenaCasaMemento> escenas = new List<EscenaCasaMemento>();

    7 referencias
    public IReadOnlyList<EscenaCasaMemento> Escenas => escenas.AsReadOnly();

    4 referencias
    public bool TieneEscenas => escenas.Count > 0;

    1 referencia
    public void AgregarEscena(EscenaCasaMemento escena)
    {
        if (escena == null) throw new ArgumentNullException(nameof(escena));
        escenas.Add(escena);
    }

    2 referencias
    public EscenaCasaMemento ObtenerEscena(int indice)
    {
        if (indice < 0 || indice >= escenas.Count) return null;
        return escenas[indice];
    }

    1 referencia
    public void EliminarEscena(int indice)
    {
        if (indice < 0 || indice >= escenas.Count) return;
        escenas.RemoveAt(indice);
    }

    1 referencia
    public void Limpiar()
    {
        escenas.Clear();
    }
}
```

IDispositivo

```
public interface IDispositivo
{
    41 referencias
    string Nombre { get; }
    15 referencias
    bool EstaEncendido { get; }
    17 referencias
    void Encender();
    17 referencias
    void Apagar();
}
```

Program

```
namespace Presentacion
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            CentroDeControlCasa.Instancia.Iniciar();

        }
    }
}
```

Impresión

```
====   Dispositivos disponibles para decorar:   ====================

1. Televisor
2. Bocinas
3. Foco sala
4. Microondas
5. Refrigerador
6. Luces LED cuarto
7. PC escritorio
8. Foco Sanitario
9. Terminar decoracion

Seleccione una opción:
```

```
====   Control de dispositivos inteligentes   ====================

Seleccione un grupo:
1. Sala
2. Cocina
3. Cuarto
4. Baño
6. Casa
7. Salir
```

```
Seleccione acción para el grupo Casa

1. Encender
2. Apagar
3. Mostrar estado de dispositivos
4. Gestionar  guardado dee escenas
5. Regresar
```

```
====   Gestor de Guardado de casa    ====================

1. Guardar configuración actual como nueva escena
2. Listar escenas guardadas
3. Restaurar una escena
4. Eliminar una escena
5. Limpiar todas las escenas
6. Regresar
```

Conclusiones

En conclusión, el uso de los dos patrones dentro de este proyecto ,ayuda a organizar el código teniendo un entorno donde ,cada patrón agregaba un apartado de mejora, desde el patron singleton hasta ,el patron decorador,Memento,compuesto y por capas.