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 4 - 5

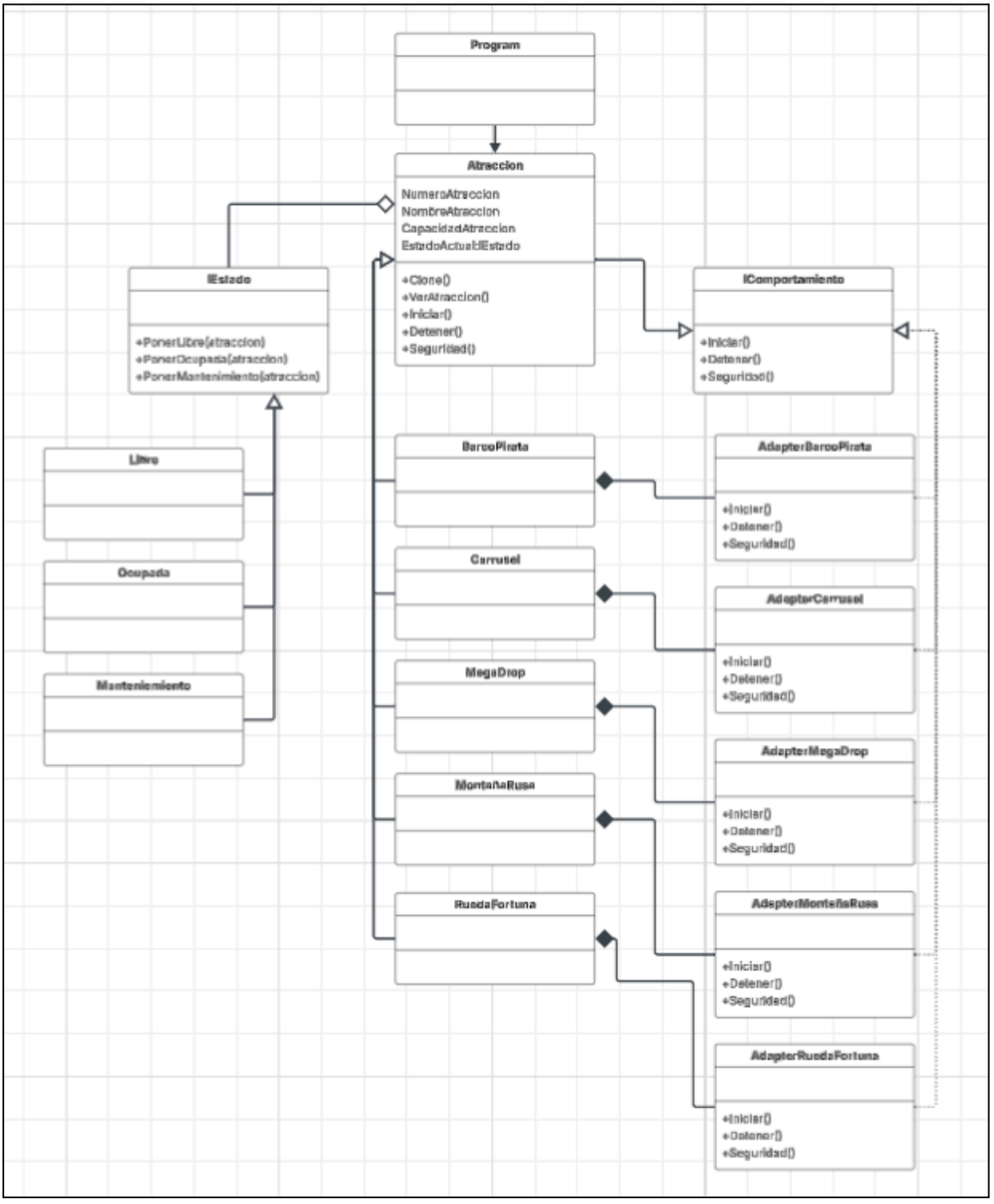
UNIDAD A EVALUAR:

Unidad 4 - 5

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:
Miguel Angel Briones Hernandez 21212325

NOMBRE DEL MAESTRO (A):
Maribel Guerrero Luis

DIAGRAMA UML



CÓDIGO

Patrón Adaptador

IComportamiento

```
namespace ExamenUnidad4y5.Modelos.Adapter
{
    public interface IComportamiento
    {
        void Iniciar();
        void Detener();
        void Seguridad();
    }
}
```

_

AdapterBarcoPirata

```
namespace ExamenUnidad4y5.Capa_de_Negocio.Adapter.Adapters
{
    internal class AdapterBarcoPirata:IComportamiento
    {
        public void Iniciar()
        {
            Console.WriteLine("Mesiendo barco pirata");
        }
        public void Detener()
        {
            Console.WriteLine("Desacelerando barco pirata.");
```

```
        }
        public void Seguridad()
        {
            Console.WriteLine("Comprobando sistema de sugeción del barco.");
        }
    }
}
```

–

AdapterCarrusel

```
namespace ExamenUnidad4y5.Modelos.Adapter.Adapters
{
    internal class AdapterCarrusel:IComportamiento
    {
        public void Iniciar()
        {
            Console.WriteLine("Iniciando giro de carrusel.");
        }
        public void Detener()
        {
            Console.WriteLine("Desacelerando carrusel");
        }
        public void Seguridad()
        {
            Console.WriteLine("Comprobando estado de sistema de giro.");
        }
```

```
    }
}
```

–

AdaoterMegaDrop

```
namespace ExamenUnidad4y5.Capa_de_Negocio.Adapter.Adapters
{
    internal class AdapterMegaDrop:IComportamiento
    {
        public void Iniciar()
        {
            Console.WriteLine("Subiendo plataforma de caída.");
        }
        public void Detener()
        {
            Console.WriteLine("Bajando plataforma de caída.");
        }
        public void Seguridad()
        {
                Console.WriteLine("Comprobando cinturones de seguridad y sistema de frenado de plataforma.");
        }
    }
}
```

–

AdapterMontañaRusa

```
namespace ExamenUnidad4y5.Modelos.Adapter
{
    internal class AdapterMontañaRusa : IComportamiento
    {
        public void Iniciar()
        {
            Console.WriteLine("Montaña rusa acelerando.");
        }
        public void Detener()
        {
                        Console.WriteLine("Montaña rusa desacelerando... Regresando a plataforma.");
        }
        public void Seguridad()
        {
            Console.WriteLine("Comprobando medidas de seguridad.");
        }
    }
}
```

–

AdapterRuedaFortuna

```
namespace ExamenUnidad4y5.Capa_de_Negocio.Adapter.Adapters
{
    internal class AdapterRuedaFortuna:IComportamiento
    {
        public void Iniciar()
        {
            Console.WriteLine("Acelerando rueda de la fortuna.");
        }
        public void Detener()
        {
            Console.WriteLine("Reduciendo velocidad de giro.");
        }
        public void Seguridad()
        {
            Console.WriteLine("Comprobando sistema de giro y puertas de cabinas");
        }
    }
}
```

–

Prototype

Atraccion

```
namespace ExamenUnidad4y5.Modelos
{
    public abstract class Atraccion : IComportamiento
    {
        protected int _numero;
        protected string _nombre;
        protected int _capacidad;

        public int NumeroAtraccion { get => _numero; set => _numero = value; }
        public string NombreAtraccion { get => _nombre; set => _nombre = value; }
         public int CapacidadAtraccion { get => _capacidad; set => _capacidad = value;
}

        public IEstado EstadoActual { get; set; }

        public Atraccion()
        {
            EstadoActual = new Libre();
        }

        public abstract Atraccion Clone();
        public abstract string VerAtraccion();
```

```
        // Adapter
        public abstract void Iniciar();
        public abstract void Detener();
        public abstract void Seguridad();
    }

}
```

—

BarcoPirata

```
namespace ExamenUnidad4y5.Capa_de_Negocio.Prototype
{
    internal class BarcoPirata:Atraccion
    {
        private IComportamiento _comportamiento;

        public BarcoPirata()
        {
            NombreAtraccion = "Barco Pirata";
            CapacidadAtraccion = 25;
            _comportamiento = new AdapterBarcoPirata();
        }

        private BarcoPirata(BarcoPirata original)
        {
```

```
            NombreAtraccion = original.NombreAtraccion;
            NumeroAtraccion = original.NumeroAtraccion;
            CapacidadAtraccion = original.CapacidadAtraccion;

            _comportamiento = new AdapterBarcoPirata();

            EstadoActual = new Libre();
        }

        public override Atraccion Clone()
        {
            return new BarcoPirata(this);
        }

        public override string VerAtraccion()
        {
                        return $"{NumeroAtraccion}. {NombreAtraccion}, capacidad:
{CapacidadAtraccion} personas";
        }
        public override void Iniciar() => _comportamiento.Iniciar();
        public override void Detener() => _comportamiento.Detener();
        public override void Seguridad() => _comportamiento.Seguridad();
    }
}
_
```

Carrusel

```
namespace ExamenUnidad4y5.Modelos.Prototype
{
    internal class Carrusel:Atraccion
    {
        private IComportamiento _comportamiento;

        public Carrusel()
        {
            NombreAtraccion = "Carrusel";
            CapacidadAtraccion = 12;
            _comportamiento = new AdapterCarrusel();
        }

        private Carrusel(Carrusel original)
        {
            NombreAtraccion = original.NombreAtraccion;
            NumeroAtraccion = original.NumeroAtraccion;
            CapacidadAtraccion = original.CapacidadAtraccion;

            _comportamiento = new AdapterCarrusel();

            EstadoActual = new Libre();
        }
```

```
        public override Atraccion Clone()
        {
            return new Carrusel(this);
        }

        public override string VerAtraccion()
        {
                        return $"{NumeroAtraccion}. {NombreAtraccion}, capacidad: {CapacidadAtraccion} personas";
        }

        public override void Iniciar() => _comportamiento.Iniciar();
        public override void Detener() => _comportamiento.Detener();
        public override void Seguridad() => _comportamiento.Seguridad();

    }
}
_
```

MegaDrop

```
namespace ExamenUnidad4y5.Capa_de_Negocio.Prototype
{
    internal class MegaDrop:Atraccion
    {
        private IComportamiento _comportamiento;

        public MegaDrop()
        {
            NombreAtraccion = "Caída libre";
            CapacidadAtraccion = 20;
            _comportamiento = new AdapterMegaDrop();
        }

        private MegaDrop(MegaDrop original)
        {
            NombreAtraccion = original.NombreAtraccion;
            NumeroAtraccion = original.NumeroAtraccion;
            CapacidadAtraccion = original.CapacidadAtraccion;

            _comportamiento = new AdapterMegaDrop();

            EstadoActual = new Libre();
        }
```

```
        public override Atraccion Clone()
        {
            return new MegaDrop(this);
        }

        public override string VerAtraccion()
        {
                        return $"{NumeroAtraccion}. {NombreAtraccion}, capacidad: {CapacidadAtraccion} personas";
        }

        public override void Iniciar() => _comportamiento.Iniciar();
        public override void Detener() => _comportamiento.Detener();
        public override void Seguridad() => _comportamiento.Seguridad();

    }

}
_
```

MontañaRusa

```
namespace ExamenUnidad4y5.Modelos.Prototype
{
    internal class MontañaRusa : Atraccion
    {
        private IComportamiento _comportamiento;

        public MontañaRusa()
        {
            NombreAtraccion = "Montaña Rusa";
            CapacidadAtraccion = 20;
            _comportamiento = new AdapterMontañaRusa();
        }

        // Constructor por copia
        private MontañaRusa(MontañaRusa original)
        {
            NombreAtraccion = original.NombreAtraccion;
            NumeroAtraccion = original.NumeroAtraccion;
            CapacidadAtraccion = original.CapacidadAtraccion;

            _comportamiento = new AdapterMontañaRusa();

            EstadoActual = new Libre();
        }
```

```
        public override Atraccion Clone()
        {
            return new MontañaRusa(this);
        }

        public override string VerAtraccion()
        {
                                return $"{NumeroAtraccion}. {NombreAtraccion}, capacidad: {CapacidadAtraccion} personas";
        }

        public override void Iniciar() => _comportamiento.Iniciar();
        public override void Detener() => _comportamiento.Detener();
        public override void Seguridad() => _comportamiento.Seguridad();
    }
}
_
```

RuedaFortuna

```
namespace ExamenUnidad4y5.Capa_de_Negocio.Prototype
{
    internal class RuedaFortuna:Atraccion
    {
        private IComportamiento _comportamiento;
```

```
public RuedaFortuna()
{
    NombreAtraccion = "Rueda de la Fortuna";
    CapacidadAtraccion = 50;
    _comportamiento = new AdapterCarrusel();
}

public RuedaFortuna(RuedaFortuna original)
{
    NombreAtraccion = original.NombreAtraccion;
    NumeroAtraccion = original.NumeroAtraccion;
    CapacidadAtraccion = original.CapacidadAtraccion;

    _comportamiento = new AdapterRuedaFortuna();

    EstadoActual = new Libre();
}

public override Atraccion Clone()
{
    return new RuedaFortuna(this);
}

public override string VerAtraccion()
```

```
        {
                        return $"{NumeroAtraccion}. {NombreAtraccion}, capacidad: {CapacidadAtraccion} personas";
        }

        public override void Iniciar() => _comportamiento.Iniciar();
        public override void Detener() => _comportamiento.Detener();
        public override void Seguridad() => _comportamiento.Seguridad();

    }
}
```

_

State

IEstado

```
namespace ExamenUnidad4y5.Modelos.State
{
    public interface IEstado
    {
        void PonerLibre(Atraccion atraccion);
        void PonerOcupada(Atraccion atraccion);
        void PonerMantenimiento(Atraccion atraccion);
    }

}
```

_

Libre

```
namespace ExamenUnidad4y5.Modelos.State
{
    public class Libre : IEstado
    {
        public void PonerLibre(Atraccion atraccion)
        {
            Console.WriteLine("La atracción ya está libre.");
        }

        public void PonerOcupada(Atraccion atraccion)
        {
            Console.WriteLine("La atracción pasa a estar ocupada.");
            atraccion.EstadoActual = new Ocupada();
        }

        public void PonerMantenimiento(Atraccion atraccion)
        {
            Console.WriteLine("La atracción se pone en mantenimiento.");
            atraccion.EstadoActual = new Mantenimiento();
        }
    }

}
```

–

Mantenimiento

```
namespace ExamenUnidad4y5.Modelos.State
{
    public class Mantenimiento : IEstado
    {
        public void PonerLibre(Atraccion atraccion)
        {
            Console.WriteLine("Atracción lista, pasa a estado libre.");
            atraccion.EstadoActual = new Libre();
        }

        public void PonerOcupada(Atraccion atraccion)
        {
            Console.WriteLine("No se puede ocupar: está en mantenimiento.");
        }

        public void PonerMantenimiento(Atraccion atraccion)
        {
            Console.WriteLine("La atracción ya está en mantenimiento.");
        }
    }
}
```

–

Ocupada

```
namespace ExamenUnidad4y5.Modelos.State
{
    public class Ocupada : IEstado
    {
        public void PonerLibre(Atraccion atraccion)
        {
            Console.WriteLine("La atracción ahora está libre.");
            atraccion.EstadoActual = new Libre();
        }

        public void PonerOcupada(Atraccion atraccion)
        {
            Console.WriteLine("La atracción ya está ocupada.");
        }

        public void PonerMantenimiento(Atraccion atraccion)
        {
                Console.WriteLine("No se puede poner en mantenimiento mientras está ocupada.");
        }
    }
}
```

–

Program.cs

```
namespace ExamenUnidad4y5
{
    internal class Program
    {
        static void Main(string[] args)
        {
            MontañaRusa plantillaMontaña = new MontañaRusa();
            Carrusel plantillaCarrusel = new Carrusel();
            RuedaFortuna plantillaRueda = new RuedaFortuna();
            BarcoPirata plantillaBarco = new BarcoPirata();
            MegaDrop plantillaM = new MegaDrop();

            // Clones únicos
                                    Atraccion montañaRusa = plantillaMontaña.Clone();
montañaRusa.NumeroAtraccion = 1;
            Atraccion carrusel = plantillaCarrusel.Clone(); carrusel.NumeroAtraccion = 2;
            Atraccion ruedaF = plantillaRueda.Clone(); ruedaF.NumeroAtraccion = 3;
            Atraccion barco = plantillaBarco.Clone(); barco.NumeroAtraccion = 4;
            Atraccion mega = plantillaM.Clone(); mega.NumeroAtraccion = 5;

            List<Atraccion> atracciones = new List<Atraccion>()
            {
                montañaRusa, carrusel, ruedaF, barco, mega
            };
```

```
    MenuPrincipal(atracciones);
}

// Dinero global del parque
static decimal dineroTotal = 0;
static decimal precioUso = 15;

static void MenuPrincipal(List<Atraccion> atracciones)
{
    string opcion = "";

    while (opcion != "0")
    {
        Console.Clear();
        Console.WriteLine("=== PARQUE DE ATRACCIONES ===\n");

        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine($"Ingresos totales del parque: ${dineroTotal}\n");
        Console.ResetColor();

        // Mostrar lista de atracciones
        foreach (var atr in atracciones)
        {
            string estado = atr.EstadoActual.GetType().Name;
```

```
            switch (estado)
            {
                case "Libre":
                    Console.ForegroundColor = ConsoleColor.Green;
                    break;
                case "Ocupada":
                    Console.ForegroundColor = ConsoleColor.DarkYellow;
                    break;
                case "Mantenimiento":
                    Console.ForegroundColor = ConsoleColor.Red;
                    break;
            }

                Console.WriteLine($"{atr.NumeroAtraccion}. {atr.NombreAtraccion} -
Estado: {estado}");
            Console.ResetColor();
        }

        Console.Write("\nSeleccione una atracción (0 para salir): ");
        opcion = Console.ReadLine();

        if (opcion == "0")
            break;
```

```
            if (int.TryParse(opcion, out int num))
            {
                var seleccionada = atracciones.FirstOrDefault(a => a.NumeroAtraccion
== num);

                if (seleccionada != null)
                    MenuAtraccion(seleccionada);
                else
                {
                    Console.WriteLine("Atracción no encontrada.");
                    Console.ReadKey();
                }
            }
        }

        Console.WriteLine("Saliendo del sistema...");
    }

    static void MenuAtraccion(Atraccion atr)
    {
        string opcion = "";

        while (opcion != "0")
        {
            Console.Clear();
```

```
            Console.WriteLine($"=== CONTROL DE {atr.NombreAtraccion.ToUpper()}
===\n");

            Console.WriteLine(atr.VerAtraccion());
            Console.WriteLine($"Estado actual: {atr.EstadoActual.GetType().Name}");
            Console.WriteLine($"Dinero del parque: {dineroTotal}\n");

            Console.WriteLine("1. Operación (Iniciar / Detener / Seguridad)");
                        Console.WriteLine("2. Cambiar Estado (Libre / Ocupada /
Mantenimiento)");
            Console.WriteLine("0. Volver al menú principal");
            Console.Write("\nSeleccione una opción: ");

            opcion = Console.ReadLine();

            switch (opcion)
            {
                case "1":
                    SubMenuOperacion(atr);
                    break;

                case "2":
                    SubMenuEstados(atr);
                    break;

                case "0":
```

```
                return;

            default:
                Console.WriteLine("Opción no válida.");
                break;
        }
    }
}

static void SubMenuOperacion(Atraccion atr)
{
    string opcion = "";

    while (opcion != "0")
    {
        Console.Clear();
        Console.WriteLine($"--- OPERACIÓN: {atr.NombreAtraccion} ---\n");

        Console.WriteLine("1. Iniciar (marca Ocupada y genera dinero)");
        Console.WriteLine("2. Detener (marca Libre)");
        Console.WriteLine("3. Seguridad");
        Console.WriteLine("0. Volver\n");
        Console.Write("Opción: ");

        opcion = Console.ReadLine();
```

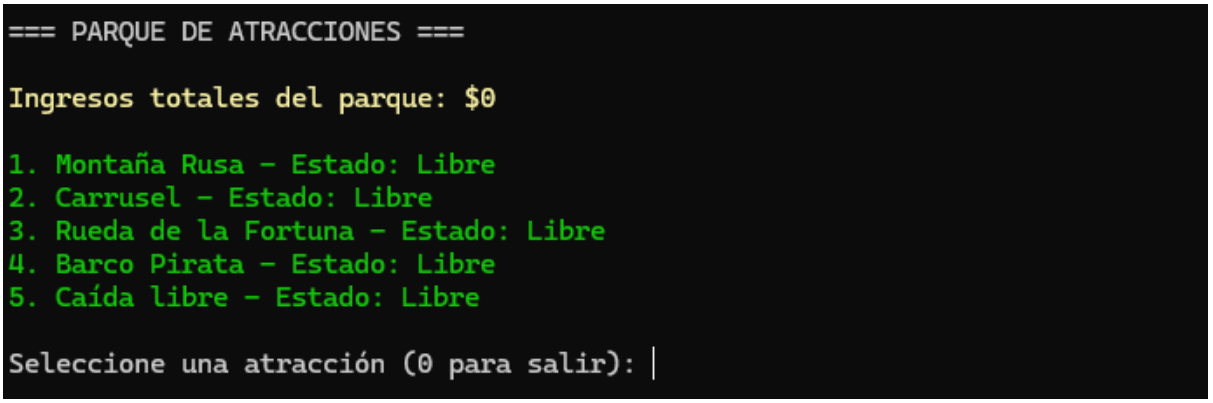
```
switch (opcion)
{
    case "1":
        atr.Iniciar();
        atr.EstadoActual.PonerOcupada(atr);
        GenerarIngreso();
        break;

    case "2":
        atr.Detener();
        atr.EstadoActual.PonerLibre(atr);
        break;

    case "3":
        atr.Seguridad();
        break;

    case "0":
        return;

    default:
        Console.WriteLine("Opción inválida.");
        break;
}
```

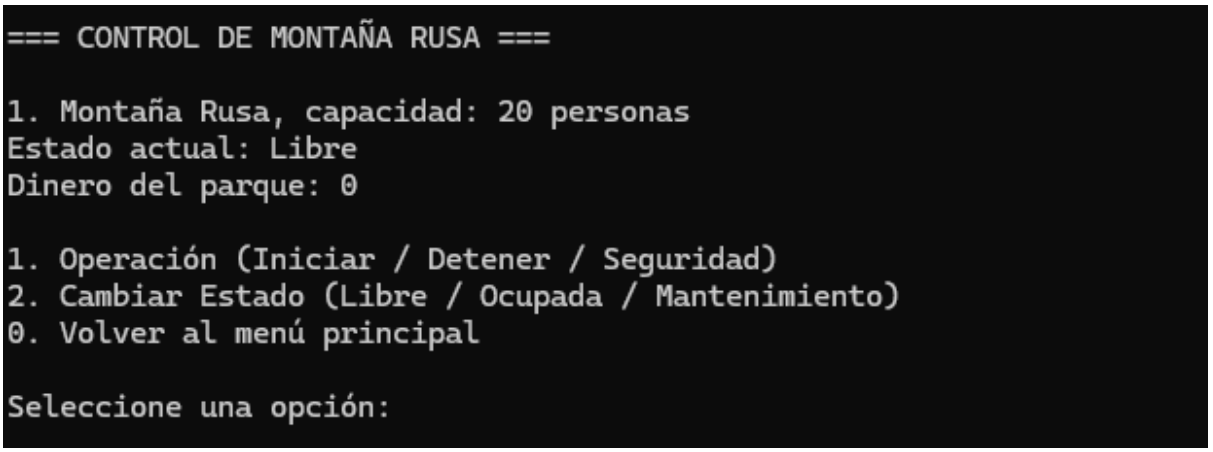
```
            Console.WriteLine("\nPresione cualquier tecla...");
            Console.ReadKey();
        }
    }

    static void SubMenuEstados(Atraccion atr)
    {
        string opcion = "";

        while (opcion != "0")
        {
            Console.Clear();
            Console.WriteLine($"--- ESTADOS: {atr.NombreAtraccion} ---\n");

            Console.WriteLine("1. Poner Libre");
            Console.WriteLine("2. Poner Ocupada");
            Console.WriteLine("3. Poner Mantenimiento");
            Console.WriteLine("0. Volver\n");
            Console.Write("Opción: ");

            opcion = Console.ReadLine();

            switch (opcion)
            {
```

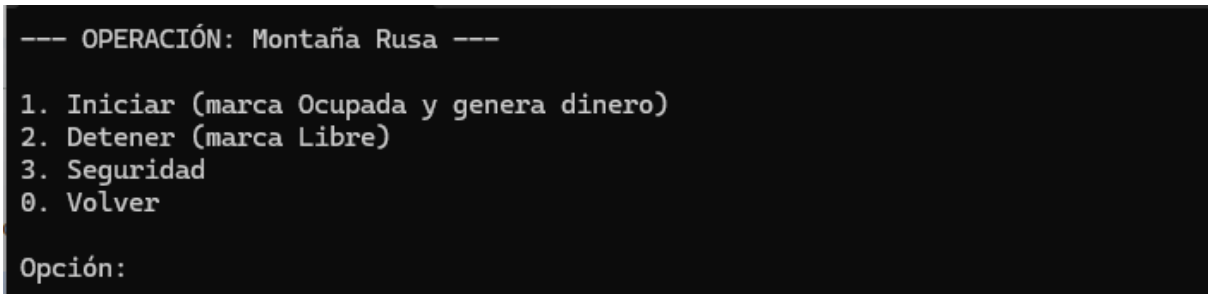
```
            case "1":
                atr.EstadoActual.PonerLibre(atr);
                break;

            case "2":
                atr.EstadoActual.PonerOcupada(atr);
                GenerarIngreso();
                break;

            case "3":
                atr.EstadoActual.PonerMantenimiento(atr);
                break;

            case "0":
                return;

            default:
                Console.WriteLine("Opción inválida.");
                break;
        }

        Console.WriteLine("\nPresione cualquier tecla...");
        Console.ReadKey();
    }
}
```

```
        static void GenerarIngreso()
        {
            dineroTotal += precioUso;
        }
    }
}
```

CAPTURAS







```
--- ESTADOS: Montaña Rusa ---

1. Poner Libre
2. Poner Ocupada
3. Poner Mantenimiento
0. Volver

Opción:
```

```
--- OPERACIÓN: Montaña Rusa ---

1. Iniciar (marca Ocupada y genera dinero)
2. Detener (marca Libre)
3. Seguridad
0. Volver

Opción: 1
Montaña rusa acelerando.
La atracción pasa a estar ocupada.

Presione cualquier tecla...
```

```
=== PARQUE DE ATRACCIONES ===

Ingresos totales del parque: $15

1. Montaña Rusa - Estado: Ocupada
2. Carrusel - Estado: Libre
3. Rueda de la Fortuna - Estado: Libre
4. Barco Pirata - Estado: Libre
5. Caída libre - Estado: Libre

Seleccione una atracción (0 para salir):
```

```
Opción: 2
Montaña rusa desacelerando... Regresando a plataforma.
La atracción ahora está libre.

Presione cualquier tecla...
```

```
Opción: 3
Comprobando medidas de seguridad.

Presione cualquier tecla...
```

CONCLUSIÓN

El presente proyecto de programación busca la integración de diferentes patrones de diseño para la resolución de un problema de manera más eficiente. La problemática gira en torno a un parque de diversiones el cual requiere administrar sus atracciones así como los ingresos que llegan al mismo por el uso de las atracciones, por lo cual por medio del uso de patrones de diseño se busca crear una aplicación que resuelva esto.

El programa incorpora 4 patrones diferentes, por un lado se incorpora el patrón prototype, con este lo que se busca es crear una plantilla general para cada una de las atracciones del parque que se pueda clonar, esto debido a que todas las atracciones comparten ciertas características en común. Por otro lado se incorporó el patrón State, por medio de este lo que se hizo fue establecer 3 estados en los cuales todas las atracciones pueden estar, de este modo dentro del sistema se puede interpretar que una atracción se encuentra ocupada, libre o en mantenimiento. Además se incorporó el patrón Adapter, con este se definen algunos comportamientos comunes que hacen todas las atracciones pero variando en cada uno de los clones de la plantilla, es decir, en cada una de las atracciones. Por último se incorporó un patrón de arquitectura en capas, el cual permite tener una mejor organización del proyecto en su totalidad de modo que no sea un conjunto desordenado de clases, de esta forma el proyecto se puede dividir en capa de presentación y de negocio en la cual se hallan todos los patrones.

Para concluir, este proyecto hace una integración de diferentes patrones como parte de una propuesta para resolver un problema, esta clase de enfoques permiten la creación de sistemas y aplicaciones más eficientes y ordenadas que cumplen de la mejor manera su propósito. Se puede destacar que una buena implementación y combinación de estos patrones permite diseñar mejores soluciones a diferentes problemas en diferentes contextos como lo puede ser la programación web, móvil o de escritorio.