TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto-Diciembre 2025

CARRERA:
INGENIERÍA EN SISTEMAS COMPUTACIONALES

MATERIA:
Patrones de diseño

TÍTULO ACTIVIDAD :
Examen unidad 4 y 5

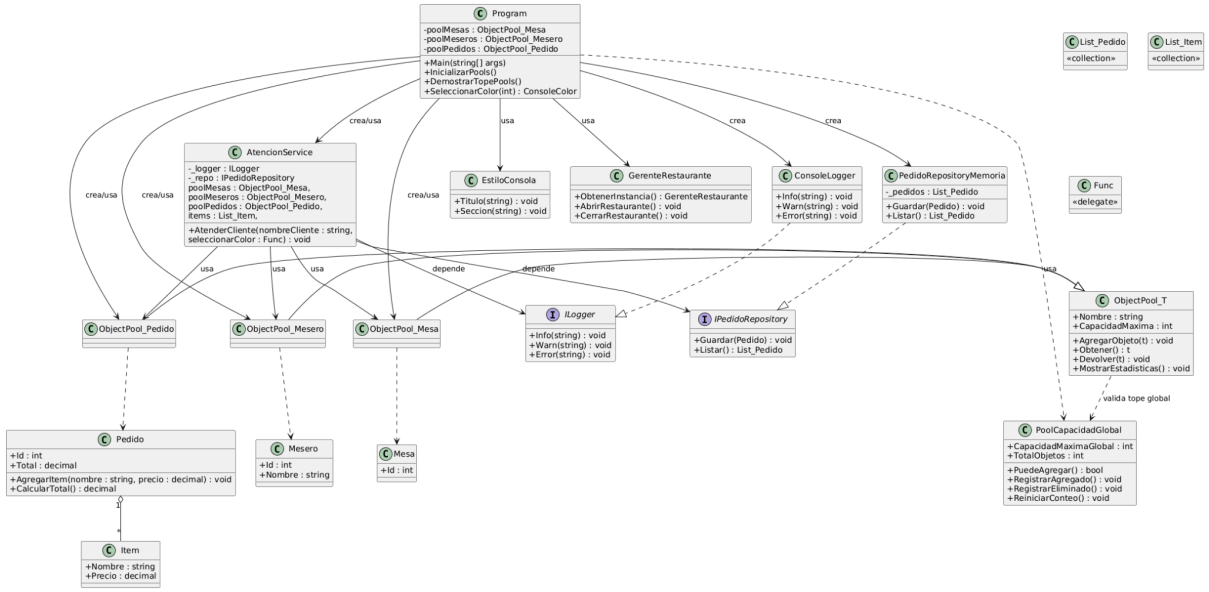
UNIDAD 4, 5

NOMBRE Y NÚMERO DE CONTROL:
NARVAEZ MATA ALEJANDRO- 22210325

NOMBRE DEL MAESTRO (A):

MARIBEL GUERRERO LUIS

UML


Program
-poolMesas : ObjectPool_Mesa
-poolMeseros : ObjectPool_Mesero
-poolPedidos : ObjectPool_Pedido
+Main(string[] args)
+InicializarPools()
+DemostrarTopePools()
+SeleccionarColor(int) : ConsoleColor
List_Pedido
«collection»
List_Item
«collection»
crea/usa
usa
usa
crea
crea
AtencionService
-_logger : ILogger
-_repo : IPedidoRepository
poolMesas : ObjectPool_Mesa,
poolMeseros : ObjectPool_Mesero,
poolPedidos : ObjectPool_Pedido,
items : List_Item,
+AtenderCliente(nombreCliente : string, seleccionarColor : Func) : void
EstiloConsola
+Titulo(string) : void
+Seccion(string) : void
GerenteRestaurante
+ObtenerInstancia() : GerenteRestaurante
+AbrirRestaurante() : void
+CerrarRestaurante() : void
ConsoleLogger
+Info(string) : void
+Warn(string) : void
+Error(string) : void
PedidoRepositoryMemoria
-_pedidos : List_Pedido
+Guardar(Pedido) : void
+Listar() : List_Pedido
Func
«delegate»
crea/usa
crea/usa
crea/usa
usa
usa
usa
depende
depende
usa
ObjectPool_Pedido
ObjectPool_Mesero
ObjectPool_Mesa
ILogger
+Info(string) : void
+Warn(string) : void
+Error(string) : void
IPedidoRepository
+Guardar(Pedido) : void
+Listar() : List_Pedido
ObjectPool_T
+Nombre : string
+CapacidadMaxima : int
+AgregarObjeto(t) : void
+Obtener() : t
+Devolver(t) : void
+MostrarEstadisticas() : void
valida tope global
Pedido
+Id : int
+Total : decimal
+AgregarItem(nombre : string, precio : decimal) : void
+CalcularTotal() : decimal
Mesero
+Id : int
+Nombre : string
Mesa
+Id : int
PoolCapacidadGlobal
+CapacidadMaximaGlobal : int
+TotalObjetos : int
+PuedeAgregar() : bool
+RegistrarAgregado() : void
+RegistrarEliminado() : void
+ReiniciarConteo() : void
1
*
Item
+Nombre : string
+Precio : decimal

Program.cs

```
using System;
using System.Threading;
using RestauranteSimulacion.Infrastructure;
using RestauranteSimulacion.Application;

namespace RestauranteSimulacion
{
    class Program
    {
        static ObjectPool<Mesa> poolMesas = null!;
        static ObjectPool<Mesero> poolMeseros = null!;
        static ObjectPool<Pedido> poolPedidos = null!;

        static void Main(string[] args)
        {
            EstiloConsola.Titulo("Simulación de Restaurante");
            var gerente = GerenteRestaurante.ObtenerInstancia();
            gerente.AbrirRestaurante();

            InicializarPools();

            DemostrarTopePools();
            PoolCapacidadGlobal.ReiniciarConteo();
            InicializarPools();
```

```
            var logger = new ConsoleLogger();
            var pedidoRepo = new PedidoRepositoryMemoria();
            var atencion = new AtencionService(logger, pedidoRepo);

            atencion.AtenderCliente("Frenkie de Jong", poolMesas, poolMeseros, poolPedidos,
new[] { ("Hamburguesa", 150m), (" Bacardi con coca ", 30m) }, SeleccionarColor);

            EstiloConsola.Seccion("Tiempo de Espera");
            Console.WriteLine("[TIEMPO] Esperando 30 segundos para el siguiente cliente...");
            Thread.Sleep(30000);

            atencion.AtenderCliente("Aitana Bombati", poolMesas, poolMeseros, poolPedidos,
new[] { ("Pizza", 200m) }, SeleccionarColor);

            poolMesas.MostrarEstadisticas();
            poolMeseros.MostrarEstadisticas();
            poolPedidos.MostrarEstadisticas();

            gerente.CerrarRestaurante();
            Console.ReadKey();
        }

        static void InicializarPools()
        {
            poolMesas = new ObjectPool<Mesa>("Mesas", capacidadMaxima: 3);
            poolMesas.AgregarObjeto(new Mesa(1));
```

```
            poolMesas.AgregarObjeto(new Mesa(2));
            poolMesas.AgregarObjeto(new Mesa(3));
            poolMesas.AgregarObjeto(new Mesa(99));

            poolMeseros = new ObjectPool<Mesero>("Meseros", capacidadMaxima: 2);
            poolMeseros.AgregarObjeto(new Mesero(1, "Gilberto"));
            poolMeseros.AgregarObjeto(new Mesero(2, "Marcela"));
            poolMeseros.AgregarObjeto(new Mesero(99, "Extra"));

            poolPedidos = new ObjectPool<Pedido>("Pedidos", capacidadMaxima: 2);
            poolPedidos.AgregarObjeto(new Pedido(1));
            poolPedidos.AgregarObjeto(new Pedido(2));
            poolPedidos.AgregarObjeto(new Pedido(99));
        }


        static ConsoleColor SeleccionarColor(int meseroId)
        {
            var colores = new[] { ConsoleColor.Green, ConsoleColor.Yellow, ConsoleColor.Cyan,
ConsoleColor.Magenta, ConsoleColor.Blue };
            int idx = (meseroId - 1) % colores.Length;
            return colores[idx];
        }

        static void DemostrarTopePools()
        {
            EstiloConsola.Seccion("Demostración de Tope de Capacidad");
```

```
for (int i = 4; i <= 10; i++)
{
    poolMesas.AgregarObjeto(new Mesa(i));
}
poolMesas.AgregarObjeto(new Mesa(100));
poolMesas.AgregarObjeto(new Mesa(101));

for (int i = 3; i <= 10; i++)
{
    poolMeseros.AgregarObjeto(new Mesero(i, $"Mesero #{i}"));
}
poolMeseros.AgregarObjeto(new Mesero(100, "Mesero extra 1"));
poolMeseros.AgregarObjeto(new Mesero(101, "Mesero extra 2"));

for (int i = 3; i <= 10; i++)
{
    poolPedidos.AgregarObjeto(new Pedido(i));
}
poolPedidos.AgregarObjeto(new Pedido(100));
poolPedidos.AgregarObjeto(new Pedido(101));

EstiloConsola.Seccion("Prueba de extracción hasta vaciar");
for (int i = 0; i < 12; i++)
{
    var m = poolMesas.Obtener();
    if (m == null) break;
```

```
            }
            for (int i = 0; i < 12; i++)
            {
                var me = poolMeseros.Obtener();
                if (me == null) break;
            }
            for (int i = 0; i < 12; i++)
            {
                var p = poolPedidos.Obtener();
                if (p == null) break;
            }

            poolMesas.MostrarEstadisticas();
            poolMeseros.MostrarEstadisticas();
            poolPedidos.MostrarEstadisticas();

            EstiloConsola.Seccion("Recuperación tras prueba");
            poolMesas.Devolver(new Mesa(-1));
        }
    }
}
```

ObjectPool.cs

```
using System;
```

```
using System.Collections.Generic;


namespace RestauranteSimulacion
{
    public class ObjectPool<T> where T : class
    {
        private readonly List<T> _disponibles;
        private readonly List<T> _enUso;
        public int CapacidadMaxima { get; private set; }
        public string NombrePool { get; private set; }


        public ObjectPool(string nombrePool)
        {
            NombrePool = nombrePool;
            _disponibles = new List<T>();
            _enUso = new List<T>();
            CapacidadMaxima = int.MaxValue;
            Console.WriteLine($"[POOL] Pool '{nombrePool}' creado");
        }


        public ObjectPool(string nombrePool, int capacidadMaxima)
        {
            NombrePool = nombrePool;
            CapacidadMaxima = capacidadMaxima > 0 ? capacidadMaxima : throw new
ArgumentOutOfRangeException(nameof(capacidadMaxima), "La capacidad debe ser mayor
a cero.");
            _disponibles = new List<T>();
```

```
        _enUso = new List<T>();

        Console.WriteLine($"[POOL] Pool '{nombrePool}' creado (Tope local: {CapacidadMaxima})");

    }



    public void AgregarObjeto(T objeto)

    {

        int totalLocal = _disponibles.Count + _enUso.Count;

        if (totalLocal >= CapacidadMaxima)

        {

            Console.WriteLine($"[POOL-LIMITE-LOCAL] ✗ No se puede agregar al pool '{NombrePool}': tope local {CapacidadMaxima} alcanzado (Total local: {totalLocal})");

            return;

        }

        if (!PoolCapacidadGlobal.PuedeAgregar())

        {

            Console.WriteLine($"[POOL-LIMITE-GLOBAL] ✗ No se puede agregar: capacidad global {PoolCapacidadGlobal.CapacidadMaximaGlobal} alcanzada (Total global: {PoolCapacidadGlobal.TotalObjetos})");

            return;

        }

        PoolCapacidadGlobal.RegistrarAgregado();

        _disponibles.Add(objeto);

        string topeLocalTxt = CapacidadMaxima == int.MaxValue ? "sin tope" : CapacidadMaxima.ToString();

        Console.WriteLine($"[POOL-PUSH] ✓ Objeto agregado al pool '{NombrePool}' (Disponibles: {_disponibles.Count} | En uso: {_enUso.Count} | Tope local: {topeLocalTxt} | Global: {PoolCapacidadGlobal.TotalObjetos}/{PoolCapacidadGlobal.CapacidadMaximaGlobal})");

    }
```

```
    public T Obtener()
    {
        if (_disponibles.Count > 0)
        {
            T objeto = _disponibles[0];
            _disponibles.RemoveAt(0);
            _enUso.Add(objeto);
            string topeLocalTxt = CapacidadMaxima == int.MaxValue ? "sin tope" :
CapacidadMaxima.ToString();
            Console.WriteLine($"[POOL-POP] ← Objeto SACADO del pool '{NombrePool}'
(Disponibles: {_disponibles.Count} | En uso: {_enUso.Count} | Tope local: {topeLocalTxt} |
Global:
{PoolCapacidadGlobal.TotalObjetos}/{PoolCapacidadGlobal.CapacidadMaximaGlobal})");
            return objeto;
        }
        Console.WriteLine($"[POOL-ERROR] ✗ Pool '{NombrePool}' vacío - No hay objetos
disponibles");
        return null!;
    }

    public void Devolver(T objeto)
    {
        if (objeto == null) return;

        if (_enUso.Remove(objeto))
        {
            ReiniciarObjeto(objeto);
            _disponibles.Add(objeto);
```

```
            string topeLocalTxt = CapacidadMaxima == int.MaxValue ? "sin tope" :
CapacidadMaxima.ToString();

            Console.WriteLine($"[POOL-RETURN] → Objeto DEVUELTO al pool
'{NombrePool}' (Disponibles: {_disponibles.Count} | En uso: {_enUso.Count} | Tope local:
{topeLocalTxt} | Global:
{PoolCapacidadGlobal.TotalObjetos}/{PoolCapacidadGlobal.CapacidadMaximaGlobal})");

        }

    }



    private void ReiniciarObjeto(T objeto)

    {

        if (objeto is Mesa mesa) mesa.Reiniciar();

        else if (objeto is Mesero mesero) mesero.Reiniciar();

        else if (objeto is Pedido pedido) pedido.Reiniciar();

    }



    public void MostrarEstadisticas()

    {

        string topeLocalTxtFinal = CapacidadMaxima == int.MaxValue ? "sin tope" :
CapacidadMaxima.ToString();

        Console.WriteLine($"\n[{NombrePool}] Disponibles: {_disponibles.Count} | En uso:
{_enUso.Count} | Tope local: {topeLocalTxtFinal} | Global:
{PoolCapacidadGlobal.TotalObjetos}/{PoolCapacidadGlobal.CapacidadMaximaGlobal}");

    }

  }

}
```

Mesa.cs

```
using System;

namespace RestauranteSimulacion
{
    public class Mesa
    {
        public int NumeroMesa { get; private set; }
        public bool EstaOcupada { get; private set; }

        public Mesa(int numeroMesa)
        {
            NumeroMesa = numeroMesa;
            EstaOcupada = false;
        }

        public void Ocupar(string cliente)
        {
            EstaOcupada = true;
            Console.WriteLine($"  [MESA] Mesa {NumeroMesa} → OCUPADA por {cliente}");
        }

        public void Reiniciar()
        {
            EstaOcupada = false;
            Console.WriteLine($"  [MESA] Mesa {NumeroMesa} → LIMPIADA y lista para
reusar");
        }
```

```
    }
}
```

Mesero.cs

```
using System;

namespace RestauranteSimulacion
{
    public class Mesero
    {
        public int Id { get; }
        public string Nombre { get; private set; }
        private IFormateadorMesero _formateador = new FormateadorMeseroBase();

        public Mesero(int id, string nombre)
        {
            Id = id;
            Nombre = nombre;
        }

        public void AsignarCliente(string cliente)
        {
            System.Console.Write("  [MESERO] ");
            EscribirNombre();
```

```
            System.Console.WriteLine($" → ATENDIENDO a {cliente}");
        }

        public void Reiniciar()
        {
            System.Console.Write("  [MESERO] ");
            EscribirNombre();
            System.Console.WriteLine(" → DISPONIBLE y listo para reusar");
        }

        public void ConfigurarFormateador(IFormateadorMesero formateador)
        {
            _formateador = formateador ?? new FormateadorMeseroBase();
        }

        internal void EscribirNombre()
        {
            _formateador.EscribirNombre(this);
        }
    }
}
```

Pedido.cs

```
using System;
```

```
using System.Collections.Generic;


namespace RestauranteSimulacion
{
    public class Pedido
    {
        public int NumeroPedido { get; private set; }
        public decimal Total { get; private set; }


        public Pedido(int numeroPedido)
        {
            NumeroPedido = numeroPedido;
            Total = 0;
        }


        public void IniciarPedido(string cliente)
        {
            Console.WriteLine($"  [PEDIDO] Pedido {NumeroPedido} → INICIADO para
{cliente}");
        }


        public void AgregarItem(string item, decimal precio)
        {
            Total += precio;
            Console.WriteLine($"    + {item} (${precio})");
        }
```

```
        public void FinalizarPedido()
        {
            Console.WriteLine($"  [PEDIDO] → FINALIZADO - Total: ${Total}");
        }

        public void Reiniciar()
        {
            Total = 0;
            Console.WriteLine($"  [PEDIDO] Pedido {NumeroPedido} → LIMPIADO y listo para reusar");
        }
    }
}
```

EstiloConsola.cs

```
using System;

namespace RestauranteSimulacion
{
    public static class EstiloConsola
    {
        private const int Ancho = 60;

        public static void Linea(char ch = '=')
        {
```

```
    Console.WriteLine(new string(ch, Ancho));
}

public static void Titulo(string texto)
{
    Linea('=');
    Console.WriteLine(Centrar(texto, Ancho));
    Linea('=');
}

public static void Seccion(string texto)
{
    Console.WriteLine();
    Console.WriteLine(Centrar($"{texto}", Ancho));
    Linea('-');
}

public static void Etiqueta(string etiqueta, string valor)
{
    Console.WriteLine($"{etiqueta}: {valor}");
}

public static void ConColor(ConsoleColor color, Action accion)
{
    var previo = Console.ForegroundColor;
    Console.ForegroundColor = color;
```

```
            try { accion(); }
            finally { Console.ForegroundColor = previo; }
        }

        private static string Centrar(string texto, int ancho)
        {
            if (string.IsNullOrEmpty(texto)) return string.Empty;
            if (texto.Length >= ancho) return texto;
            int padding = (ancho - texto.Length) / 2;
            return new string(' ', padding) + texto;
        }
    }
}
```

GerenteRestaurante.cs

```
using System;

namespace RestauranteSimulacion
{
    public sealed class GerenteRestaurante
    {
        private static GerenteRestaurante _instancia = null!;
        private static readonly object _lock = new object();
```

```
    public int ClientesAtendidos { get; private set; }


    private GerenteRestaurante()
    {
        ClientesAtendidos = 0;
        Console.WriteLine("\n[SINGLETON]  INSTANCIA ÚNICA CREADA - Gerente del
restaurante");
    }


    public static GerenteRestaurante ObtenerInstancia()
    {
        if (_instancia == null)
        {
            lock (_lock)
            {
                if (_instancia == null)
                {
                    _instancia = new GerenteRestaurante();
                }
            }
        }
        else
        {
            Console.WriteLine("  [SINGLETON] → Reutilizando instancia única existente");
        }
        return _instancia;
    }
```

```
    public void AbrirRestaurante()

    {

        Console.WriteLine("\n[RESTAURANTE] El gerente ha abierto el restaurante\n");

    }



    public void CerrarRestaurante()

    {

        Console.WriteLine($"\n[RESTAURANTE] El gerente ha cerrado el restaurante -
Clientes atendidos: {ClientesAtendidos}\n");

    }



    public void RecibirCliente(string nombreCliente)

    {

        Console.WriteLine($"  [SINGLETON-HOST] Gerente recibe a {nombreCliente} en la
entrada");

    }



    public void AsignarMesa(string nombreCliente, int numeroMesa)

    {

        Console.WriteLine($"  [SINGLETON-HOST] Gerente asigna Mesa {numeroMesa} a
{nombreCliente}");

    }



    public void RegistrarClienteAtendido()

    {

        ClientesAtendidos++;
```

```
            Console.WriteLine($"  [SINGLETON-HOST] Gerente registra cliente
#{ClientesAtendidos} atendido");
        }

        public void DespedirCliente(string nombreCliente)
        {
            Console.WriteLine($"  [SINGLETON-HOST] Gerente despide a {nombreCliente}");
        }
    }
}
```

AtencionService.cs

```
using System;
using System.Threading;

namespace RestauranteSimulacion.Application
{
    using RestauranteSimulacion.Infrastructure;

    public class AtencionService
    {
        private readonly ILogger _logger;
        private readonly IPedidoRepository _pedidoRepo;

        public AtencionService(ILogger logger, IPedidoRepository pedidoRepo)
```

```
{
    _logger = logger;
    _pedidoRepo = pedidoRepo;
}

public void AtenderCliente(
    string nombreCliente,
    ObjectPool<Mesa> poolMesas,
    ObjectPool<Mesero> poolMeseros,
    ObjectPool<Pedido> poolPedidos,
    (string item, decimal precio)[] items,
    Func<int, ConsoleColor> seleccionarColor)
{
    var gerente = GerenteRestaurante.ObtenerInstancia();
    var cliente = new Cliente(nombreCliente);

    EstiloConsola.Seccion($"Cliente: {nombreCliente}");
    cliente.CambiarEstado(ClienteEstado.Esperando);
    Thread.Sleep(TimeSpan.FromSeconds(15));

    var mesa = poolMesas.Obtener();
    var mesero = poolMeseros.Obtener();
    var pedido = poolPedidos.Obtener();

    gerente.RecibirCliente(nombreCliente);
    gerente.AsignarMesa(nombreCliente, mesa.NumeroMesa);
```

```
var color = seleccionarColor(mesero.Id);
var formateador = new ColorFormateador(color, new FormateadorMeseroBase());
mesero.ConfigurarFormateador(formateador);
mesero.AsignarCliente(nombreCliente);

pedido.IniciarPedido(nombreCliente);
foreach (var (item, precio) in items)
{
    pedido.AgregarItem(item, precio);
}

cliente.CambiarEstado(ClienteEstado.Atendido);
Thread.Sleep(TimeSpan.FromSeconds(15));

pedido.FinalizarPedido();
_logger.Info($"Pago total: ${pedido.Total}");
_pedidoRepo.Guardar(pedido);

cliente.CambiarEstado(ClienteEstado.Finalizado);
gerente.RegistrarClienteAtendido();
gerente.DespedirCliente(nombreCliente);

poolMesas.Devolver(mesa);
poolMeseros.Devolver(mesero);
poolPedidos.Devolver(pedido);
```

```
        }
    }
}
```

ConsoleLogger.cs

```
using System;

namespace RestauranteSimulacion.Infrastructure
{
    public class ConsoleLogger : ILogger
    {
        public void Info(string mensaje) => Console.WriteLine(mensaje);
        public void Warn(string mensaje) => Console.WriteLine(mensaje);
        public void Error(string mensaje) => Console.WriteLine(mensaje);
    }
}
```

ILogger.cs

```
namespace RestauranteSimulacion.Infrastructure
{
    public interface ILogger
    {
```

```
        void Info(string mensaje);
        void Warn(string mensaje);
        void Error(string mensaje);
    }
}
```

IPedidoRepository.cs

```
using System.Collections.Generic;

namespace RestauranteSimulacion.Infrastructure
{
    public interface IPedidoRepository
    {
        void Guardar(Pedido pedido);
        IEnumerable<Pedido> ObtenerTodos();
        void Limpiar();
    }
}
```

PedidoRepositoryMemoria.cs

```
using System.Collections.Generic;
```

```
namespace RestauranteSimulacion.Infrastructure
{
    public class PedidoRepositoryMemoria : IPedidoRepository
    {
        private readonly List<Pedido> _pedidos = new();

        public void Guardar(Pedido pedido)
        {
            _pedidos.Add(pedido);
        }

        public IEnumerable<Pedido> ObtenerTodos()
        {
            return _pedidos;
        }

        public void Limpiar()
        {
            _pedidos.Clear();
        }
    }
}
```

Cliente.cs

```
using System;

namespace RestauranteSimulacion
{
    public enum ClienteEstado
    {
        Esperando,
        Atendido,
        Finalizado
    }

    public class Cliente
    {
        public string Nombre { get; }
        public ClienteEstado Estado { get; private set; }

        public Cliente(string nombre)
        {
            Nombre = nombre;
            Estado = ClienteEstado.Esperando;
        }

        public void CambiarEstado(ClienteEstado nuevo)
        {
            Estado = nuevo;
            Console.WriteLine($"  [CLIENTE-ESTADO] {Nombre} → {Estado}");
```

```
        }
    }
}
```

ColorFormateador.cs

```
using System;

namespace RestauranteSimulacion
{
    public class ColorFormateador : FormateadorMeseroDecorator
    {
        private readonly ConsoleColor _color;
        public ColorFormateador(ConsoleColor color, IFormateadorMesero inner) : base(inner)
        {
            _color = color;
        }

        public override string FormatearNombre(Mesero mesero)
        {
            return base.FormatearNombre(mesero);
        }

        public override void EscribirNombre(Mesero mesero)
        {
```

```
            var previo = Console.ForegroundColor;
            Console.ForegroundColor = _color;
            Console.Write(FormatearNombre(mesero));
            Console.ForegroundColor = previo;
        }
    }
}
```

FormateadorMeseroBase.cs

```
namespace RestauranteSimulacion
{
    public class FormateadorMeseroBase : IFormateadorMesero
    {
        public string FormatearNombre(Mesero mesero) => mesero.Nombre;
        public void EscribirNombre(Mesero mesero)
        {
            System.Console.Write(FormatearNombre(mesero));
        }
    }
}
```

FormateadorMeseroDecorator.cs

```
namespace RestauranteSimulacion
{
    public abstract class FormateadorMeseroDecorator : IFormateadorMesero
    {
        protected readonly IFormateadorMesero _inner;
        protected FormateadorMeseroDecorator(IFormateadorMesero inner)
        {
            _inner = inner;
        }

        public virtual string FormatearNombre(Mesero mesero) =>
_inner.FormatearNombre(mesero);
        public virtual void EscribirNombre(Mesero mesero) => _inner.EscribirNombre(mesero);
    }
}
```

IFormateadorMesero.cs

```
namespace RestauranteSimulacion
{
    public interface IFormateadorMesero
    {
        string FormatearNombre(Mesero mesero);
        void EscribirNombre(Mesero mesero);
    }
}
```

PoolCapacidadGlobal.cs

```
using System;


namespace RestauranteSimulacion
{
    public static class PoolCapacidadGlobal
    {
        public static int CapacidadMaximaGlobal { get; set; } = 10;
        public static int TotalObjetos { get; private set; } = 0;


        public static bool PuedeAgregar()
        {
            return TotalObjetos < CapacidadMaximaGlobal;
        }


        public static void RegistrarAgregado()
        {
            if (TotalObjetos >= CapacidadMaximaGlobal)
            {
                throw new InvalidOperationException($"Capacidad global
{CapacidadMaximaGlobal} alcanzada");
            }
            TotalObjetos++;
        }
```

```
        public static void RegistrarEliminado()
        {
            if (TotalObjetos > 0) TotalObjetos--;
        }

        public static void ReiniciarConteo()
        {
            TotalObjetos = 0;
        }
    }
}
```

Conclusión

El proyecto está organizado en capas para mantener el código ordenado y fácil de modificar.

Program.cs es la capa de Presentación; solo inicia la simulación.

AtencionService es la capa de Aplicación; ahí se simula todo el caso de uso que en este caso es el atender cliente

El Dominio contiene las reglas del negocio y los patrones (Singleton, Pool, Decorator, State) que representan cómo funciona el restaurante.

Infrastructure contiene adaptadores técnicos como el ConsoleLogger y el repositorio en memoria.

Con esta separación puedo cambiar la consola, el almacenamiento o incluso agregar nuevas salidas sin tocar la lógica del restaurante. Por eso esta arquitectura esta buena, porque hace el sistema más claro, mantenible y explicable