TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

MATERIA:
Patrones de diseño

TÍTULO ACTIVIDAD:
Examen unidad 2

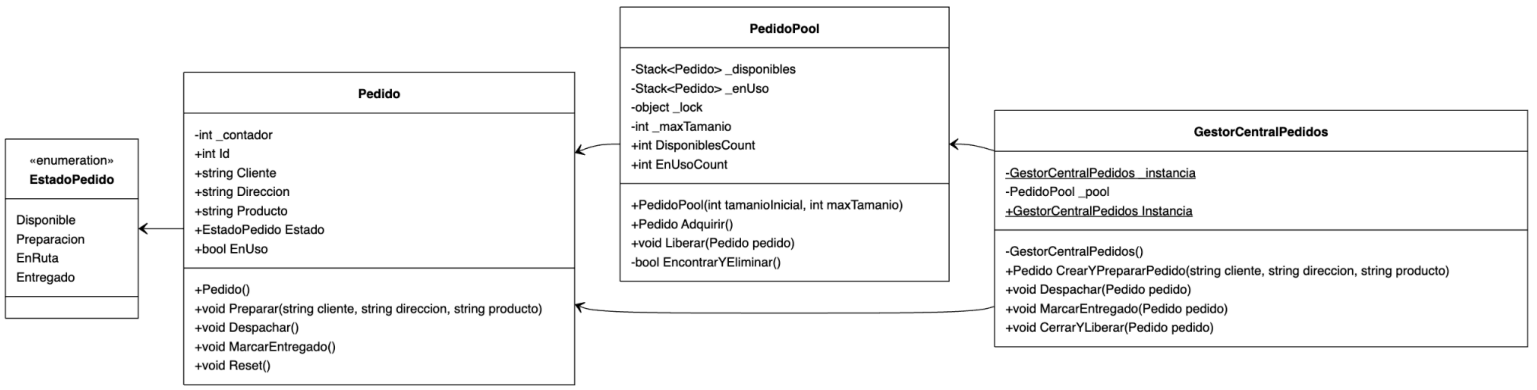
UNIDAD A EVALUAR:
Unidad 2

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

Vizuet Acosta Andre 21212372

NOMBRE DEL MAESTRO (A):
Maribel Guerrero Luis

UML



EstadoPedido

```
    8 references
1   enum EstadoPedido
2   {
        1 reference
3       Disponible,
        2 references
4       Preparacion,
        2 references
5       EnRuta,
        2 references
6       Entregado
7   }
```

Pedido

```
18 references
class Pedido
{
    3 references
    public int Id { get; private set; }
    5 references
    public string Cliente { get; private set; } = "";
    5 references
    public string Direccion { get; private set; } = "";
    5 references
    public string Producto { get; private set; } = "";
    10 references
    public EstadoPedido Estado { get; private set; }
    2 references
    public bool EnUso { get; set; }

    1 reference
    private static int _contador = 1;
    2 references
    public Pedido()
    {
        Id = _contador++;
        Reset();
    }

    1 reference
    public void Preparar(string cliente, string direccion, string producto)
    {
        Cliente = cliente;
        Direccion = direccion;
        Producto = producto;
        Estado = EstadoPedido.Preparacion;
        Console.WriteLine($"Pedido en {this.Estado}: {this.Cliente}, {this.Direccion}, {this.Producto}");
    }
```

```
25
        1 reference
26      public void Despachar()
27      {
28          if (Estado ≠ EstadoPedido.Preparacion)
29              throw new InvalidOperationException("Solo se puede despachar un pedido en preparación.");
30          Estado = EstadoPedido.EnRuta;
31          Console.WriteLine($"Pedido {this.Estado}: {this.Cliente}, {this.Direccion}, {this.Producto}");
32      }
33
        1 reference
34      public void MarcarEntregado()
35      {
36          if (Estado ≠ EstadoPedido.EnRuta)
37              throw new InvalidOperationException("Solo se puede marcar entregado un pedido en ruta.");
38          Estado = EstadoPedido.Entregado;
39          Console.WriteLine($"Pedido {this.Estado}: {this.Cliente}, {this.Direccion}, {this.Producto}");
40      }
41
        2 references
42      public void Reset()
43      {
44          Cliente = "";
45          Direccion = "";
46          Producto = "";
47          Estado = EstadoPedido.Disponible;
48          EnUso = false;
49      }
50  }
```

Singleton

```
6 references
sealed class GestorCentralPedidos
{
    1 reference
    private static readonly GestorCentralPedidos _instancia = new GestorCentralPedidos();
    1 reference
    public static GestorCentralPedidos Instancia => _instancia;

    3 references
    private readonly PedidoPool _pool;

    1 reference
    private GestorCentralPedidos()
    {
        _pool = new PedidoPool(tamanioInicial: 2, maxTamanio: 100);
    }

    2 references
    public Pedido CrearYPrepararPedido(string cliente, string direccion, string producto)
    {
        Pedido pedido = _pool.Adquirir();
        pedido.Preparar(cliente, direccion, producto);
        return pedido;
    }

    2 references
    public void Despachar(Pedido pedido) => pedido.Despachar();
    2 references
    public void MarcarEntregado(Pedido pedido) => pedido.MarcarEntregado();

    2 references
    public void CerrarYLiberar(Pedido pedido)
    {
        if (pedido.Estado != EstadoPedido.Entregado)
            throw new InvalidOperationException("No se puede liberar un pedido que no ha sido entregado.");
        _pool.Liberar(pedido);
    }
}
```

ObjectPool

```
3 references
class PedidoPool
{
    4 references
    private readonly Stack<Pedido> _disponibles = new Stack<Pedido>();
    4 references
    private readonly Stack<Pedido> _enUso = new Stack<Pedido>();
    4 references
    private readonly object _lock = new object();
    2 references
    private readonly int _maxTamanio;

    2 references
    public int DisponiblesCount { get { lock (_lock) return _disponibles.Count; } }
    2 references
    public int EnUsoCount { get { lock (_lock) return _enUso.Count; } }

    1 reference
    public PedidoPool(int tamanioInicial, int maxTamanio)
    {
        _maxTamanio = maxTamanio;
        for (int i = 0; i < tamanioInicial; i++)
            _disponibles.Push(new Pedido());
    }

```

```
        1 reference
17      public Pedido Adquirir()
18      {
19          lock (_lock)
20          {
21              Pedido p;
22              if (DisponiblesCount > 0)
23                  p = _disponibles.Pop();
24              else
25              {
26                  int totalActual = DisponiblesCount + EnUsoCount;
27                  if (totalActual ≥ _maxTamanio)
28                      throw new InvalidOperationException("El pool alcanzó su tamaño máximo.");
29
30                  p = new Pedido();
31              }
32              p.EnUso = true;
33              _enUso.Push(p);
34              return p;
35          }
36      }
37
        1 reference
38      public void Liberar(Pedido pedido)
39      {
40          if (pedido == null) return;
41
42          lock (_lock)
43          {
44              bool EncontrarYEliminar()
45              {
46                  if (EnUsoCount == 0) return false;
47                  Pedido top = _enUso.Pop();
48                  if (top == pedido)
49                  {
50                      return true;
51                  }
52                  bool encontradoEnResto = EncontrarYEliminar();
53                  _enUso.Push(top);
54                  return encontradoEnResto;
55              }
56              pedido.Reset();
57              _disponibles.Push(pedido);
58          }
59      }
60  }
```

Main

```
using System;
using System.Collections.Generic;
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        GestorCentralPedidos gestor = GestorCentralPedidos.Instancia;

        Console.WriteLine("== Pedido 1 ==");
        Pedido pedido1 = gestor.CrearYPrepararPedido(
            cliente: "Edgar Manuel",
            direccion: "Sta. Ximena 49",
            producto: "Azzaro The Most Wanted EDP Intense 100ml"
        );
        gestor.Despachar(pedido1);
        gestor.MarcarEntregado(pedido1);
        gestor.CerrarYLiberar(pedido1);

        Console.WriteLine("\n== Pedido 2 ==");
        Pedido pedido2 = gestor.CrearYPrepararPedido(
            cliente: "Andre Vizuet",
            direccion: "Villas del Real",
            producto: "Jean Paul Gaultier Le Male Elixir 125ml"
        );
        gestor.Despachar(pedido2);
        gestor.MarcarEntregado(pedido2);
        gestor.CerrarYLiberar(pedido2);

        Console.WriteLine($"\nComprobación de reutilización: \n Pedido 1: {pedido1.Id}\n Pedido 2: {pedido2.Id}");
    }
}
```

Ejecución

```
ExamenUnidad2_Patrones_VizuetAcostaAndre/src on  main [?] via .NET v9.0.102 🎯 net9.0
❯ dotnet run
== Pedido 1 ==
Pedido en Preparacion: Edgar Manuel, Sta. Ximena 49, Azzaro The Most Wanted EDP Intense 100ml
Pedido EnRuta: Edgar Manuel, Sta. Ximena 49, Azzaro The Most Wanted EDP Intense 100ml
Pedido Entregado: Edgar Manuel, Sta. Ximena 49, Azzaro The Most Wanted EDP Intense 100ml

== Pedido 2 ==
Pedido en Preparacion: Andre Vizuet, Villas del Real, Jean Paul Gaultier Le Male Elixir 125ml
Pedido EnRuta: Andre Vizuet, Villas del Real, Jean Paul Gaultier Le Male Elixir 125ml
Pedido Entregado: Andre Vizuet, Villas del Real, Jean Paul Gaultier Le Male Elixir 125ml

Comprobación de reutilización:
 Pedido 1: 2
 Pedido 2: 2
```

Conclusión

Con singleton centralizamos todo en una sola instancia que va a gestionar los recursos y optimiza el uso de objetos mediante la reutilización con el patrón objectpool. Esta combinación va a mejorar el rendimiento, evita los duplicados de instancias, el código es más fácil de mantener y eficiente.