



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®



INSTITUTO TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

MATERIA

Patrones de diseño

NOMBRE DEL TRABAJO

Examen unidad 2

UNIDAD 2

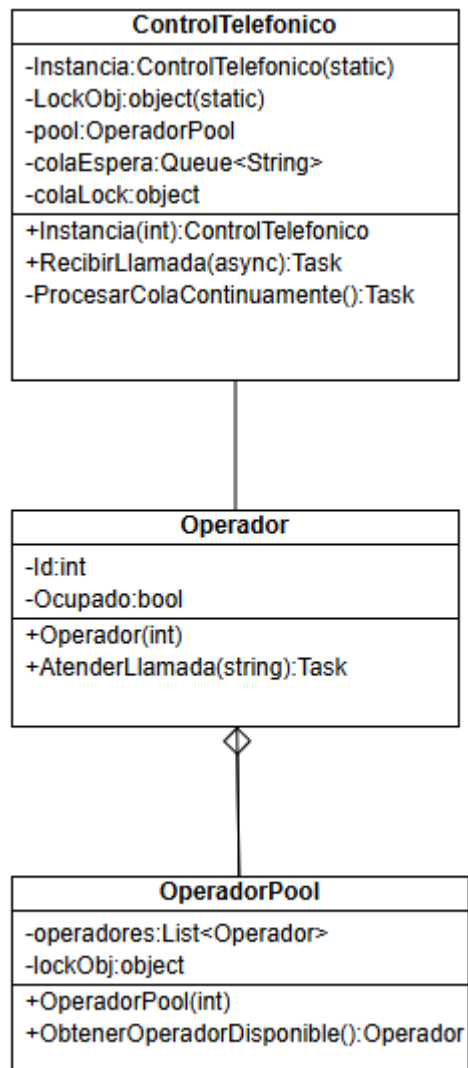
ALUMNO

Cuevas Aparicio Edwin - 21212328

MAESTRA

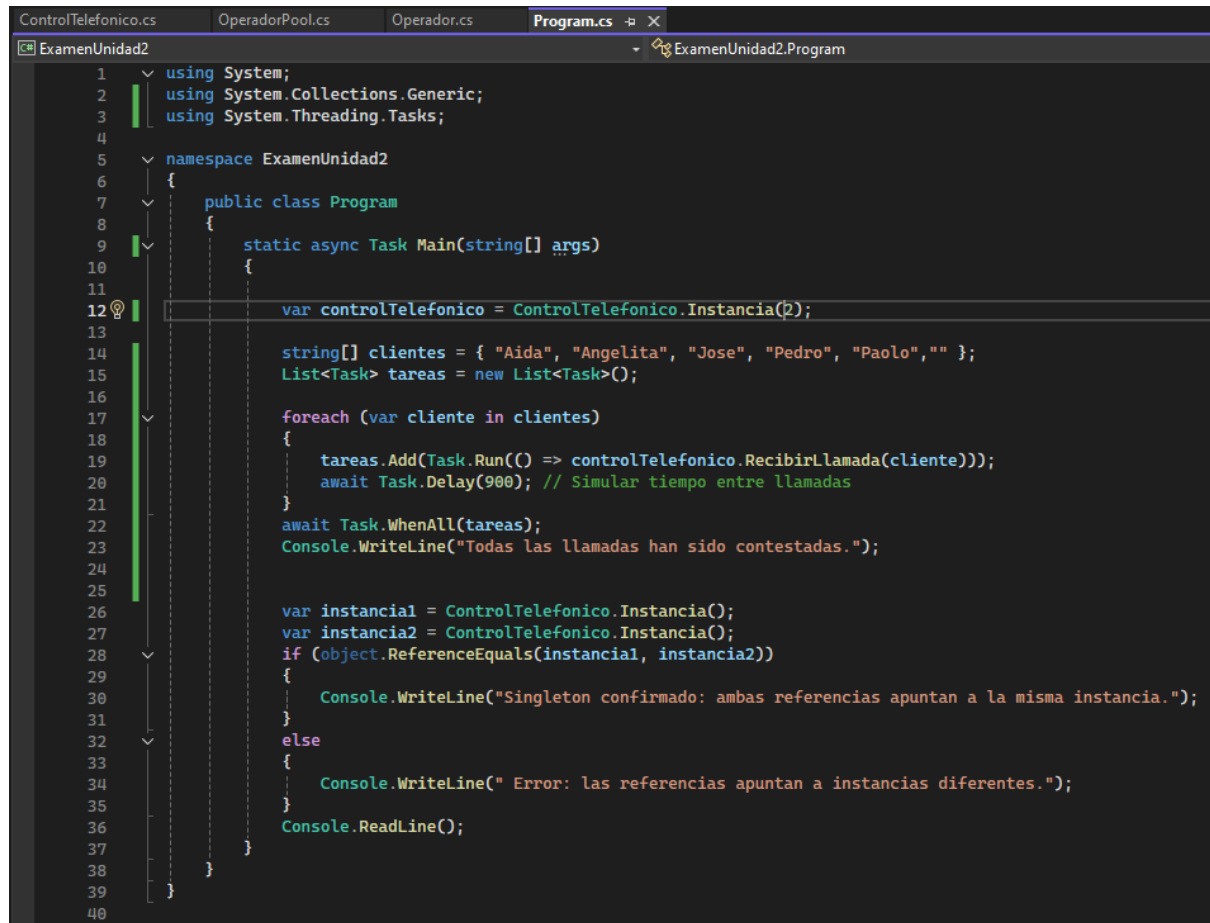
Maribel Guerrero Luis

Diagrama de clases UML



Código

Program



```
1  using System;
2  using System.Collections.Generic;
3  using System.Threading.Tasks;
4
5  namespace ExamenUnidad2
6  {
7      public class Program
8      {
9          static async Task Main(string[] args)
10         {
11
12             var controlTelefonico = ControlTelefonico.Instancia(2);
13
14             string[] clientes = { "Aida", "Angelita", "Jose", "Pedro", "Paolo", "" };
15             List<Task> tareas = new List<Task>();
16
17             foreach (var cliente in clientes)
18             {
19                 tareas.Add(Task.Run(() => controlTelefonico.RecibirLlamada(cliente)));
20                 await Task.Delay(900); // Simular tiempo entre llamadas
21             }
22             await Task.WhenAll(tareas);
23             Console.WriteLine("Todas las llamadas han sido contestadas.");
24
25
26             var instancial = ControlTelefonico.Instancia();
27             var instancia2 = ControlTelefonico.Instancia();
28             if (object.ReferenceEquals(instancial, instancia2))
29             {
30                 Console.WriteLine("Singleton confirmado: ambas referencias apuntan a la misma instancia.");
31             }
32             else
33             {
34                 Console.WriteLine(" Error: las referencias apuntan a instancias diferentes.");
35             }
36             Console.ReadLine();
37         }
38     }
39 }
40
```

Operador

```
ExamenUnidad2 ExamenUnidad2.Operador
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExamenUnidad2
8  {
9      public class Operador
10     {
11         public int Id { get; set; }
12         public bool Ocupado { get; set; }
13
14         public Operador(int id)
15         {
16             Id = id;
17             Ocupado = false;
18         }
19
20         public void AtenderLlamada(string cliente)
21         {
22             Ocupado = true;
23             Console.WriteLine($"Operador {Id} atendiendo llamada de {cliente}...");
24             // Simular tiempo de atención
25             System.Threading.Thread.Sleep(1000);
26             Console.WriteLine($"Operador {Id} terminó la llamada de {cliente}.");
27             Ocupado = false;
28         }
29     }
30 }
31
```

ControlTelefonico

```
ExamenUnidad2 - ExamenUnidad2.ControlTelefonico - RecibirLlan
1  using System;
2  using System.Threading.Tasks;
3
4  namespace ExamenUnidad2
5  {
6      public class ControlTelefonico
7      {
8
9          private static ControlTelefonico instancia;
10         private static readonly object lockObj = new object();
11         private readonly OperadorPool operadorPool;
12
13         private ControlTelefonico(int cantidadOperadores)
14         {
15             operadorPool = new OperadorPool(cantidadOperadores);
16         }
17
18         public static ControlTelefonico Instancia(int cantidadOperadores = 3)
19         {
20             if (instancia == null)
21             {
22                 lock (lockObj)
23                 {
24                     if (instancia == null)
25                     {
26                         instancia = new ControlTelefonico(cantidadOperadores);
27                     }
28                 }
29             }
30             return instancia;
31         }
32
33         public void RecibirLlamada(string cliente)
34         {
35             var operador = operadorPool.ObtenerOperadorLibre();
36             if (operador != null)
37             {
38                 operador.AtenderLlamada(cliente);
39             }
40             else
41             {
42                 Console.WriteLine($"No hay operadores disponibles para atender la llamada de {cliente}. Por favor, espere.");
43             }
44         }
45     }
46 }
47
```

OperadorPool

```
ExamenUnidad2
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Security.Cryptography.X509Certificates;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace ExamenUnidad2
9  {
10     public class OperadorPool
11     {
12
13         private readonly List<Operador> operadores;
14         private readonly object lockObject = new object();
15
16         public OperadorPool(int cantidad)
17         {
18             operadores = new List<Operador>();
19             for (int i = 0; i <= 1; i++)
20             {
21                 operadores.Add(new Operador(i + 1));
22             }
23         }
24         public Operador ObtenerOperadorLibre()
25         {
26             lock (lockObject)
27             {
28                 foreach (var operador in operadores)
29                 {
30                     if (!operador.Ocupado)
31                     {
32                         return operador;
33                     }
34                 }
35             }
36             return null;
37         }
38     }
39 }
40
41
```

Ejecución

```
Seleccíon F:\Patrones\ExamenUnidad2\ExamenUnidad2\bin\Debug\ExamenUnidad2.exe
Operador 1 atendiendo llamada de Aida...
Operador 2 atendiendo llamada de Angelita...
Operador 1 terminó la llamada de Aida.
Operador 1 atendiendo llamada de Jose...
Operador 2 terminó la llamada de Angelita.
Operador 2 atendiendo llamada de Pedro...
Operador 1 terminó la llamada de Jose.
Operador 1 atendiendo llamada de Paolo...
Operador 2 terminó la llamada de Pedro.
Operador 2 atendiendo llamada de ...
Operador 1 terminó la llamada de Paolo.
Operador 2 terminó la llamada de .
Todas las llamadas han sido contestadas.
Singleton confirmado: ambas referencias apuntan a la misma instancia.
```

Conclusión

Los 2 patrones en este problema ayuda ya que el singleton y el object pool se combinan para que se ejecute el programa en una sola instancia y que contenga operadores contestando diferentes llamadas pero de una sola línea y eso ayuda a que no se sature la línea por lo cual ayuda demasiado los 2 patrones combinados.

En resumen, estos patrones fortalecen la arquitectura del sistema, ya que brindan una solución escalable, eficiente y mantenible para la gestión de los operadores.