

**TECNOLOGICO 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 A EVALUAR:**

unidad 2

**NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:**

Casales Crus Josue Jacob 21211927

**NOMBRE DEL MAESTRO (A):**

Maribel Guerrero Luis

Código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

public class Program
{
    // Lista de piezas de música

    private static readonly Dictionary<int, string> RockMusicPieces = new
        Dictionary<int, string>
    {
        { 1, "Bohemian Rhapsody - Queen" }, { 2, "Stairway to Heaven - Led Zeppelin" },
        { 3, "Smells Like Teen Spirit - Nirvana" }, { 4, "Sweet Child O' Mine - Guns N'
            Roses" },
        { 5, "Thunderstruck - AC/DC" }, { 6, "Imagine - John Lennon" },
        { 7, "Livin' on a Prayer - Bon Jovi" }, { 8, "Hotel California - Eagles" },
        { 9, "Back in Black - AC/DC" }, { 10, "Don't Stop Believin' - Journey" }
    };

    //
    =====
    =

    // 1. Patrón SINGLETON 1: PlayerConfiguration (Configuración de Audio)

    //
    =====
    =

    public sealed class PlayerConfiguration
    {
```

```
private static readonly PlayerConfiguration instance = new  
    PlayerConfiguration();
```

```
public string AudioFormat { get; private set; }
```

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

```
private PlayerConfiguration()
```

```
{
```

```
    AudioFormat = "MP3";
```

```
    VolumeLevel = 50;
```

```
}
```

```
public static PlayerConfiguration Instance => instance;
```

```
public void SetVolume(int volume)
```

```
{
```

```
    // 🐛 CORRECCIÓN DEL ERROR DE CLAMP
```

```
    // Implementación manual de Clamp: asegura que el volumen esté entre 0 y  
    100
```

```
    this.VolumeLevel = Math.Max(0, Math.Min(100, volume));
```

```
}
```

```
public override string ToString()
```

```
{
```

```
    return $"Formato: {AudioFormat}\nVolumen: {VolumeLevel}%";
```

```
}
```

```
}
```

```

//
=====
=
// 2. Objeto Reutilizable: AudioBuffer
//
=====
=
public class AudioBuffer
{
    private int id;

    public bool InUse { get; set; }

    public AudioBuffer(int id) { this.id = id; this.InUse = false; }

    public void LoadData(string musicSegment)
    {
        Console.WriteLine($"[Cargando] Buffer {id} (64KB) cargado con:
        {musicSegment}.");

        InUse = true;
    }

    public void Flush()
    {
        Console.WriteLine($"[Liberando] Buffer {id} listo para reutilizar.");

        InUse = false;
    }

    public void Play()
    {
        var config = PlayerConfiguration.Instance;

```

```
Console.WriteLine($"[REPRODUCIENDO] Buffer {id} al  
{config.VolumeLevel}%...");
```

```
}
```

```
}
```

```
//
```

```
=====
```

```
=
```

```
// 3. Patrón SINGLETON 2: SoundBufferPool (Object Pool)
```

```
//
```

```
=====
```

```
=
```

```
public sealed class SoundBufferPool
```

```
{
```

```
private static readonly SoundBufferPool instance = new SoundBufferPool();
```

```
private List<AudioBuffer> pool;
```

```
private const int MaxPoolSize = 3;
```

```
private SoundBufferPool()
```

```
{
```

```
pool = new List<AudioBuffer>();
```

```
for (int i = 0; i < MaxPoolSize; i++)
```

```
{
```

```
pool.Add(new AudioBuffer(i + 1));
```

```
}
```

```
Console.WriteLine($"*** SoundBufferPool (Singleton) inicializado con  
{MaxPoolSize} buffers. ***");
```

```
}
```

```
public static SoundBufferPool Instance => instance;
```

```
public AudioBuffer AcquireBuffer()
```

```
{
```

```
    foreach (var buffer in pool)
```

```
    {
```

```
        if (!buffer.InUse)
```

```
        {
```

```
            return buffer;
```

```
        }
```

```
    }
```

```
    Console.WriteLine("POOL LLENO: No hay buffers disponibles.  
        Esperando...");
```

```
    return null;
```

```
}
```

```
public void ReleaseBuffer(AudioBuffer buffer)
```

```
{
```

```
    if (buffer != null)
```

```
    {
```

```
        buffer.Flush();
```

```
    }
```

```
}
```

```
}
```

```
//
```

```
=====
```

```
=
```

// 4. Patrón SINGLETON 3: MusicStreamer (Gestor de Reproducción Continua)

//

=====

=

public sealed class MusicStreamer

{

private static readonly MusicStreamer instance = new MusicStreamer();

private Queue<AudioBuffer> \_playbackQueue;

private MusicStreamer()

{

\_playbackQueue = new Queue<AudioBuffer>();

Console.WriteLine("\n\*\*\* MusicStreamer (Singleton de Reproducción)  
Inicializado. \*\*\*");

}

public static MusicStreamer Instance => instance;

public void StartContinuousPlayback(string musicPiece)

{

Console.WriteLine(\$"Iniciando \*\*REPRODUCCIÓN CONTINUA\*\* de:  
{musicPiece}...");

var bufferPool = SoundBufferPool.Instance;

int segmentCount = 0;

bool isPlaying = true;

// Llenado inicial (Pre-buffering)

```

while (_playbackQueue.Count < 2 && segmentCount < 3)
{
    AudioBuffer buffer = bufferPool.AcquireBuffer();
    if (buffer != null)
    {
        segmentCount++;
        buffer.LoadData($"{musicPiece} (Segmento {segmentCount})");
        _playbackQueue.Enqueue(buffer);
    }
}

```

// Bucle que simula la continuidad de la reproducción (buffer por buffer)

```

while (isPlaying)
{
    // Reproducir el segmento actual
    if (_playbackQueue.Any())
    {
        AudioBuffer currentBuffer = _playbackQueue.Dequeue();
        currentBuffer.Play();

        Thread.Sleep(500);

        // Liberar el buffer usado para que pueda ser reutilizado
        bufferPool.ReleaseBuffer(currentBuffer);
    }
}

```

// Cargar el siguiente segmento (Continuidad / Reutilización del Object Pool)



```

        AudioBuffer newBuffer = bufferPool.AcquireBuffer();

        if (newBuffer != null)
        {
            segmentCount++;

            newBuffer.LoadData($"{musicPiece} (Segmento {segmentCount})");

            _playbackQueue.Enqueue(newBuffer);
        }
    }
    else
    {
        isPlaying = false;
    }

    // Simulación de fin de canción
    if (segmentCount > 8) isPlaying = false;
    }

    // Liberación final
    while (_playbackQueue.Any())
        bufferPool.ReleaseBuffer(_playbackQueue.Dequeue());

    Console.WriteLine("\n--- SIMULACIÓN DE REPRODUCCIÓN CONTINUA
        FINALIZADA ---");
    }
}

//
=====
=

```

```
// 5. MAIN (Prueba de Instancias Únicas y Ejecución)
```

```
//
```

```
=====
```

```
=
```

```
public static void Main(string[] args)
```

```
{
```

```
Console.WriteLine("--- Prueba de INSTANCIA ÚNICA (SINGLETON) ---\n");
```

```
var conf1 = PlayerConfiguration.Instance;
```

```
var conf2 = PlayerConfiguration.Instance;
```

```
Console.WriteLine($"Configuración: Misma Instancia?  
{object.ReferenceEquals(conf1, conf2)}");
```

```
var poolA = SoundBufferPool.Instance;
```

```
var poolB = SoundBufferPool.Instance;
```

```
Console.WriteLine($"Buffer Pool: Misma Instancia?  
{object.ReferenceEquals(poolA, poolB)}");
```

```
var streamerA = MusicStreamer.Instance;
```

```
var streamerB = MusicStreamer.Instance;
```

```
Console.WriteLine($"Reproductor: Misma Instancia?  
{object.ReferenceEquals(streamerA, streamerB)}");
```

```
Console.WriteLine("\n--- INICIO DE REPRODUCCIÓN CONTINUA CON  
INSTANCIAS ÚNICAS ---\n");
```

```
bool continuePlaying = true;
```

```
while (continuePlaying)
```

```
{
```

```

Console.WriteLine("=====
=====");

Console.WriteLine("Selecciona una pieza musical (1-10):");

foreach (var musicPiece in RockMusicPieces)
Console.WriteLine($"{musicPiece.Key}: {musicPiece.Value}");


Console.WriteLine("\nIngresa el número: ");

if (int.TryParse(Console.ReadLine(), out int selection) &&
    RockMusicPieces.ContainsKey(selection))
{
    string selectedMusic = RockMusicPieces[selection];

    conf1.SetVolume(80);

    Console.WriteLine($"CONFIGURACIÓN GLOBAL (Singleton) antes de
    iniciar:");

    Console.WriteLine(conf1.ToString());

    // Se usa la única instancia del Streamer (Singleton) para iniciar la lógica
    continua

    streamerA.StartContinuousPlayback(selectedMusic);

    Console.WriteLine("\n¿Deseas reproducir otra pieza? (S/N)");

    if (Console.ReadLine()?.ToUpper() != "S") continuePlaying = false;
    }
    else
    {

    Console.WriteLine("\nSelección inválida. Presiona cualquier tecla para
    reintentar...");

    Console.ReadKey();
}

```

```

    }

    Console.Clear();

    }

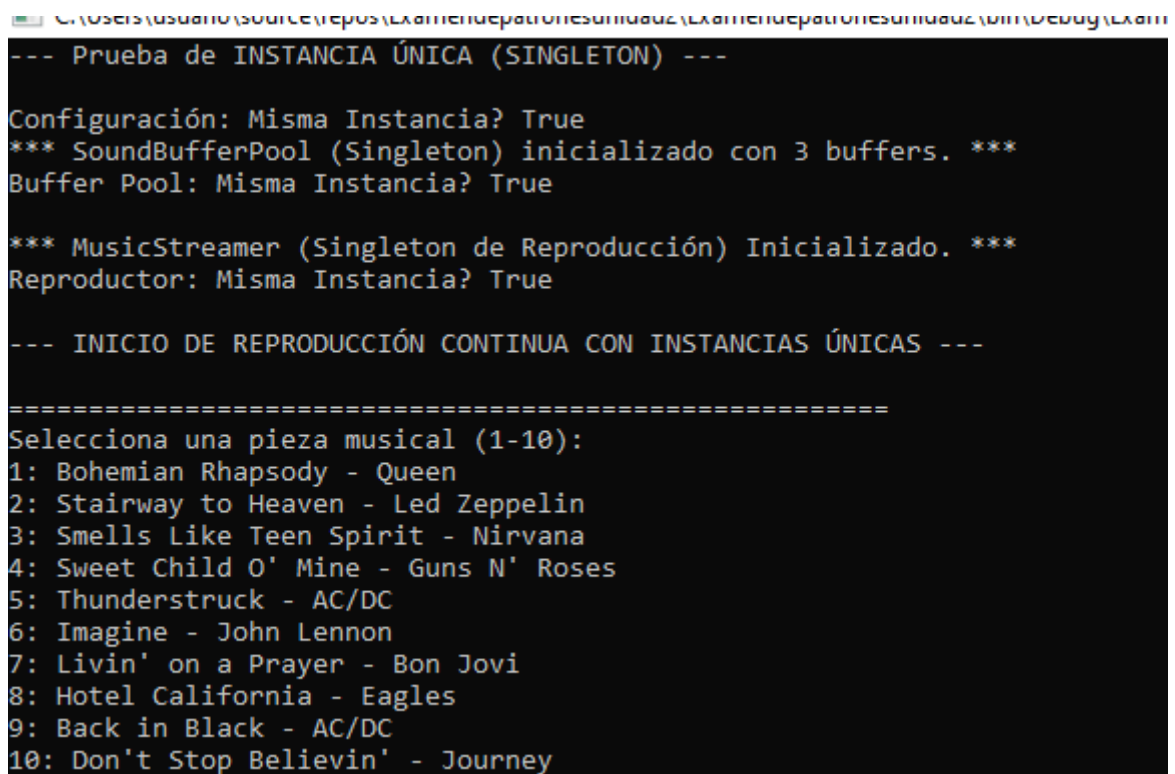
    Console.WriteLine("Reproductor detenido. Saliendo.");

    }

}

```

Ejecución:



```

C:\Users\usuario\source\repos\Examen de patrones de diseño\Examen de patrones de diseño\bin\Debug\Examen
--- Prueba de INSTANCIA ÚNICA (SINGLETON) ---

Configuración: Misma Instancia? True
*** SoundBufferPool (Singleton) inicializado con 3 buffers. ***
Buffer Pool: Misma Instancia? True

*** MusicStreamer (Singleton de Reproducción) Inicializado. ***
Reproductor: Misma Instancia? True

--- INICIO DE REPRODUCCIÓN CONTINUA CON INSTANCIAS ÚNICAS ---

=====
Selecciona una pieza musical (1-10):
1: Bohemian Rhapsody - Queen
2: Stairway to Heaven - Led Zeppelin
3: Smells Like Teen Spirit - Nirvana
4: Sweet Child O' Mine - Guns N' Roses
5: Thunderstruck - AC/DC
6: Imagine - John Lennon
7: Livin' on a Prayer - Bon Jovi
8: Hotel California - Eagles
9: Back in Black - AC/DC
10: Don't Stop Believin' - Journey

```

Ingresa el número: 1

CONFIGURACIÓN GLOBAL (Singleton) antes de iniciar:

Formato: MP3

Volumen: 80%

Iniciando **\*\*REPRODUCCIÓN CONTINUA\*\*** de: Bohemian Rhapsody - Queen...

[Cargando] Buffer 1 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 1).

[Cargando] Buffer 2 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 2).

[REPRODUCIENDO] Buffer 1 al 80%...

[Liberando] Buffer 1 listo para reutilizar.

[Cargando] Buffer 1 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 3).

[REPRODUCIENDO] Buffer 2 al 80%...

[Liberando] Buffer 2 listo para reutilizar.

[Cargando] Buffer 2 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 4).

[REPRODUCIENDO] Buffer 1 al 80%...

[Liberando] Buffer 1 listo para reutilizar.

[Cargando] Buffer 1 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 5).

[REPRODUCIENDO] Buffer 2 al 80%...

[Liberando] Buffer 2 listo para reutilizar.

[Cargando] Buffer 2 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 6).

[REPRODUCIENDO] Buffer 1 al 80%...

[Liberando] Buffer 1 listo para reutilizar.

[Cargando] Buffer 1 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 7).

[REPRODUCIENDO] Buffer 2 al 80%...

[Liberando] Buffer 2 listo para reutilizar.

[Cargando] Buffer 2 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 8).

[REPRODUCIENDO] Buffer 1 al 80%...

[Liberando] Buffer 1 listo para reutilizar.

[Cargando] Buffer 1 (64KB) cargado con: Bohemian Rhapsody - Queen (Segmento 9).

[Liberando] Buffer 2 listo para reutilizar.

[Liberando] Buffer 1 listo para reutilizar.

--- SIMULACIÓN DE REPRODUCCIÓN CONTINUA FINALIZADA ---

¿Deseas reproducir otra pieza? (S/N)

C:\Users\usuario\source\repos\Examen de patrones unidad 2\Examen de patrones unidad 2\bin\Debug\Examen de patrones unidad 2.exe

```
=====
Selecciona una pieza musical (1-10):
1: Bohemian Rhapsody - Queen
2: Stairway to Heaven - Led Zeppelin
3: Smells Like Teen Spirit - Nirvana
4: Sweet Child O' Mine - Guns N' Roses
5: Thunderstruck - AC/DC
6: Imagine - John Lennon
7: Livin' on a Prayer - Bon Jovi
8: Hotel California - Eagles
9: Back in Black - AC/DC
10: Don't Stop Believin' - Journey

Ingresa el número: 3

CONFIGURACIÓN GLOBAL (Singleton) antes de iniciar:
Formato: MP3
Volumen: 80%

Iniciando **REPRODUCCIÓN CONTINUA** de: Smells Like Teen Spirit - Nirvana...
[Cargando] Buffer 1 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 1).
[Cargando] Buffer 2 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 2).
[REPRODUCIENDO] Buffer 1 al 80%...
[Liberando] Buffer 1 listo para reutilizar.
[Cargando] Buffer 1 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 3).
[REPRODUCIENDO] Buffer 2 al 80%...
[Liberando] Buffer 2 listo para reutilizar.
[Cargando] Buffer 2 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 4).
[REPRODUCIENDO] Buffer 1 al 80%...
[Liberando] Buffer 1 listo para reutilizar.
[Cargando] Buffer 1 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 5).
[REPRODUCIENDO] Buffer 2 al 80%...
[Liberando] Buffer 2 listo para reutilizar.
[Cargando] Buffer 2 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 6).
[REPRODUCIENDO] Buffer 1 al 80%...
[Liberando] Buffer 1 listo para reutilizar.
[Cargando] Buffer 1 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 7).
[REPRODUCIENDO] Buffer 2 al 80%...
[Liberando] Buffer 2 listo para reutilizar.
[Cargando] Buffer 2 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 8).
[REPRODUCIENDO] Buffer 1 al 80%...
[Liberando] Buffer 1 listo para reutilizar.
[Cargando] Buffer 1 (64KB) cargado con: Smells Like Teen Spirit - Nirvana (Segmento 9).
[Liberando] Buffer 2 listo para reutilizar.
[Liberando] Buffer 1 listo para reutilizar.

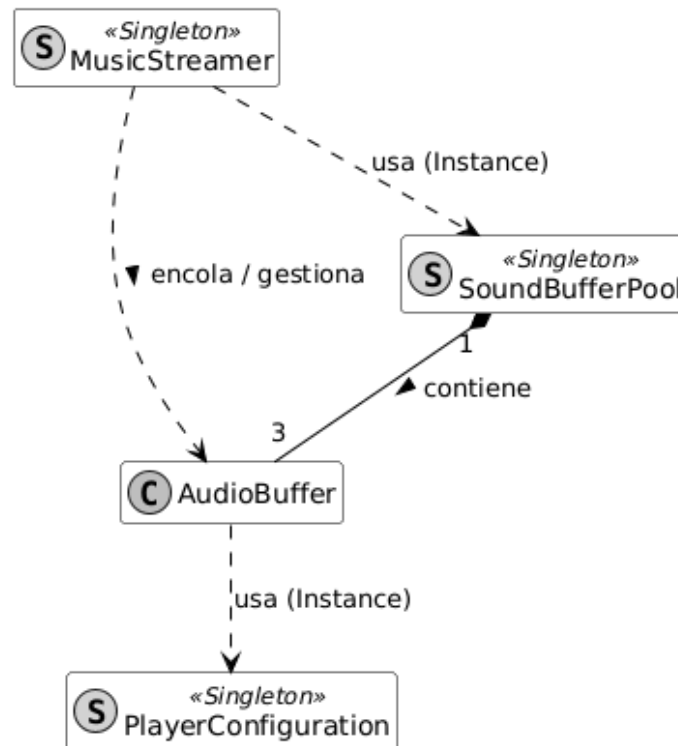
--- SIMULACIÓN DE REPRODUCCIÓN CONTINUA FINALIZADA ---

¿Deseas reproducir otra pieza? (S/N)

```

Diagrama uml:

### Diagrama de Clases: Singleton y Object Pool para Reproductor de Música



### Conclusiones:

Se establecieron tres componentes clave como Singletons: la **Configuración** (**PlayerConfiguration**), el **Gestor de Buffers** (**SoundBufferPool**) y el **Reproductor Principal** (**MusicStreamer**). Esto asegura que solo exista una instancia global de cada uno, eliminando conflictos y centralizando las decisiones de volumen y formato. El verdadero motor de la solución es la instancia única de **MusicStreamer**, que ejecuta la lógica de **reproducción continua**. Para ello, depende de la también única instancia de **SoundBufferPool**, reutilizando de manera eficiente los objetos de memoria (**AudioBuffer**). Este ciclo constante de adquirir, reproducir y liberar buffers simula el proceso de *streaming*, asegurando un flujo de audio ininterrumpido sin la sobrecarga de crear nuevos objetos en cada segmento. En esencia, la combinación de Singletons y Object Pool resuelve el

requisito de instancia única y proporciona una base sólida y eficiente para la reproducción de música continua.