

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 2

UNIDAD A EVALUAR:

2

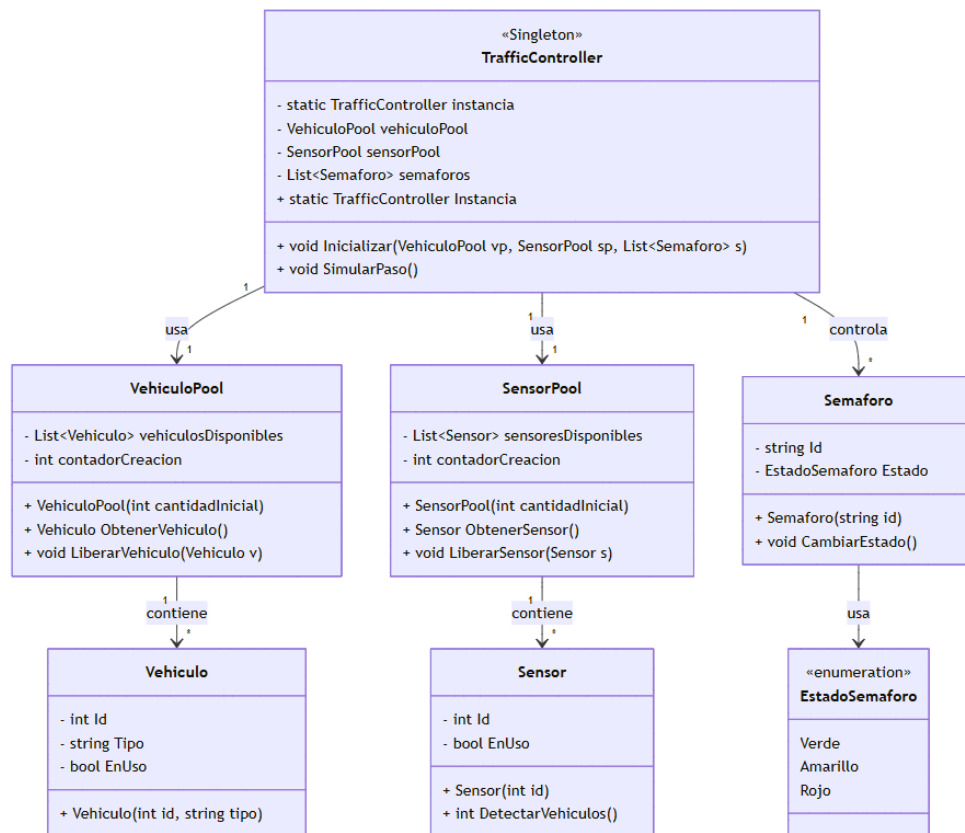
Nombre y número de control:

Pérez Villa Belen, 21212579

NOMBRE DEL MAESTRO (A):

Maribel Guerrero Luis

UML



Codigo

```

public class VehiculoPool
{
    private List<Vehiculos> vehiculosDisponibles = new
List<Vehiculos>();
    private static int contadorId = 1;

    public VehiculoPool(int cantidadInicial)
    {
        for (int i = 0; i < cantidadInicial; i++)
        {
            vehiculosDisponibles.Add(new Vehiculos(contadorId++,
"Auto"));
        }
    }
    public Vehiculos ObtenerVehiculo()
    {
        foreach(var v in vehiculosDisponibles)
        {
            if (!v.EnUso)
            {
                v.EnUso = true;
                return v;
            }
        }
    }
}

```

```

        }
    }
    // Si no hay disponibles, crear uno nuevo (expansion del
pool)
    var nuevo = new Vehiculos(contadorId++, "Auto Extra");
    nuevo.EnUso = true;
    vehiculosDisponibles.Add (nuevo);
    return nuevo;
}
public void LiberarVehiculo(Vehiculos v)
{
    v.EnUso = false;
}
}
public class Vehiculos
{
    public int Id { get; private set; }
    public string Tipo { get; private set; }
    public bool EnUso { get; set; }

    public Vehiculos(int id, string tipo)
    {
        Id = id;
        Tipo = tipo;
        EnUso = false;
    }
}

}
public class ControlCentral
{
    private static ControlCentral instance;
    private List<Semaforo> semaforos = new List<Semaforo>();
    private VehiculoPool vehiculoPool;
    private SensorPool sensorPool;
    private Random rnd = new Random();

    private ControlCentral() { }
    public static ControlCentral Instance
    {
        get
        {
            if (instance == null)
                instance = new ControlCentral();
            return instance;
        }
    }
    public void Inicializar(VehiculoPool vpool, SensorPool spool,
List<Semaforo> listaSemaforos)

```

```

{
    vehiculoPool = vpool;
    sensorPool = spool;
    semaforos = listaSemaforos;
}
public void SimularPaso()
{
    Console.WriteLine("\n===== CICLO DE SIMULACION =====");

    foreach(var semaforo in semaforos)
    {
        semaforo.CambiarEstado();
        semaforo.MostrarEstado();

        var sensor = sensorPool.ObtenerSensor();
        int vehiculosEsperando = sensor.DetectarVehiculos();
        Console.WriteLine($"Sensor {sensor.Id}:
{vehiculosEsperando} vehiculos detectados.");
        if(semaforo.Estado == EstadoSemaforo.verde &&
vehiculosEsperando > 0)
        {
            List<int> idsVehiculos = new List<int>();
            for(int i = 0; i< vehiculosEsperando; i++)
            {
                var vehiculo = vehiculoPool.ObtenerVehiculo();

                vehiculoPool.LiberarVehiculo(vehiculo);
                idsVehiculos.Add(vehiculo.Id);
            }
            Console.WriteLine(" Vehículos que pasaron: [" +
string.Join(", ", idsVehiculos) + "]");
        }
        sensorPool.LiberarSensor(sensor);
    }
    Console.WriteLine("=====");
}
}
public class Sensor
{
    public int Id { get; private set; }
    public bool EnUso { get; set; }

    public Sensor(int id)
    {
        Id = id;
        EnUso = false;
    }
    public int DetectarVehiculos()

```

```

        {
            Random rnd = new Random();
            return rnd.Next(0, 10);
        }
    }
}

public enum EstadoSemaforo { verde, Amarillo, Rojo}
public class Semaforo
{
    public string Id { get; private set; }
    public EstadoSemaforo Estado { get; private set; }
    private int tiempoCambio = 0;

    public Semaforo(string id)
    {
        Id = id;
        Estado = EstadoSemaforo.Rojo;
    }

    public void CambiarEstado()
    {
        tiempoCambio++;
        if(tiempoCambio % 3 ==0 )
        {
            Estado = Estado == EstadoSemaforo.verde ?
EstadoSemaforo.Rojo : EstadoSemaforo.verde;
        }

    }

    public void MostrarEstado()
    {
        Console.WriteLine($"Semaforo {Id}: {Estado}");
    }

}

public class SensorPool
{
    private List<Sensor> sensoresDisponibles = new
List<Sensor>();
    private static int contadorId = 1;

    public SensorPool(int cantidadInicial)
    {
        for(int i = 0; i < cantidadInicial; i++)
        {
            sensoresDisponibles.Add(new Sensor(contadorId++));
        }
    }

    public Sensor ObtenerSensor()
    {

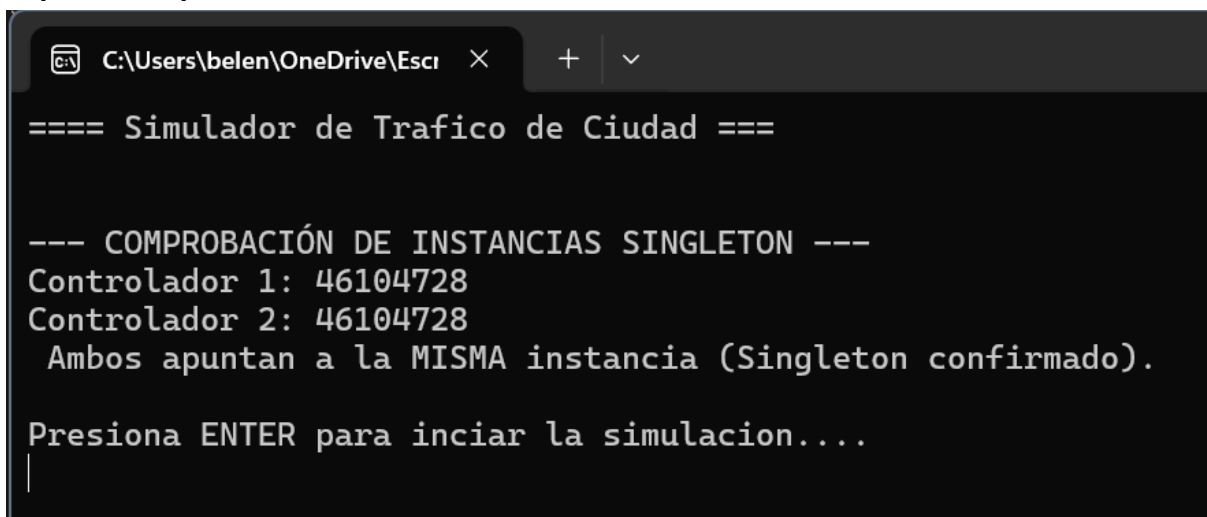
```

```

        foreach (var s in sensoresDisponibles)
        {
            if (!s.EnUso)
            {
                s.EnUso = true;
                return s;
            }
        }
        var nuevo = new Sensor(contadorId++);
        nuevo.EnUso = true;
        sensoresDisponibles.Add(nuevo);
        return nuevo;
    }
    public void LiberarSensor(Sensor s)
    {
        s.EnUso = false;
    }
}

```

Capturas de pantalla



```

C:\Users\belen\OneDrive\Escri...
==== Simulador de Trafico de Ciudad ====

--- COMPROBACIÓN DE INSTANCIAS SINGLETON ---
Controlador 1: 46104728
Controlador 2: 46104728
Ambos apuntan a la MISMA instancia (Singleton confirmado).

Presiona ENTER para inciar la simulacion....
|

```

```
=== SIMULADOR DE TRÁFICO DE CIUDAD ===  
Ciclo #1
```

```
===== CICLO DE SIMULACION =====
```

```
Semaforo A1: Rojo
```

```
Sensor 1: 8 vehiculos detectados.
```

```
Semaforo B2: Rojo
```

```
Sensor 1: 8 vehiculos detectados.
```

```
Semaforo C3: Rojo
```

```
Sensor 1: 8 vehiculos detectados.
```

```
Semaforo D4: Rojo
```

```
Sensor 1: 8 vehiculos detectados.
```

```
=====
```

```
Presiona ENTER para continuar al siguiente ciclo...
```

```
=== SIMULADOR DE TRÁFICO DE CIUDAD ===  
Ciclo #4
```

```
===== CICLO DE SIMULACION =====
```

```
Semaforo A1: verde
```

```
Sensor 1: 6 vehiculos detectados.
```

```
Vehículos que pasaron: [1, 1, 1, 1, 1, 1]
```

```
Semaforo B2: verde
```

```
Sensor 1: 4 vehiculos detectados.
```

```
Vehículos que pasaron: [1, 1, 1, 1]
```

```
Semaforo C3: verde
```

```
Sensor 1: 4 vehiculos detectados.
```

```
Vehículos que pasaron: [1, 1, 1, 1]
```

```
Semaforo D4: verde
```

```
Sensor 1: 4 vehiculos detectados.
```

```
Vehículos que pasaron: [1, 1, 1, 1]
```

```
=====
```

```
Presiona ENTER para continuar al siguiente ciclo...
```

```
SIMULADOR DE TRÁFICO DE CIUDAD ===  
#6
```

```
CICLO DE SIMULACION =====  
Semaforo A1: Rojo  
Sensor 1: 5 vehiculos detectados.  
Semaforo B2: Rojo  
Sensor 1: 5 vehiculos detectados.  
Semaforo C3: Rojo  
Sensor 1: 5 vehiculos detectados.  
Semaforo D4: Rojo  
Sensor 1: 5 vehiculos detectados.
```

```
=== SIMULADOR DE TRÁFICO DE CIUDAD ===  
Ciclo #9
```

```
===== CICLO DE SIMULACION =====  
Semaforo A1: verde  
Sensor 1: 6 vehiculos detectados.  
Vehículos que pasaron: [1, 1, 1, 1, 1, 1]  
Semaforo B2: verde  
Sensor 1: 6 vehiculos detectados.  
Vehículos que pasaron: [1, 1, 1, 1, 1, 1]  
Semaforo C3: verde  
Sensor 1: 6 vehiculos detectados.  
Vehículos que pasaron: [1, 1, 1, 1, 1, 1]  
Semaforo D4: verde  
Sensor 1: 6 vehiculos detectados.  
Vehículos que pasaron: [1, 1, 1, 1, 1, 1]  
=====
```

```
Presiona ENTER para continuar al siguiente ciclo...
```


Conclusion

El proyecto **Simulador de Tráfico de Ciudad** demuestra cómo la aplicación consciente de los **patrones de diseño** puede generar soluciones eficientes, estructuradas y sostenibles en el desarrollo de software.

A través de la simulación del flujo vehicular en una ciudad, se implementaron con éxito los patrones **Singleton** y **Object Pool**, cuya fusión permitió mantener un equilibrio óptimo entre control centralizado y reutilización eficiente de recursos.

La implementación del **patrón Singleton** garantizó la existencia de un **único controlador de tráfico**, responsable de coordinar todos los semáforos y sensores de forma coherente y sincronizada.

Por otro lado, el **patrón Object Pool** permitió gestionar de manera eficiente los **vehículos** y **sensores**, evitando la creación constante de instancias y promoviendo un uso responsable de la memoria y del tiempo de ejecución.

La **fusión de ambos patrones** refleja buenas prácticas de diseño, como:

- **Separación clara de responsabilidades**, evitando dependencias innecesarias entre clases.
- **Reutilización controlada de objetos**, maximizando el rendimiento del sistema.
- **Centralización de la lógica de control**, asegurando consistencia y facilidad de mantenimiento.
- **Escalabilidad del código**, permitiendo ampliar la simulación sin afectar la arquitectura base.

Gracias a estas prácticas, el sistema logra una estructura sólida y extensible, demostrando que la combinación de patrones de diseño no solo mejora la eficiencia técnica, sino que también fortalece la calidad del código y la mantenibilidad del proyecto.

En conclusión, el **Simulador de Tráfico de Ciudad** no solo cumple su objetivo funcional y educativo, sino que ejemplifica el valor de aplicar **buenas prácticas de ingeniería de software** al combinar patrones de diseño de manera coherente, estratégica y sustentable.

