TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería Informática

MATERIA:
Patrones de diseño de software

TÍTULO ACTIVIDAD:

Reporte de practica

UNIDAD A EVALUAR:
2

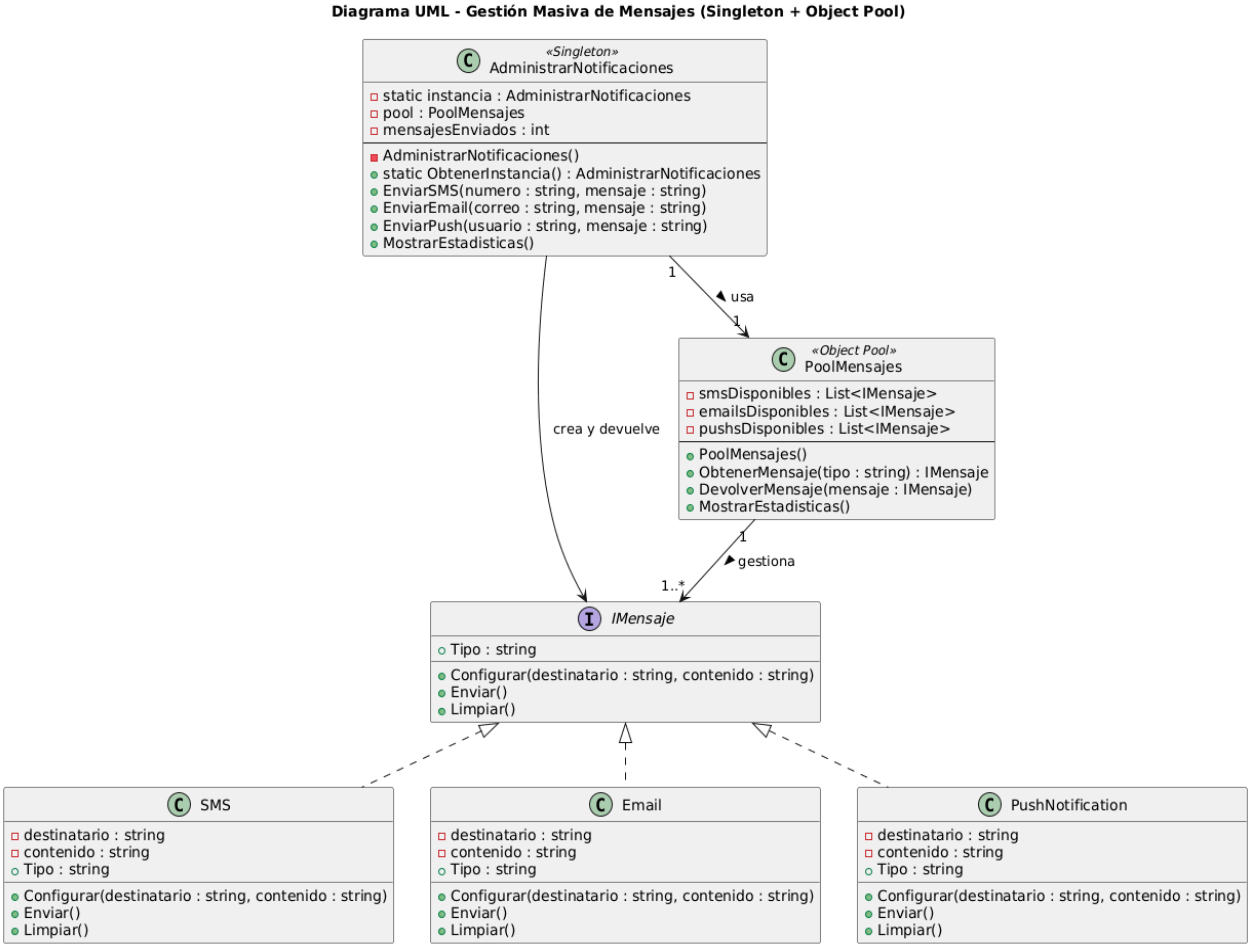
NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

Acuña Gomez Carlos Yael 21212320

NOMBRE DEL MAESTRO (A):

Maribel Guerrero Luis

Diagrama UML



Código

Program

```
internal class Program
{
    0 referencias
    static void Main(string[] args)
    {
        Console.WriteLine("=== SISTEMA DE MENSAJES ===\n");

        // Probamos que el Singleton funciona
        var admin1 = AdministrarNotificaciones.ObtenerInstancia();
        var admin2 = AdministrarNotificaciones.ObtenerInstancia();

        // Deberían ser la misma instancia
        Console.WriteLine($"¿Son la misma instancia?");
        Console.WriteLine(ReferenceEquals(admin1, admin2));

        // Enviamos algunos mensajes
        Console.WriteLine("\n--- ENVIANDO MENSAJES ---");

        admin1.EnviarSMS("+123456789", "Hola somos de telcel por favor envienos su numero de tarjeta, su fecha de vencimiento y su cvc");
        Console.WriteLine();
        admin1.EnviarEmail("usuario@email.com", "Hola le enviamos este correo porque se gano un IPHONE 17PRO no deje pasar esta oportunidad!!");
        Console.WriteLine();
        admin1.EnviarPush("dispositivo IPHONE 17 PRO", "Compra un cargador nuevo ya!!");
        Console.WriteLine();

        admin2.EnviarEmail("1234565", "");
        // Mostramos estadísticas finales
        admin1.MostrarEstadisticas();
```

```
        Console.WriteLine("\nPresiona cualquier tecla para salir...");
        Console.ReadKey();
    }
}
```

IMensaje

```
namespace ExamenU2
{
    12 referencias
    public interface IMensaje
    {
        5 referencias
        string Tipo { get; }
        6 referencias
        void Configurar(string destinatario, string contenido);
        6 referencias
        void Enviar();
        4 referencias
        void Limpiar();
    }
}
```

SMS

```
public class SMS : IMensaje
{
    3 referencias
    public string Tipo => "SMS";
    private string _destinatario;
    private string _contenido;

    4 referencias
    public void Configurar(string destinatario, string contenido)
    {
        _destinatario = destinatario;
        _contenido = contenido;
    }

    4 referencias
    public void Enviar()
    {
        Console.WriteLine($"Enviando SMS a {_destinatario}: {_contenido}");
    }

    2 referencias
    public void Limpiar()
    {
        _destinatario = "";
        _contenido = "";
    }
}
```

Email

```
public class Email : IMensaje
{
    3 referencias
    public string Tipo => "Email";
    private string _destinatario;
    private string _contenido;

    4 referencias
    public void Configurar(string destinatario, string contenido)
    {
        _destinatario = destinatario;
        _contenido = contenido;
    }

    4 referencias
    public void Enviar()
    {
        Console.WriteLine($"Enviando Email a {_destinatario}: {_contenido}");
    }

    2 referencias
    public void Limpiar()
    {
        _destinatario = "";
        _contenido = "";
    }
}
```

PushNotification

```
    public class Email : IMensaje
    {
        3 referencias
        public string Tipo => "Email";
        private string _destinatario;
        private string _contenido;

        4 referencias
        public void Configurar(string destinatario, string contenido)
        {
            _destinatario = destinatario;
            _contenido = contenido;
        }

        4 referencias
        public void Enviar()
        {
            Console.WriteLine($"Enviando Email a {_destinatario}: {_contenido}");
        }

        2 referencias
        public void Limpiar()
        {
            _destinatario = "";
            _contenido = "";
        }
    }
}
```

PoolMensajes

```
public class PoolMensajes
{
    // Listas para guardar los mensajes disponibles
    private List<IMensaje> smsDisponibles = new List<IMensaje>();
    private List<IMensaje> emailsDisponibles = new List<IMensaje>();
    private List<IMensaje> pushsDisponibles = new List<IMensaje>();

    1 referencia
    public PoolMensajes()
    {
        // Aqui podemos aumentar la cantidad inicial si se desea
        for (int i = 0; i < 1; i++)
        {
            smsDisponibles.Add(new SMS());
            emailsDisponibles.Add(new Email());
            pushsDisponibles.Add(new PushNotification());
        }
    }

    // Pedir un mensaje del pool
    3 referencias
    public IMensaje ObtenerMensaje(string tipo)
    {
        List<IMensaje> pool = null;

        // Elegir el pool correcto según el tipo
        switch (tipo.ToLower())
        {
            case "sms":
                pool = smsDisponibles;
                break;
            case "email":
                pool = emailsDisponibles;
                break;
            case "push":
                pool = pushsDisponibles;
                break;
        }
```

```
        // Si hay mensajes disponibles, tomar uno
        if (pool.Count > 0)
        {
            var mensaje = pool[0];
            pool.RemoveAt(0);
            Console.WriteLine($"Tomando {tipo} del pool");
            return mensaje;
        }
        else
        {
            // Si no hay, crear uno nuevo
            Console.WriteLine($"Creando nuevo {tipo} (pool vacío)");
            switch (tipo.ToLower())
            {
                case "sms": return new SMS();
                case "email": return new Email();
                case "push": return new PushNotification();
                default: return null;
            }
        }
    }

    // Devolver un mensaje al pool
    3 referencias
    public void DevolverMensaje(IMensaje mensaje)
    {
        // Limpiar el mensaje para reusarlo
        mensaje.Limpiar();

        // Devolverlo al pool correcto
        switch (mensaje.Tipo.ToLower())
        {
            case "sms":
                smsDisponibles.Add(mensaje);
                break;
            case "email":
                emailsDisponibles.Add(mensaje);
                break;
            case "push":
                pushsDisponibles.Add(mensaje);
                break;
        }
        Console.WriteLine($"Devolviendo {mensaje.Tipo} al pool");
    }
```

```
        // Mostrar cuantos mensajes hay disponibles
        1 referencia
        public void MostrarEstadisticas()
        {
            Console.WriteLine($"\nSMS disponibles: {smsDisponibles.Count}");
            Console.WriteLine($"Emails disponibles: {emailsDisponibles.Count}");
            Console.WriteLine($"Push disponibles: {pushsDisponibles.Count}");
        }
    }
}
```

AdministrarNotificaciones

```
public class AdministrarNotificaciones
{
    // Esta es la única instancia que existirá
    private static AdministrarNotificaciones _instancia;
    private static readonly object _lock = new object();

    private PoolMensajes _pool;
    private int _mensajesEnviados;

    1 referencia
    private AdministrarNotificaciones()
    {
        _pool = new PoolMensajes();
        _mensajesEnviados = 0;
        Console.WriteLine("Se creó el Administrador de Notificaciones");
    }

    // Método público para obtener la instancia
    2 referencias
    public static AdministrarNotificaciones ObtenerInstancia()
    {
        // Si no existe la instancia, la creamos
        lock (_lock)
        {
            if (_instancia == null)
            {
                _instancia = new AdministrarNotificaciones();
            }
            return _instancia;
        }
    }

    // Método para enviar SMS
    1 referencia
    public void EnviarSMS(string numero, string mensaje)
    {
        var sms = _pool.ObtenerMensaje("sms");
        sms.Configurar(numero, mensaje);
        sms.Enviar();
        _mensajesEnviados++;
        _pool.DevolverMensaje(sms);
    }
```

```
        // Método para enviar Email
        1 referencia
        public void EnviarEmail(string email, string mensaje)
        {
            var emailObj = _pool.ObtenerMensaje("email");
            emailObj.Configurar(email, mensaje);
            emailObj.Enviar();
            _mensajesEnviados++;
            _pool.DevolverMensaje(emailObj);
        }

        // Método para enviar Push

        public void EnviarPush(string dispositivo, string mensaje)
        {
            var push = _pool.ObtenerMensaje("push");
            push.Configurar(dispositivo, mensaje);
            push.Enviar();
            _mensajesEnviados++;
            _pool.DevolverMensaje(push);
        }


        public void MostrarEstadisticas()
        {
            Console.WriteLine($"\n=== ESTADÍSTICAS ===");
            Console.WriteLine($"Total mensajes enviados: {_mensajesEnviados}");
            _pool.MostrarEstadisticas();
        }
    }
}
```

Conclusión

La integración del patrón Singleton con el patrón Object Pool representa una práctica sólida y eficiente ya que ambos patrones se complementan para mejorar el control, la reutilización y el rendimiento de los recursos del sistema. El uso del Singleton garantiza la existencia de una única instancia global del administrador de notificaciones, centralizando la lógica de envío y evitando duplicidad de control en la gestión de mensajes. Por su parte, el Object Pool permite reutilizar objetos ya creados en lugar de instanciar nuevos de forma constante, lo cual reduce el consumo de memoria y mejora la eficiencia del programa. Al fusionar ambos patrones, se logra una arquitectura más limpia, escalable y mantenible, donde cada clase cumple una responsabilidad bien definida: el Singleton administra y coordina, mientras que el Object Pool optimiza el uso de los objetos.