Bridge Pattern - Sistema de Gestión de Notificaciones

📋 Escenario

Estás desarrollando una aplicación que gestiona la visualización de notificaciones en diferentes plataformas (por ejemplo: escritorio, móvil, web). Las notificaciones pueden ser de distintos tipos (mensaje, alerta, advertencia, confirmación) y cada tipo puede mostrarse de distintas formas según la plataforma.

⚠️ Problema

Si usas herencia tradicional, tendrías que crear clases como:

- `NotificacionMensajeWeb`
- `NotificacionAlertaWeb`
- `NotificacionMensajeMovil`
- `NotificacionAlertaMovil`
- Y muchas más combinaciones...

Esto lleva rápidamente a una **explosión combinatoria** de subclases difíciles de mantener.

🎯 Beneficios Esperados de la Solución

- **Separación de responsabilidades:** Separar la lógica de la notificación del medio por el que se presenta.
- **Escalabilidad:** Poder agregar nuevas plataformas o tipos de notificación sin modificar el resto del sistema.
- **Reducción de clases:** Evitar la multiplicación de clases para cada combinación.

- **Flexibilidad en tiempo de ejecución:** Poder cambiar la plataforma dinámicamente si es necesario.

🔍 Identificación del Patrón

Basándome en el problema descrito, el patrón más adecuado es el **Bridge Pattern (Patrón Bridge)**.

¿Por qué Bridge Pattern?

1. **Problema de explosión combinatoria:** El escenario describe exactamente el problema que resuelve Bridge - cuando tienes dos dimensiones independientes (tipo de notificación y plataforma) que pueden variar independientemente.
2. **Separación de abstracciones:** Bridge separa la abstracción (tipos de notificación) de su implementación (plataformas), permitiendo que ambas evolucionen independientemente.
3. **Flexibilidad:** Permite agregar nuevos tipos de notificación o nuevas plataformas sin modificar el código existente.

Justificación contra otros patrones:

Patrón	¿Por qué no es adecuado?
Factory Method	Solo resuelve la creación de objetos, no la estructura del problema de múltiples dimensiones.
Abstract Factory	Crearía familias de productos, pero no resuelve la separación entre abstracción e implementación.

Strategy	Se enfoca en algoritmos intercambiables, no en la separación de dimensiones independientes.
Adapter	Convierte interfaces incompatibles, no es el caso aquí.

📐 Diseño del Diagrama de Clases

El Bridge Pattern separa la abstracción de su implementación en dos jerarquías independientes:

Estructura del Patrón Bridge

Abstracción (Tipos de Notificación)

- **Notificacion:** Clase abstracta que define la interfaz común para todos los tipos de notificación
- **NotificacionMensaje:** Implementación concreta para notificaciones de mensaje
- **NotificacionAlerta:** Implementación concreta para notificaciones de alerta
- **NotificacionAdvertencia:** Implementación concreta para notificaciones de advertencia
- **NotificacionConfirmacion:** Implementación concreta para notificaciones de confirmación

Implementación (Plataformas)

- **Notificador:** Interfaz que define cómo se muestran las notificaciones en diferentes plataformas
- **NotificadorWeb:** Implementación para plataforma web
- **NotificadorMovil:** Implementación para plataforma móvil
- **NotificadorEscritorio:** Implementación para plataforma de escritorio

Ejemplo de Implementación en .NET (C#)

```
// Interfaz de implementación public interface
INotificador { void MostrarNotificacion(string
mensaje, string tipo); void SonarAlerta(); void
Vibrar(); } // Implementación para Web public class
NotificadorWeb : INotificador { public void
MostrarNotificacion(string mensaje, string tipo) {
Console.WriteLine($"Web: Mostrando {tipo}:
{mensaje}"); // Aquí podrías integrar con JavaScript
para mostrar notificaciones del navegador } public
void SonarAlerta() { Console.WriteLine("Web:
Reproduciendo sonido de alerta"); // Integración con
Audio API del navegador } public void Vibrar() {
Console.WriteLine("Web: No se puede vibrar en web");
} } // Implementación para Móvil (Xamarin/MAUI)
public class NotificadorMovil : INotificador { public
void MostrarNotificacion(string mensaje, string tipo)
{ Console.WriteLine($"Móvil: Mostrando {tipo}:
{mensaje}"); // Implementación específica para
dispositivos móviles } public void SonarAlerta() {
Console.WriteLine("Móvil: Reproduciendo sonido de
alerta"); // Uso de APIs nativas del dispositivo }
public void Vibrar() { Console.WriteLine("Móvil:
Vibrando dispositivo"); // Implementación de
vibración nativa } } // Implementación para
Escritorio (WPF/WinForms) public class
NotificadorEscritorio : INotificador { public void
MostrarNotificacion(string mensaje, string tipo) {
Console.WriteLine($"Escritorio: Mostrando {tipo}:
{mensaje}"); // Implementación con MessageBox o
notificaciones del sistema } public void
SonarAlerta() { Console.WriteLine("Escritorio:
Reproduciendo sonido de alerta"); // Uso de
SystemSounds } public void Vibrar() {
Console.WriteLine("Escritorio: No se puede vibrar en
escritorio"); } } // Abstracción base public abstract
class Notificacion { protected readonly INotificador
_notificador; protected Notificacion(INotificador
notificador) { _notificador = notificador ?? throw
new ArgumentNullException(nameof(notificador)); }
public abstract void Mostrar(); public abstract void
Enviar(); // Método para cambiar el notificador
dinámicamente public void
CambiarNotificador(INotificador nuevoNotificador) {
if (nuevoNotificador != null) { // En una
implementación real, esto requeriría una propiedad
protegida // o un método para reemplazar el
```

```
notificador Console.WriteLine("Notificador cambiado
dinámicamente"); } } } // Implementación concreta de
notificación de mensaje public class
NotificacionMensaje : Notificacion { private readonly
string _mensaje; public
NotificacionMensaje(INotificador notificador, string
mensaje) : base(notificador) { _mensaje = mensaje ??
throw new ArgumentNullException(nameof(mensaje)); }
public override void Mostrar() {
_notificador.MostrarNotificacion(_mensaje,
"MENSAJE"); } public override void Enviar() {
Console.WriteLine($"Enviando mensaje: {_mensaje}");
Mostrar(); } } // Implementación concreta de
notificación de alerta public class
NotificacionAlerta : Notificacion { private readonly
string _mensaje; private readonly string
_nivelCriticidad; public
NotificacionAlerta(INotificador notificador, string
mensaje, string nivelCriticidad = "MEDIA") :
base(notificador) { _mensaje = mensaje ?? throw new
ArgumentNullException(nameof(mensaje));
_nivelCriticidad = nivelCriticidad; } public override
void Mostrar() {
_notificador.MostrarNotificacion(_mensaje, $"ALERTA
({_nivelCriticidad})"); _notificador.SonarAlerta(); }
public override void Enviar() {
Console.WriteLine($"Enviando alerta: {_mensaje}");
Mostrar(); } } // Implementación concreta de
notificación de confirmación public class
NotificacionConfirmacion : Notificacion { private
readonly string _mensaje; private readonly string
_accionConfirmar; public
NotificacionConfirmacion(INotificador notificador,
string mensaje, string accionConfirmar) :
base(notificador) { _mensaje = mensaje ?? throw new
ArgumentNullException(nameof(mensaje));
_accionConfirmar = accionConfirmar ?? throw new
ArgumentNullException(nameof(accionConfirmar)); }
public override void Mostrar() {
_notificador.MostrarNotificacion(_mensaje,
"CONFIRMACIÓN"); Console.WriteLine($"Acción a
confirmar: {_accionConfirmar}"); } public override
void Enviar() { Console.WriteLine($"Solicitando
confirmación: {_mensaje}"); Mostrar(); } }
```

Ejemplo de Uso en .NET

```
public class Program { public static void
Main(string[] args) { Console.WriteLine("===
Demostración del Bridge Pattern ===\n"); // Crear
notificadores para diferentes plataformas
INotificador notificadorWeb = new NotificadorWeb();
INotificador notificadorMovil = new
NotificadorMovil(); INotificador
notificadorEscritorio = new NotificadorEscritorio();
// Crear diferentes tipos de notificaciones para Web
Console.WriteLine("--- Notificaciones Web ---"); var
mensajeWeb = new NotificacionMensaje(notificadorWeb,
"Tienes un nuevo mensaje"); mensajeWeb.Enviar(); var
alertaWeb = new NotificacionAlerta(notificadorWeb,
"Sistema en mantenimiento", "ALTA");
alertaWeb.Enviar(); var confirmacionWeb = new
NotificacionConfirmacion(notificadorWeb, "¿Desea
continuar con la operación?", "Eliminar archivo");
confirmacionWeb.Enviar(); Console.WriteLine("\n---
Notificaciones Móvil ---"); // Crear las mismas
notificaciones para Móvil var mensajeMovil = new
NotificacionMensaje(notificadorMovil, "Tienes un
nuevo mensaje"); mensajeMovil.Enviar(); var
alertaMovil = new
NotificacionAlerta(notificadorMovil, "Sistema en
mantenimiento", "ALTA"); alertaMovil.Enviar();
Console.WriteLine("\n--- Notificaciones Escritorio --
-"); // Crear las mismas notificaciones para
Escritorio var mensajeEscritorio = new
NotificacionMensaje(notificadorEscritorio, "Tienes un
nuevo mensaje"); mensajeEscritorio.Enviar(); var
alertaEscritorio = new
NotificacionAlerta(notificadorEscritorio, "Sistema en
mantenimiento", "ALTA"); alertaEscritorio.Enviar();
Console.WriteLine("\n=== Fin de la demostración
==="); } } // Ejemplo de uso con Dependency Injection
(ASP.NET Core) public class NotificacionService {
private readonly INotificador _notificador; public
NotificacionService(INotificador notificador) {
_notificador = notificador; } public void
EnviarMensaje(string contenido) { var notificacion =
new NotificacionMensaje(_notificador, contenido);
notificacion.Enviar(); } public void
EnviarAlerta(string mensaje, string nivel = "MEDIA")
{ var notificacion = new
NotificacionAlerta(_notificador, mensaje, nivel);
notificacion.Enviar(); } } // Configuración en
```

```
Program.cs (ASP.NET Core) /* var builder =
WebApplication.CreateBuilder(args); // Registrar el
notificador según la plataforma if
(builder.Environment.IsWeb()) {
builder.Services.AddScoped(); } else if
(builder.Environment.IsMobile()) {
builder.Services.AddScoped(); } else {
builder.Services.AddScoped(); }
builder.Services.AddScoped(); var app =
builder.Build(); */
```

Características Específicas de .NET

```
// Uso de enums para tipos de notificación public
enum TipoNotificacion { Mensaje, Alerta, Advertencia,
Confirmacion } public enum NivelCriticidad { Baja,
Media, Alta, Critica } // Implementación mejorada con
enums public class NotificacionAlerta : Notificacion
{ private readonly string _mensaje; private readonly
NivelCriticidad _nivelCriticidad; public
NotificacionAlerta(INotificador notificador, string
mensaje, NivelCriticidad nivelCriticidad =
NivelCriticidad.Media) : base(notificador) { _mensaje
= mensaje ?? throw new
ArgumentNullException(nameof(mensaje));
_nivelCriticidad = nivelCriticidad; } public override
void Mostrar() {
_notificador.MostrarNotificacion(_mensaje, $"ALERTA
({_nivelCriticidad})"); _notificador.SonarAlerta(); }
public override void Enviar() {
Console.WriteLine($"Enviando alerta de nivel
{_nivelCriticidad}: {_mensaje}"); Mostrar(); } } //
Uso de async/await para operaciones asíncronas public
interface INotificadorAsync { Task
MostrarNotificacionAsync(string mensaje, string
tipo); Task SonarAlertaAsync(); Task VibrarAsync(); }
// Implementación asíncrona public class
NotificadorWebAsync : INotificadorAsync { public
async Task MostrarNotificacionAsync(string mensaje,
string tipo) { Console.WriteLine($"Web: Mostrando
{tipo}: {mensaje}"); // Simular operación asíncrona
(ej: llamada a API) await Task.Delay(100); } public
async Task SonarAlertaAsync() {
Console.WriteLine("Web: Reproduciendo sonido de
alerta"); await Task.Delay(50); } public async Task
```

```
VibrarAsync() { Console.WriteLine("Web: No se puede
vibrar en web"); await Task.CompletedTask; } } // Uso
de records para datos inmutables public record
NotificacionData(string Mensaje, TipoNotificacion
Tipo, DateTime Timestamp) { public static
NotificacionData Crear(string mensaje,
TipoNotificacion tipo) => new(mensaje, tipo,
DateTime.UtcNow); } // Factory para crear
notificaciones public static class
NotificacionFactory { public static Notificacion
Crear(TipoNotificacion tipo, INotificador
notificador, string mensaje, string? accionExtra =
null) { return tipo switch { TipoNotificacion.Mensaje
=> new NotificacionMensaje(notificador, mensaje),
TipoNotificacion.Alerta => new
NotificacionAlerta(notificador, mensaje),
TipoNotificacion.Confirmacion => new
NotificacionConfirmacion(notificador, mensaje,
accionExtra ?? ""), _ => throw new
ArgumentException($"Tipo de notificación no
soportado: {tipo}") }; } }
```

Ejemplo con Dependency Injection y Configuration

```
// Configuración en appsettings.json /* {
"Notificaciones": { "Plataforma": "Web",
"Configuracion": { "SonarAlertas": true,
"VibrarEnMovil": true, "TimeoutSegundos": 30 } } } */
// Clase de configuración public class
NotificacionConfig { public string Plataforma { get;
set; } = "Web"; public bool SonarAlertas { get; set;
} = true; public bool VibrarEnMovil { get; set; } =
true; public int TimeoutSegundos { get; set; } = 30;
} // Servicio de notificaciones con configuración
public class NotificacionServiceConfigurado { private
readonly INotificador _notificador; private readonly
NotificacionConfig _config; public
NotificacionServiceConfigurado(INotificador
notificador, IOptions config) { _notificador =
notificador; _config = config.Value; } public async
Task EnviarNotificacionAsync(string mensaje,
TipoNotificacion tipo) { var notificacion =
NotificacionFactory.Crear(tipo, _notificador,
mensaje); // Aplicar configuración if
(_config.SonarAlertas && tipo ==
```

```
TipoNotificacion.Alerta) { if (_notificador is
INotificadorAsync notificadorAsync) { await
notificadorAsync.SonarAlertaAsync(); } }
notificacion.Enviar(); } } // Registro en Program.cs
/* var builder = WebApplication.CreateBuilder(args);
// Configurar opciones builder.Services.Configure(
builder.Configuration.GetSection("Notificaciones"));
// Registrar notificador según configuración var
config =
builder.Configuration.GetSection("Notificaciones:Plat
aforma").Value; switch (config?.ToLower()) { case
"web": builder.Services.AddScoped(); break; case
"movil": builder.Services.AddScoped(); break; case
"escritorio": builder.Services.AddScoped(); break;
default: builder.Services.AddScoped(); break; }
builder.Services.AddScoped(); var app =
builder.Build(); */
```

☑ Beneficios de este Diseño

1. **Escalabilidad:** Agregar un nuevo tipo de notificación solo requiere crear una nueva subclase de `Notificacion`
2. **Extensibilidad:** Agregar una nueva plataforma solo requiere implementar la interfaz `Notificador`
3. **Separación de responsabilidades:** La lógica de la notificación está separada de la lógica de presentación
4. **Flexibilidad:** Se puede cambiar la plataforma dinámicamente en tiempo de ejecución
5. **Mantenibilidad:** Cada clase tiene una responsabilidad específica y bien definida
6. **Principio Abierto/Cerrado:** El sistema está abierto para extensión pero cerrado para modificación

📊 Comparación: Antes vs Después

❌ Sin Bridge Pattern (Herencia Tradicional)

- Necesitaríamos crear clases como: `NotificacionMensajeWeb`, `NotificacionAlertaWeb`, `NotificacionMensajeMovil`, `NotificacionAlertaMovil`, etc.
- Si tenemos 4 tipos de notificación y 3 plataformas = 12 clases diferentes
- Si agregamos 1 tipo más = 3 clases nuevas
- Si agregamos 1 plataforma más = 4 clases nuevas
- **Total: 20 clases para 5 tipos y 4 plataformas**

☑ Con Bridge Pattern

- 4 clases para tipos de notificación + 3 clases para plataformas + 1 interfaz = **8 clases totales**
- Si agregamos 1 tipo más = 1 clase nueva
- Si agregamos 1 plataforma más = 1 clase nueva
- **Total: 10 clases para 5 tipos y 4 plataformas**

🎯 Conclusión

El **Bridge Pattern** es perfecto para este escenario porque:

- Resuelve el problema de explosión combinatoria entre tipos de notificación y plataformas
- Separa la abstracción (tipos de notificación) de su implementación (plataformas)
- Permite que ambas dimensiones evolucionen independientemente
- Facilita la extensión del sistema sin modificar código existente
- Reduce significativamente el número de clases necesarias
- Mejora la mantenibilidad y flexibilidad del código

Esta estructura permite agregar fácilmente nuevos tipos de notificación o nuevas plataformas sin afectar el resto del sistema, cumpliendo perfectamente con los beneficios esperados mencionados en el problema original.

Documento generado para el análisis del patrón Bridge aplicado a un sistema de gestión de notificaciones multi-plataforma.