TECNOLÓGICO NACIONAL DE MÉXICO

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

INSTITUTO TECNOLÓGICO DEL VALLE DE OAXACA

INTITITO TECNOLOGICO DEL VALLE DE OAXACA

Plataforma Académica Modular

DESARROLLO BACK-END

DOCENTE:
Noé Gutiérrez Osorno

Trabajo echo por:
HONORIO JUAREZ ZURITA
CESAR ADAIR MARTINEZ PARRA

CARRERA:
INGENIERIA INFORMATICA

SEMESTRE
7° “A”

NAZARENO, XOXOCOTLÁN, OAXACA MÉXICO
06 DE DICIEMBRE DEL 2025

INDICE

INTRODUCCION

En el entorno educativo actual, la gestión eficiente de la información académica es crucial para el éxito de las instituciones. Los sistemas tradicionales a menudo sufren de rigidez, dificultad de mantenimiento y falta de escalabilidad, lo que resulta en procesos manuales propensos a errores y una experiencia de usuario deficiente. La ingeniería de software moderna ofrece soluciones a estos problemas a través de la aplicación de Patrones de Diseño, que son soluciones probadas a problemas recurrentes en el desarrollo de software.

Este proyecto, denominado "EduBlocs Plataforma Académica Modular", nace como una respuesta a la necesidad de un sistema robusto, flexible y mantenible. A diferencia de un desarrollo monolítico convencional, este sistema se ha construido desde cero integrando **23 Patrones de diseño**. Esta arquitectura permite una clara separación de responsabilidades, facilitando la incorporación de nuevos módulos sin afectar el funcionamiento existente.

El sistema simula un entorno académico completo donde interactúan tres roles fundamentales: Administradores, Profesores y Estudiantes. Cada rol cuenta con funcionalidades específicas diseñadas para optimizar sus tareas diarias, desde la gestión de usuarios y cursos hasta la entrega de actividades y el control de asistencia. La implementación en C# .NET 8.0 asegura un rendimiento óptimo y aprovecha las características modernas del lenguaje para una implementación limpia y eficiente de los patrones.

A lo largo de este documento, se detallará cómo la aplicación de principios de diseño avanzado transforma requerimientos complejos en una arquitectura de software elegante y sostenible, demostrando que la calidad del código es tan importante como la funcionalidad que ofrece.

3. OBJETIVO GENERAL

Transformar la gestión operativa y académica de las instituciones educativas mediante el desarrollo e implementación de un sistema de Gestión Académica que integre y demuestre la aplicación de 23 patrones de diseño (Creacionales, Estructurales y de Comportamiento), garantizando una arquitectura modular que garantiza la eficiencia administrativa, la toma de decisiones basadas en datos unificados y la optimización de los costos operativos

4. OBJETIVOS ESPECIFICOS

1. **Implementar los 5 Patrones Creacionales** para gestionar eficientemente la instanciación de objetos complejos como usuarios, cursos y contenidos educativos.
2. **Aplicar los 7 Patrones Estructurales** para desacoplar interfaces de implementaciones y optimizar la composición de objetos, mejorando la flexibilidad del sistema.
3. **Integrar los 11 Patrones de Comportamiento** para gestionar la comunicación entre objetos, el flujo de control y las responsabilidades del sistema de manera efectiva.
4. **Desarrollar módulos funcionales** para tres roles distintos (Administrador, Profesor, Estudiante) que cubran el ciclo de vida académico completo.
5. **Simular un entorno de producción** con persistencia en memoria, autenticación segura y una interfaz de usuario por consola interactiva y amigable.

5. JUSTIFICACION

Se justifica por la urgente necesidad del sector educativo de superar el caos administrativo y la ineficiencia operativa causado por sistemas fragmentados u obsoletos o desactualizados
La elección de basar este proyecto en los Patrones de Diseño se justifica por:

- **Mantenibilidad:** Los patrones como Factory Method o Strategy permiten cambiar comportamientos o añadir nuevos tipos de objetos sin modificar el código cliente existente (Principio Open/Closed).
- **Escalabilidad:** Patrones como Observer y Mediator desacoplan los componentes, permitiendo que el sistema crezca en complejidad sin aumentar exponencialmente la dependencia entre módulos.
- **Reusabilidad:** El uso de Templates y Builders fomenta la reutilización de código, reduciendo errores y tiempos de desarrollo.
- **Estandarización:** Al usar un vocabulario común (Singleton, Facade, Adapter), se facilita la comunicación entre desarrolladores y la comprensión del sistema.

Este proyecto no solo resuelve una necesidad funcional (gestión académica), sino que sirve como una prueba de concepto de cómo la ingeniería de software profesional puede elevar la calidad del producto final.

6. DIAGRAMA DE CLASES

Diagrama sin título.drawio - draw.io

7. DIAGRAMA DE ESTRUCTURA DEL PROYECTO

```
PlataformaAcademicaModular/
├── Program.cs (Entry Point & UI Logic)
├── UserManagement/ (Gestión de Usuarios)
│   ├── UserSession.cs [Singleton]
│   ├── UserFactory.cs [Factory Method]
│   ├── UserProfile.cs [Prototype]
│   └── AcademicSystemFacade.cs [Facade]
├── CourseBuilder/ (Gestión de Cursos)
│   ├── CourseBuilder.cs [Builder]
│   ├── CourseManager.cs [Singleton]
│   └── ContentFactory.cs [Abstract Factory]
├── BehaviorExtras/ (Lógica de Negocio)
│   ├── GradingStrategy.cs [Strategy]
│   ├── AssessmentTemplate.cs [Template Method]
│   ├── DataVisitor.cs [Visitor]
│   └── Interpreter.cs [Interpreter]
├── NotificationCenter/ (Comunicaciones)
│   ├── NotificationService.cs [Observer]
│   ├── NotificationCommand.cs [Command]
│   └── NotificationMediator.cs [Mediator]
├── UIAdapter/ (Interfaz de Usuario)
│   ├── ConsoleAdapter.cs [Adapter]
│   ├── UIDecorator.cs [Decorator]
│   └── UIBridge.cs [Bridge]
└── ResourceOptimizer/ (Optimización)
    ├── ObjectPool.cs [Object Pool]
    └── ResourceFlyweight.cs [Flyweight]
```

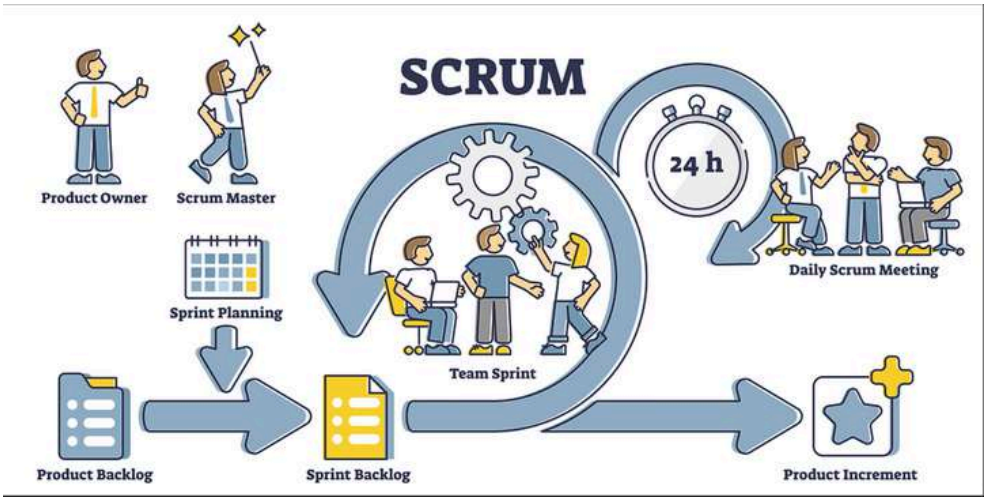
8. METODOLOGIA DE DESARROLLO

porque esta metodología ágil es ideal para gestionar la complejidad y los cambios inherentes a una plataforma modular de gestión académica.

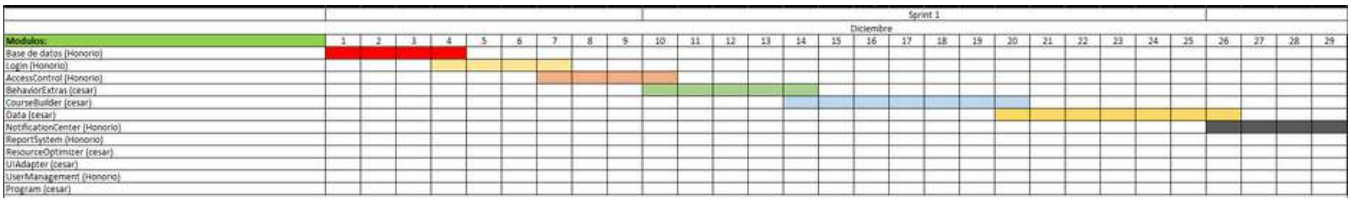
1.Entrega Rápida de Valor: Permite entregar módulos funcionales en ciclos cortos (Sprints), garantizando que los stakeholders reciban valor tempranamente.

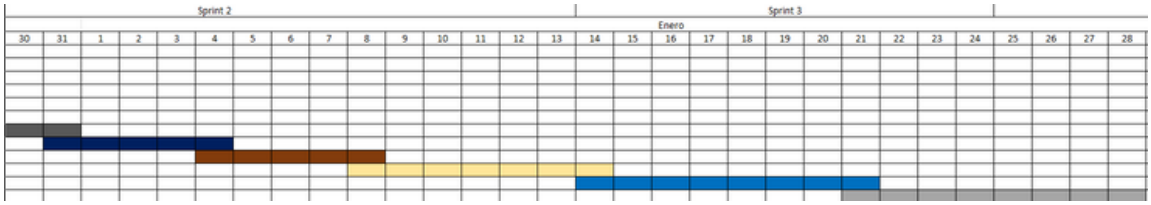
2.Adaptabilidad: Facilita la incorporación de feedback y la adaptación a nuevos requisitos de usuarios (docentes, alumnos, administradores) de manera eficiente.

3.Reducción de Riesgos: La inspección y adaptación constante detectan y corrigen problemas o desviaciones en etapas tempranas.



a. Calendario de Entregas

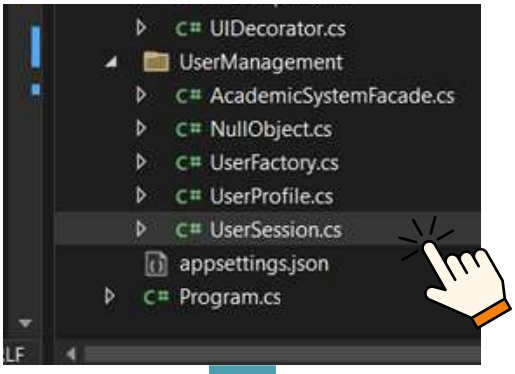




b. Aplicación de Patrones por Sección

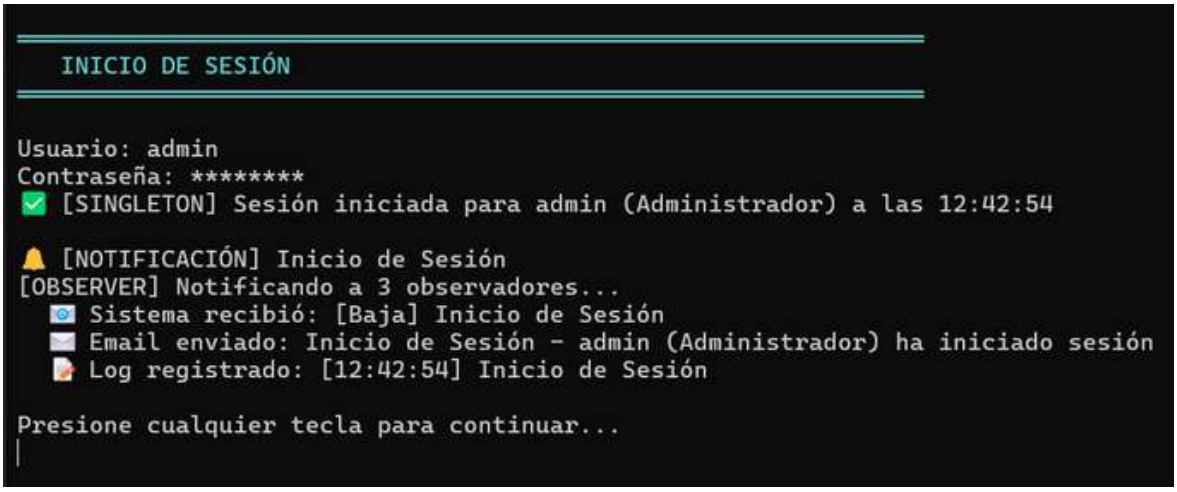
1. Gestion de Usuarios y Acceso

Singleton: **UserSession** garantiza que solo exista una sesion activa y un unico punto de acceso all registro de usuario




```
namespace PlataformaAcademicaModular.UserManagement;

/// <summary>
/// PATRÓN SINGLETON: Garantiza una única instancia de sesión en toda la aplicación
/// Thread-safe con inicialización perezosa
/// FUNCIONAL: Gestiona registro, login y persistencia de usuarios
/// </summary>
35 referencias
public sealed class UserSession
{
    private static readonly Lazy<UserSession> _instance = new(() => new UserSession());

    // Persistencia en memoria
    private static readonly Dictionary<string, UserCredentials> _userRegistry = new();
    private static readonly List<IUser> _allUsers = new();

    private IUser? _currentUser;
    private DateTime _loginTime;

    1 referencia
    private UserSession()
    {
        // Crear usuarios por defecto
        RegisterDefaultUsers();
    }

    20 referencias
    public static UserSession Instance => _instance.Value;

    /// <summary>
    /// Registra un nuevo usuario en el sistema
    /// </summary>
    8 referencias
    public bool Register(string username, string password, string role, string additionalInfo)
    {
        if (string.IsNullOrWhiteSpace(username) || string.IsNullOrWhiteSpace(password))
        {
            Console.WriteLine("❌ [SINGLETON] Usuario o contraseña no pueden estar vacíos");
            return false;
        }

        if (_userRegistry.ContainsKey(username))
        {
            Console.WriteLine($"❌ [SINGLETON] El usuario '{username}' ya existe");
            return false;
        }

        // Crear usuario usando Factory
```

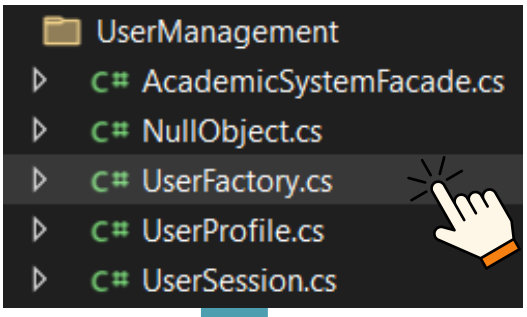
No se encontraron problemas.



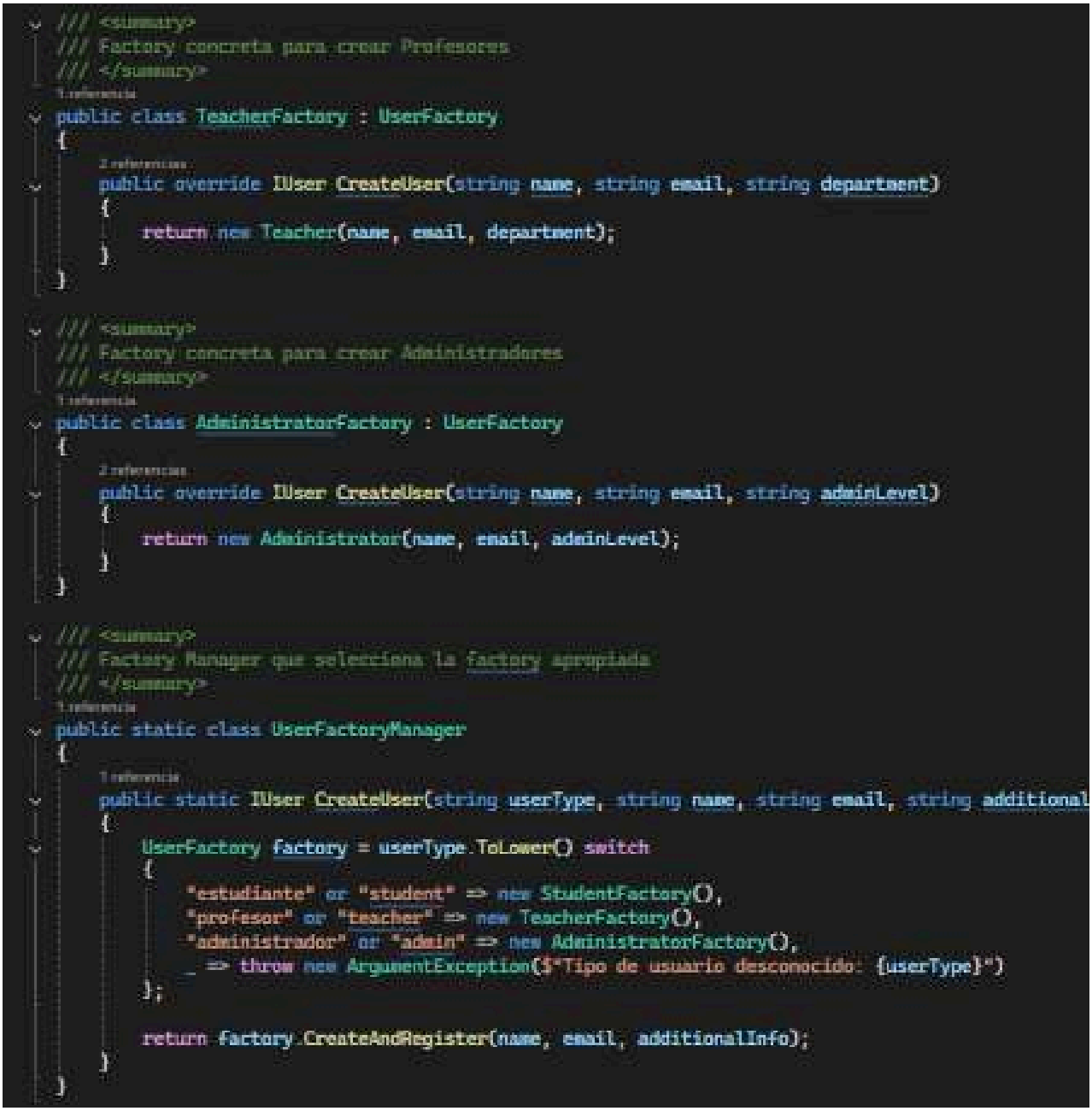
Se creo una única instancia de sesión para el usuario administrador

Se destruye o invalida la instancia única, el Sistema vuelve a estar listo para una nueva sesión

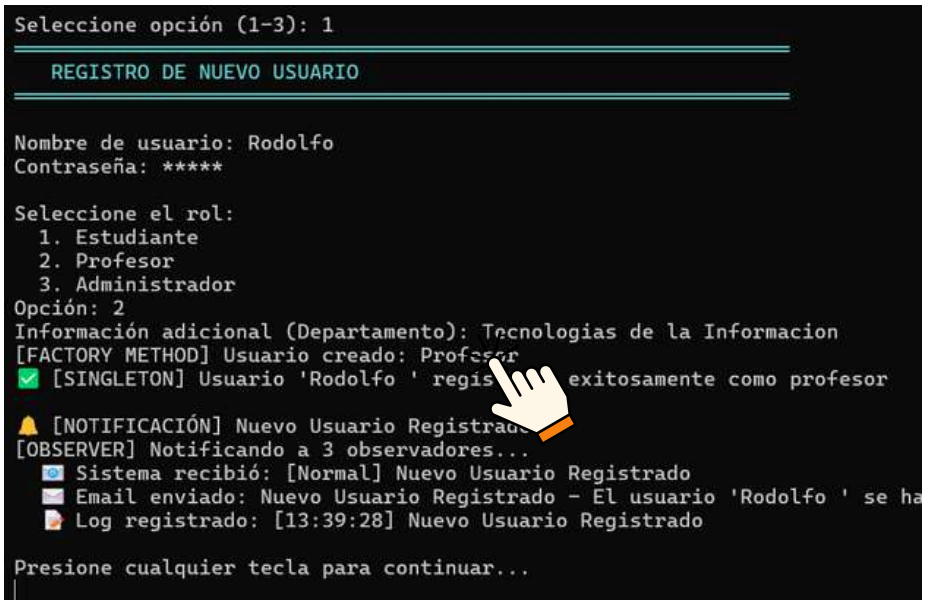
Factory Method: **UserFactory** encapsula la lógica de creación de **Student , Teacher** y **Administrador** permitiendo añadir nuevos roles fácilmente




```
namespace PlataformaAcademicaModular.UserManagement;

/// <summary>
/// PATRÓN FACTORY METHOD: Crea diferentes tipos de usuarios sin exponer la lógica de creación
/// Permite agregar nuevos tipos de usuarios sin modificar el código cliente
/// </summary>
public abstract class UserFactory
{
    public abstract IUser CreateUser(string name, string email, string additionalInfo);

    public IUser CreateAndRegister(string name, string email, string additionalInfo)
    {
        var user = CreateUser(name, email, additionalInfo);
        Console.WriteLine($"[FACTORY METHOD] Usuario creado: {user.Role}");
        return user;
    }
}

/// <summary>
/// Factory concreta para crear Estudiantes
/// </summary>
public class StudentFactory : UserFactory
{
    public override IUser CreateUser(string name, string email, string studentId)
    {
        return new Student(name, email, studentId);
    }
}
```

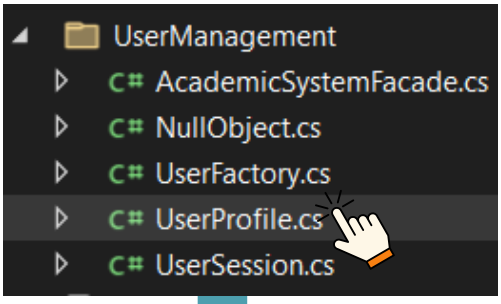


```
/// <summary>
/// Factory concreta para crear Profesores
/// </summary>
1 referencia
public class TeacherFactory : UserFactory
{
    2 referencias
    public override IUser CreateUser(string name, string email, string department)
    {
        return new Teacher(name, email, department);
    }
}

/// <summary>
/// Factory concreta para crear Administradores
/// </summary>
1 referencia
public class AdministratorFactory : UserFactory
{
    2 referencias
    public override IUser CreateUser(string name, string email, string adminLevel)
    {
        return new Administrator(name, email, adminLevel);
    }
}

/// <summary>
/// Factory Manager que selecciona la factory apropiada
/// </summary>
1 referencia
public static class UserFactoryManager
{
    1 referencia
    public static IUser CreateUser(string userType, string name, string email, string additional
    {
        UserFactory factory = userType.ToLower() switch
        {
            "estudiante" or "student" => new StudentFactory(),
            "profesor" or "teacher" => new TeacherFactory(),
            "administrador" or "admin" => new AdministratorFactory(),
            _ => throw new ArgumentException($"Tipo de usuario desconocido: {userType}")
        };

        return factory.CreateAndRegister(name, email, additionalInfo);
    }
}
```



Esto indica que **UserFactory** ha instanciado un objeto de tipo Profesor , encapsulando la lógica de creación.

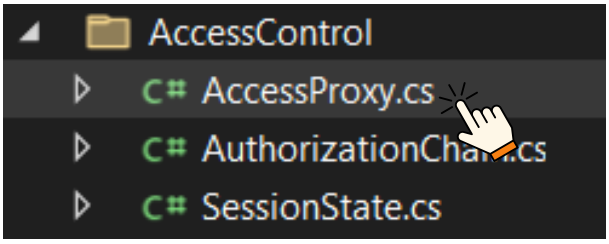
Prototype: **UserPorfile** permite clonar configuraciones de usuarios predeterminadas para agilizar el registro




```
namespace PlataformaAcademicaModular.UserManagement;

/// <summary>
/// PATRÓN PROTOTYPE: Permite clonar perfiles de usuario para crear plantillas
/// Útil para crear perfiles base que pueden ser personalizados
/// </summary>
8 referencias
public class UserProfile : ICloneable
{
    5 referencias
    public string Name { get; set; }
    2 referencias
    public string Email { get; set; }
    2 referencias
    public string Role { get; set; }
    8 referencias
    public Dictionary<string, string> Preferences { get; set; }
    6 referencias
    public List<string> Permissions { get; set; }
```

```
    2 referencias
    public UserProfile(string name, string email, string role)
    {
        Name = name;
        Email = email;
        Role = role;
        Preferences = new Dictionary<string, string>();
        Permissions = new List<string>();
    }

    /// <summary>
    /// Clonación superficial
    /// </summary>
    0 referencias
    public object Clone()
    {
        Console.WriteLine($"[PROTOTYPE] Clonando perfil de {Name}");
        return MemberwiseClone();
    }

    /// <summary>
    /// Clonación profunda para copiar colecciones
    /// </summary>
    1 referencia
    public UserProfile DeepClone()
    {
        Console.WriteLine($"[PROTOTYPE] Clonación profunda del perfil de {Name}");
        var clone = (UserProfile)MemberwiseClone();
        clone.Preferences = new Dictionary<string, string>(Preferences);
        clone.Permissions = new List<string>(Permissions);
        return clone;
    }

    1 referencia
    public void DisplayProfile()
    {
        Console.WriteLine($"  Perfil: {Name} ({Role})");
        Console.WriteLine($"  Email: {Email}");
        Console.WriteLine($"  Permisos: {string.Join(", ", Permissions)}");
        Console.WriteLine($"  Preferencias: {Preferences.Count} configuradas");
    }
}

/// <summary>
/// Registro de prototipos predefinidos
/// </summary>
2 referencias
public class ProfilePrototypeRegistry
{
    private readonly Dictionary<string, UserProfile> _prototypes = new();

    1 referencia
    public ProfilePrototypeRegistry()
    {
        // Prototipos predefinidos
        var studentTemplate = new UserProfile("Estudiante Plantilla", "student@template.com", "Estudiante");
        studentTemplate.Permissions.AddRange(new[] { "Ver Cursos", "Enviar Tareas", "Ver Calificaciones" });
        studentTemplate.Preferences["Theme"] = "Light";
        studentTemplate.Preferences["Language"] = "ES";

        var teacherTemplate = new UserProfile("Profesor Plantilla", "teacher@template.com", "Profesor");
        teacherTemplate.Permissions.AddRange(new[] { "Crear Cursos", "Calificar", "Ver Reportes", "Gestionar Alumno
        teacherTemplate.Preferences["Theme"] = "Dark";
        teacherTemplate.Preferences["Language"] = "ES";

        _prototypes["student"] = studentTemplate;
        _prototypes["teacher"] = teacherTemplate;
    }

    1 referencia
    public UserProfile GetPrototype(string key)
    {
        if (_prototypes.TryGetValue(key, out var prototype))
        {
            return prototype.DeepClone();
        }
        throw new KeyNotFoundException($"Prototipo '{key}' no encontrado");
    }

    0 referencias
    public void RegisterPrototype(string key, UserProfile prototype)
    {
        _prototypes[key] = prototype;
        Console.WriteLine($"[PROTOTYPE] Prototipo '{key}' registrado");
    }
}
```

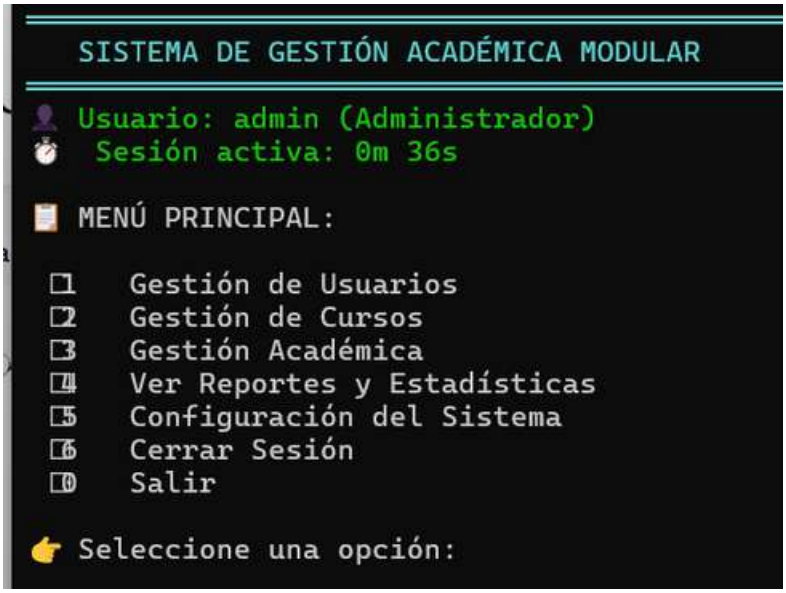
Proxy: **AccessProxy** controla el acceso a objetos sensibles, cargándolos solo cuando el usuario tiene los permisos adecuados.



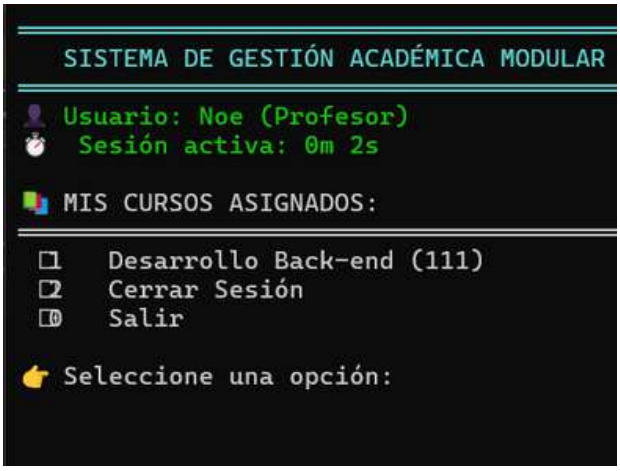

```
namespace PlataformaAcademicaModular.AccessControl;

/// <summary>
/// Interfaz para recursos protegidos
/// </summary>
2 referencias
public interface ISecureResource
{
    3 referencias
    string GetData();
    3 referencias
    void ModifyData(string newData);
}

/// <summary>
/// Recurso real que contiene información sensible
/// </summary>
3 referencias
public class SecureDatabase : ISecureResource
{
    private string _data = "Información Confidencial del Sistema";

    2 referencias
    public string GetData()
    {
        Console.WriteLine("  [REAL] Accediendo a base de datos segura...");
        return _data;
    }

    2 referencias
    public void ModifyData(string newData)
    {
        Console.WriteLine("  [REAL] Modificando datos en base de datos segura...");
        _data = newData;
    }
}
```

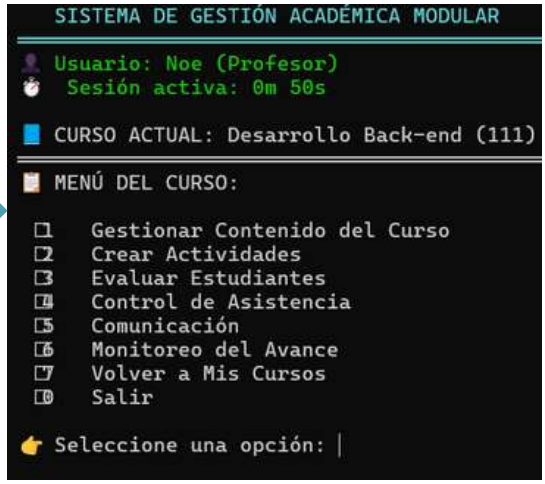
```
/// <summary>
/// PATRÓN PROXY: Controla el acceso a un objeto
/// Proporciona lazy loading, control de acceso y logging
/// </summary>
1 referencia
public class AccessProxy : ISecureResource
{
    private SecureDatabase? _realDatabase;
    private readonly string _userRole;
    private readonly List<string> _accessLog = new();

    0 referencias
    public AccessProxy(string userRole)
    {
        _userRole = userRole;
        Console.WriteLine($"[PROXY] Proxy creado para rol: {userRole}");
    }

    1 referencia
    public string GetData()
    {
        LogAccess("GetData");

        if (!HasPermission("Read"))
        {
            Console.WriteLine("[PROXY] ❌ Acceso denegado: permisos insuficientes");
            return "ACCESO DENEGADO";
        }

        // Lazy initialization
        if (_realDatabase == null)
        {
            Console.WriteLine("[PROXY] Inicializando base de datos (lazy loading)...");
            _realDatabase = new SecureDatabase();
        }

        Console.WriteLine("[PROXY] ☑ Acceso concedido");
        return _realDatabase.GetData();
    }

    1 referencia
    public void ModifyData(string newData)
    {
        LogAccess("ModifyData");

        if (!HasPermission("Write"))
        {
            Console.WriteLine("[PROXY] ❌ Modificación denegada: permisos insuficientes");
            return;
        }

        if (_realDatabase == null)
        {
            _realDatabase = new SecureDatabase();
        }

        Console.WriteLine("[PROXY] ☑ Modificación permitida");
        _realDatabase.ModifyData(newData);
    }
```

```
2 referencias
private bool HasPermission(string operation)
{
    return _userRole switch
    {
        "Administrador" => true,
        "Profesor" => operation == "Read",
        "Estudiante" => false,
        _ => false
    };
}

2 referencias
private void LogAccess(string operation)
{
    var logEntry = $"[{DateTime.Now:HH:mm:ss}] {_userRole} intentó {operation}";
    _accessLog.Add(logEntry);
    Console.WriteLine($"[PROXY] Log: {logEntry}");
}

0 referencias
public void ShowAccessLog()
{
    Console.WriteLine($"[PROXY] Registro de accesos ({_accessLog.Count}):");
    foreach (var log in _accessLog)
    {
        Console.WriteLine($"  {log}");
    }
}
}
```

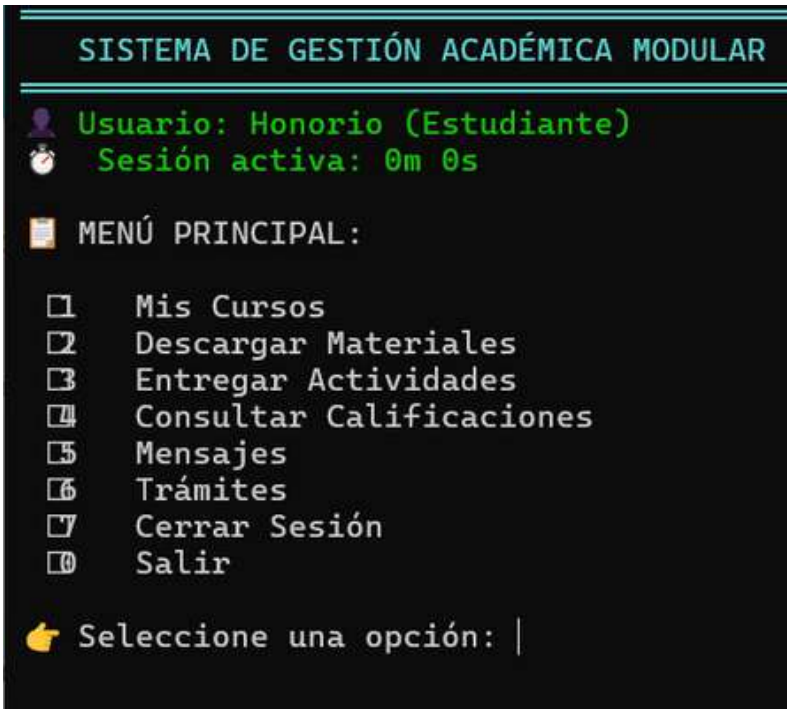


Administrador: Accede a todo el sistema (gestión de usuarios, cursos, reportes, configuración)



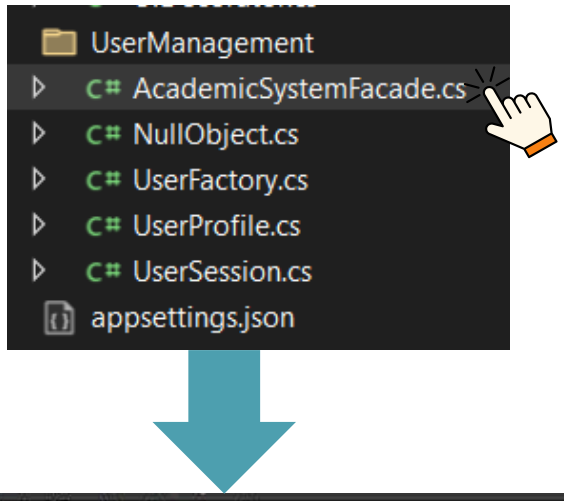


Profesor: Accede solo a sus cursos asignados y funciones docentes.

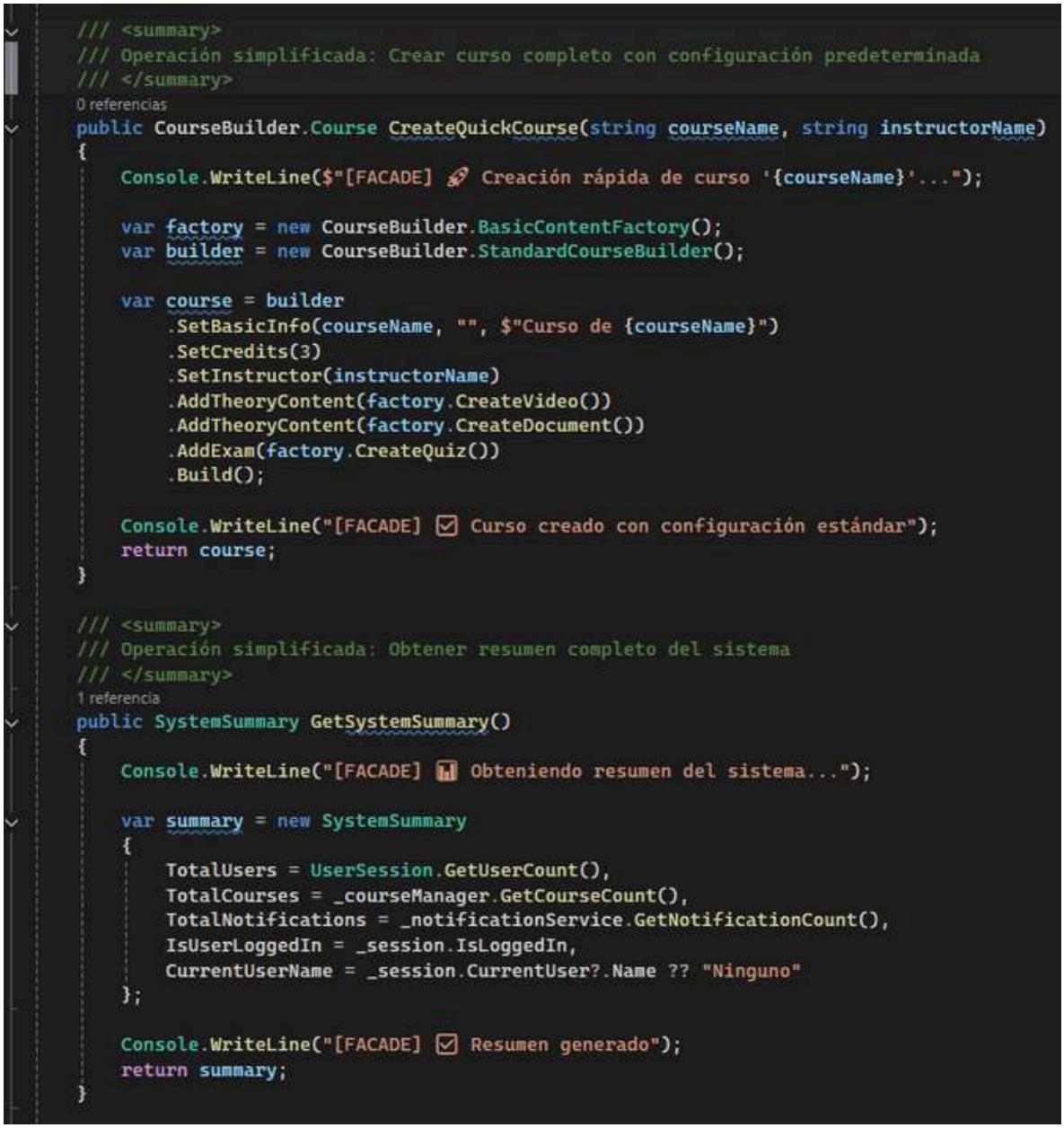


Estudiante: Accede a sus cursos, materiales, actividades y trámites.

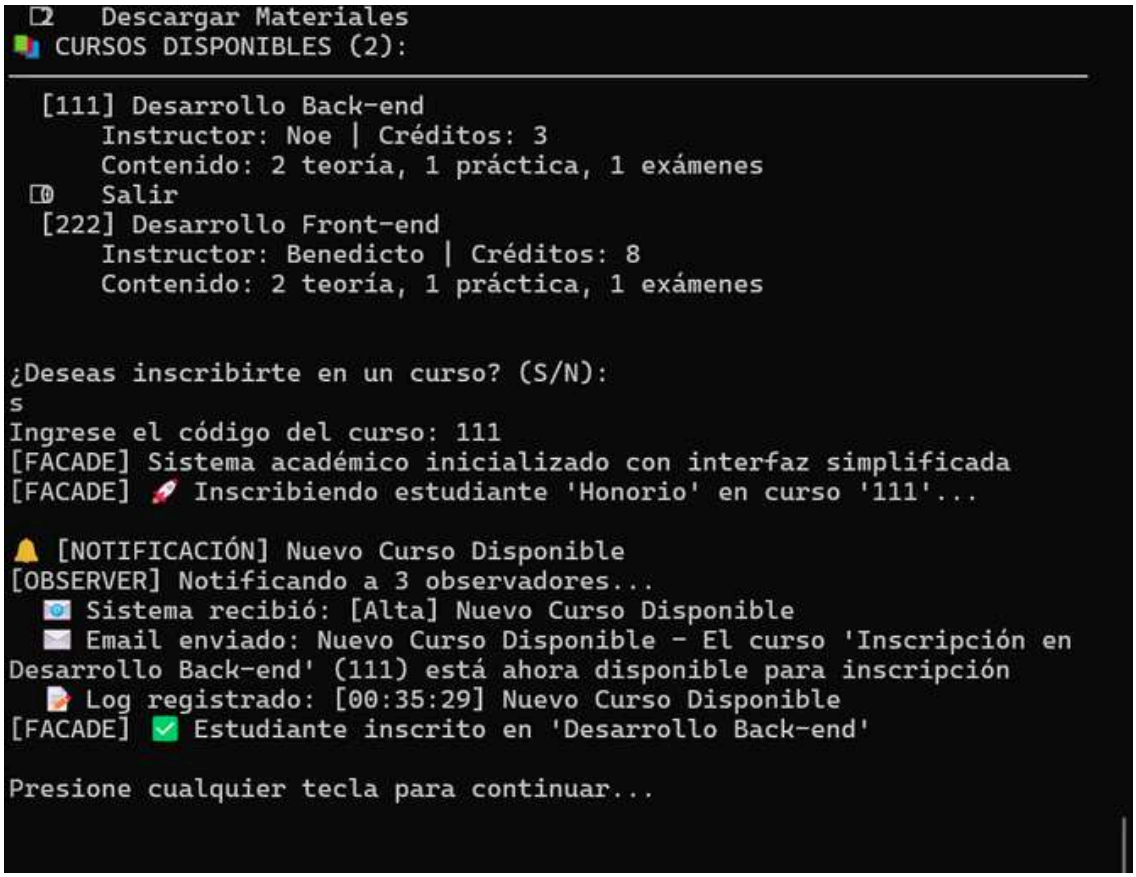
Facade: **AcademicSystemFacade** simplifica el proceso de inscripción de estudiantes, ocultando la complejidad de verificar requisitos, cupos y pagos.




```
namespace PlataformaAcademicaModular.UserManagement;

/// <summary>
/// PATRÓN FACADE: Proporciona una interfaz simplificada para un subsistema complejo
/// Oculta la complejidad de múltiples componentes detrás de una API simple
/// </summary>
4 referencias
public class AcademicSystemFacade
{
    private readonly UserSession _session;
    private readonly CourseBuilder.CourseManager _courseManager;
    private readonly NotificationCenter.NotificationService _notificationService;

    3 referencias
    public AcademicSystemFacade()
    {
        _session = UserSession.Instance;
        _courseManager = CourseBuilder.CourseManager.Instance;
        _notificationService = NotificationCenter.NotificationService.Instance;
        Console.WriteLine("[FACADE] Sistema académico inicializado con interfaz simplificada");
    }

    /// <summary>
    /// Operación simplificada: Registrar e iniciar sesión en un solo paso
    /// </summary>
    0 referencias
    public bool QuickRegisterAndLogin(string username, string password, string role, string additionalInfo)
    {
        Console.WriteLine("[FACADE] 🚀 Registro e inicio de sesión rápido...");

        // Paso 1: Registrar
        bool registered = _session.Register(username, password, role, additionalInfo);
        if (!registered)
        {
            return false;
        }

        // Paso 2: Iniciar sesión automáticamente
        bool loggedIn = _session.Login(username, password);

        if (loggedIn)
        {
            Console.WriteLine("[FACADE] ☑ Usuario registrado e iniciado sesión exitosamente");
        }

        return loggedIn;
    }
```

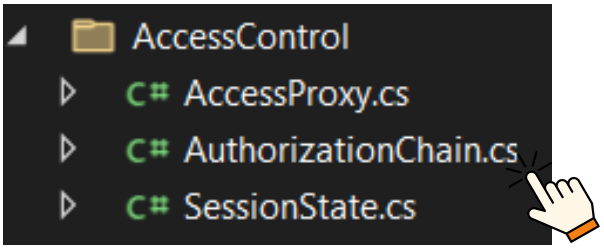


```
/// <summary>
/// Operación simplificada: Crear curso completo con configuración predeterminada
/// </summary>
0 referencias
public CourseBuilder.Course CreateQuickCourse(string courseName, string instructorName)
{
    Console.WriteLine($"[FACADE] 🚀 Creación rápida de curso '{courseName}'...");

    var factory = new CourseBuilder.BasicContentFactory();
    var builder = new CourseBuilder.StandardCourseBuilder();

    var course = builder
        .SetBasicInfo(courseName, "", $"Curso de {courseName}")
        .SetCredits(3)
        .SetInstructor(instructorName)
        .AddTheoryContent(factory.CreateVideo())
        .AddTheoryContent(factory.CreateDocument())
        .AddExam(factory.CreateQuiz())
        .Build();

    Console.WriteLine("[FACADE] ☑ Curso creado con configuración estándar");
    return course;
}

/// <summary>
/// Operación simplificada: Obtener resumen completo del sistema
/// </summary>
1 referencia
public SystemSummary GetSystemSummary()
{
    Console.WriteLine("[FACADE] 📊 Obteniendo resumen del sistema...");

    var summary = new SystemSummary
    {
        TotalUsers = UserSession.GetUserCount(),
        TotalCourses = _courseManager.GetCourseCount(),
        TotalNotifications = _notificationService.GetNotificationCount(),
        IsUserLoggedIn = _session.IsLoggedIn,
        CurrentUserName = _session.CurrentUser?.Name ?? "Ninguno"
    };

    Console.WriteLine("[FACADE] ☑ Resumen generado");
    return summary;
}
```



El usuario ve una acción única, pero el sistema ejecuta múltiples operaciones internas.

Chain of Responsibility: **AuthorizationChain** procesa solicitudes de acceso a través de una cadena de manejadores (Admin, Profesor, Estudiante) hasta que uno la aprueba.

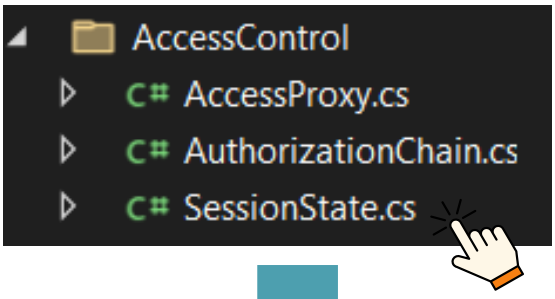



```
namespace PlataformaAcademicaModular.AccessControl;

/// <summary>
/// Solicitud de autorización
/// </summary>
6 referencias
public class AuthorizationRequest
{
    5 referencias
    public string UserRole { get; set; } = string.Empty;
    2 referencias
    public string Resource { get; set; } = string.Empty;
    6 referencias
    public string Action { get; set; } = string.Empty;
    5 referencias
    public bool IsAuthorized { get; set; }
}

/// <summary>
/// PATRÓN CHAIN OF RESPONSIBILITY: Procesa solicitudes a través de una cadena de manejadores
/// Cada manejador decide si procesa la solicitud o la pasa al siguiente
/// </summary>
7 referencias
public abstract class AuthorizationHandler
{
    protected AuthorizationHandler? _nextHandler;

    3 referencias
    public void SetNext(AuthorizationHandler handler)
    {
        _nextHandler = handler;
    }

    8 referencias
    public abstract void Handle(AuthorizationRequest request);
}

/// <summary>
/// Manejador: Verificación de administrador
/// </summary>
1 referencia
public class AdminAuthorizationHandler : AuthorizationHandler
{
    5 referencias
    public override void Handle(AuthorizationRequest request)
    {
        if (request.UserRole == "Administrador")
        {
            request.IsAuthorized = true;
            Console.WriteLine($"[CHAIN] ☑ AdminHandler: Acceso total concedido para {request.UserRole}");
            return;
        }

        Console.WriteLine($"[CHAIN] AdminHandler: No es administrador, pasando al siguiente...");
        _nextHandler?.Handle(request);
    }
}
```

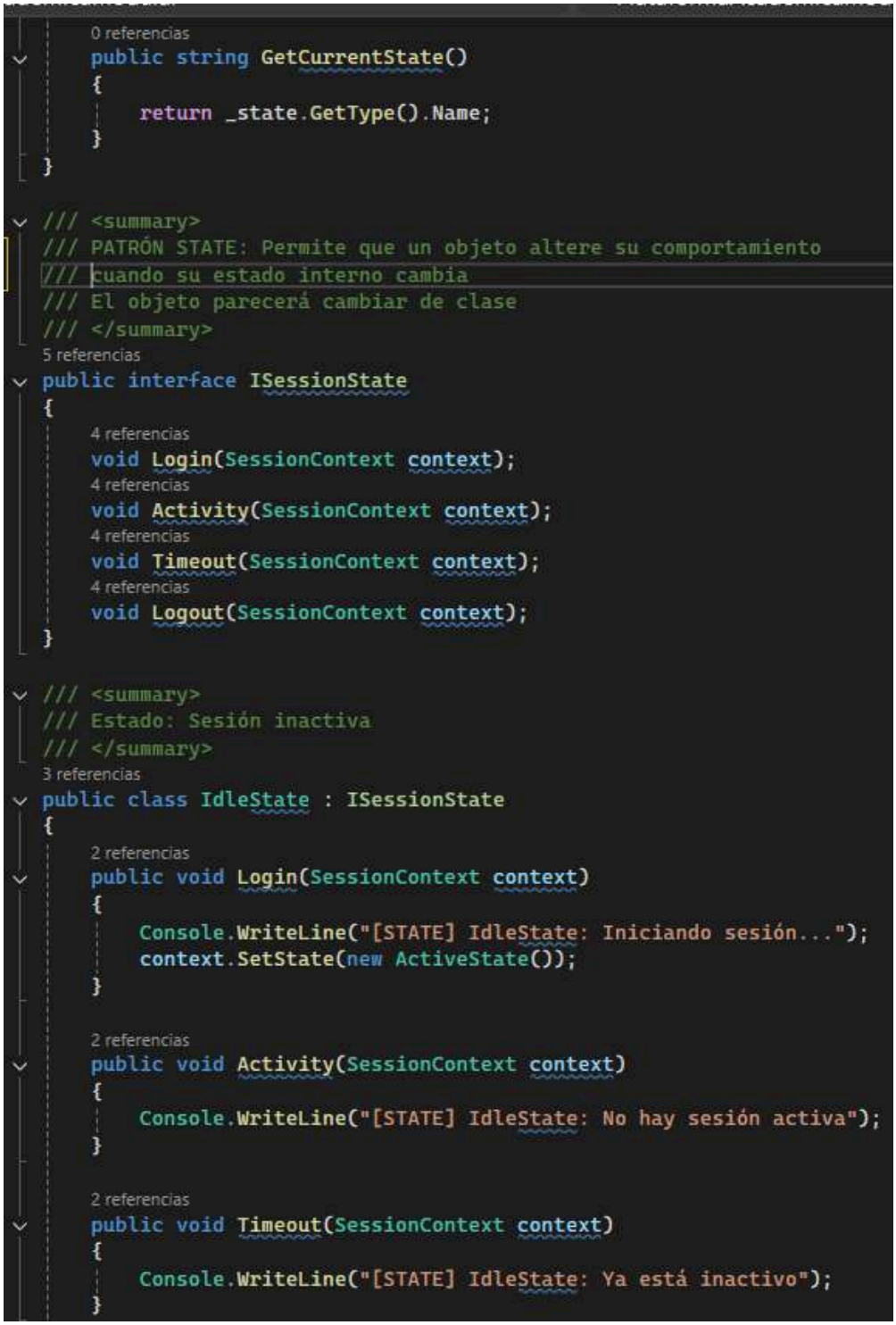
```
/// <summary>
/// Manejador: Verificación de estudiante
/// </summary>
1 referencia
public class StudentAuthorizationHandler : AuthorizationHandler
{
    5 referencias
    public override void Handle(AuthorizationRequest request)
    {
        if (request.UserRole == "Estudiante")
        {
            if (request.Action == "Read" && request.Resource == "Courses")
            {
                request.IsAuthorized = true;
                Console.WriteLine($"[CHAIN] ☑ StudentHandler: Acceso de lectura concedido");
                return;
            }
        }

        Console.WriteLine($"[CHAIN] StudentHandler: No autorizado, pasando al siguiente...");
        _nextHandler?.Handle(request);
    }
}

/// <summary>
/// Manejador final: Denegar por defecto
/// </summary>
1 referencia
public class DefaultDenyHandler : AuthorizationHandler
{
    5 referencias
    public override void Handle(AuthorizationRequest request)
    {
        request.IsAuthorized = false;
        Console.WriteLine($"[CHAIN] ❌ DefaultDenyHandler: Acceso denegado por defecto");
    }
}

/// <summary>
/// Configurador de la cadena de autorización
/// </summary>
1 referencia
public class AuthorizationChain
{
    private readonly AuthorizationHandler _chain;

    0 referencias
    public AuthorizationChain()
    {
        // Construir la cadena
        var adminHandler = new AdminAuthorizationHandler();
        var teacherHandler = new TeacherAuthorizationHandler();
        var studentHandler = new StudentAuthorizationHandler();
        var denyHandler = new DefaultDenyHandler();

        adminHandler.SetNext(teacherHandler);
        teacherHandler.SetNext(studentHandler);
        studentHandler.SetNext(denyHandler);

        _chain = adminHandler;
        Console.WriteLine("[CHAIN] Cadena de autorización configurada");
    }

    0 referencias
    public bool Authorize(string userRole, string resource, string action)
    {
        var request = new AuthorizationRequest
        {
            UserRole = userRole,
            Resource = resource,
            Action = action
        };

        Console.WriteLine($"[CHAIN] Procesando solicitud: {userRole} -> {action} en {resource}");
        _chain.Handle(request);
        return request.IsAuthorized;
    }
}
```

Define claramente los campos relevantes: UserRole, Resource, Action, IsAuthorized esto facilita la trazabilidad y la documentación de cada intento de acceso.

State: **SessionState** gestiona los estados de la sesión del usuario (Activa, Inactiva, Expirada) cambiando el comportamiento del objeto sesión dinámicamente.

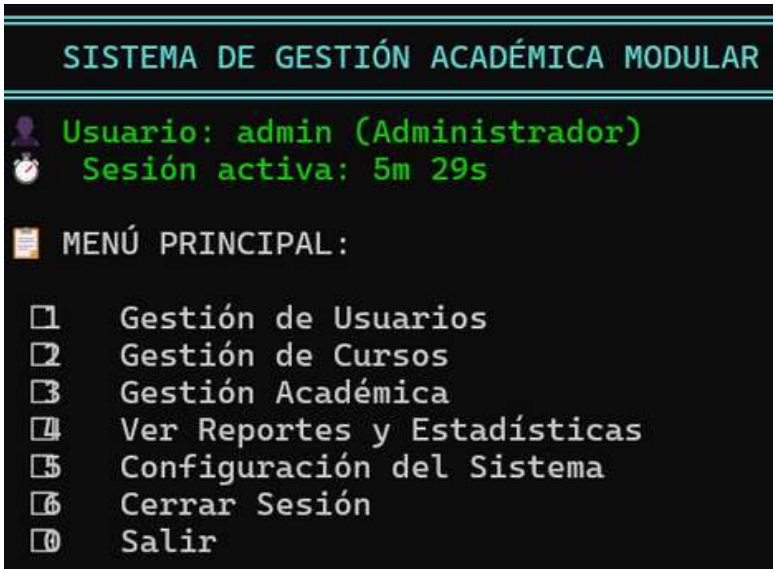



```
namespace PlataformaAcademicaModular.AccessControl;

/// <summary>
/// Contexto de sesión
/// </summary>
17 referencias
public class SessionContext
{
    private ISessionState _state;

    0 referencias
    public SessionContext()
    {
        // Estado inicial
        _state = new IdleState();
        Console.WriteLine("[STATE] Sesión creada en estado Idle");
    }

    5 referencias
    public void SetState(ISessionState state)
    {
        _state = state;
        Console.WriteLine($"[STATE] Estado cambiado a: {state.GetType().Name}");
    }

    0 referencias
    public void Login()
    {
        _state.Login(this);
    }

    0 referencias
    public void Activity()
    {
        _state.Activity(this);
    }

    0 referencias
    public void Timeout()
    {
        _state.Timeout(this);
    }

    0 referencias
    public void Logout()
    {
        _state.Logout(this);
    }
```

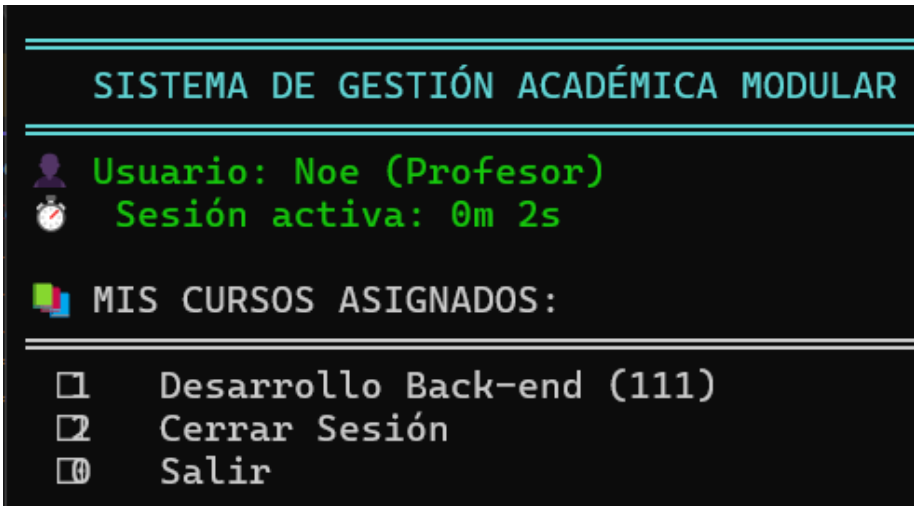


```
        0 referencias
        public string GetCurrentState()
        {
            return _state.GetType().Name;
        }
    }

    /// <summary>
    /// PATRÓN STATE: Permite que un objeto altere su comportamiento
    /// cuando su estado interno cambia
    /// El objeto parecerá cambiar de clase
    /// </summary>
    5 referencias
    public interface ISessionState
    {
        4 referencias
        void Login(SessionContext context);
        4 referencias
        void Activity(SessionContext context);
        4 referencias
        void Timeout(SessionContext context);
        4 referencias
        void Logout(SessionContext context);
    }

    /// <summary>
    /// Estado: Sesión inactiva
    /// </summary>
    3 referencias
    public class IdleState : ISessionState
    {
        2 referencias
        public void Login(SessionContext context)
        {
            Console.WriteLine("[STATE] IdleState: Iniciando sesión...");
            context.SetState(new ActiveState());
        }

        2 referencias
        public void Activity(SessionContext context)
        {
            Console.WriteLine("[STATE] IdleState: No hay sesión activa");
        }

        2 referencias
        public void Timeout(SessionContext context)
        {
            Console.WriteLine("[STATE] IdleState: Ya está inactivo");
        }
```

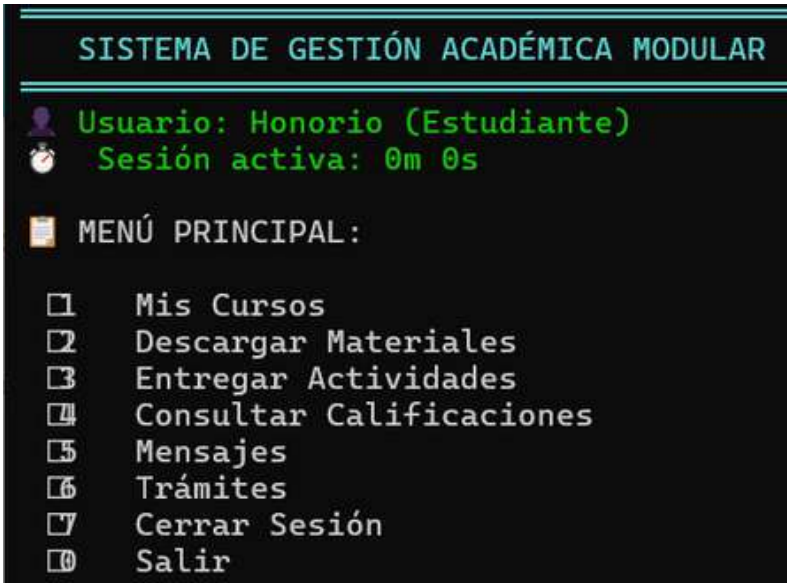
```
        2 referencias
        public void Logout(SessionContext context)
        {
            Console.WriteLine("[STATE] IdleState: No hay sesión para cerrar");
        }
    }

    /// <summary>
    /// Estado: Sesión activa
    /// </summary>
    2 referencias
    public class ActiveState : ISessionState
    {
        private DateTime _lastActivity = DateTime.Now;

        2 referencias
        public void Login(SessionContext context)
        {
            Console.WriteLine("[STATE] ActiveState: Ya hay una sesión activa");
        }

        2 referencias
        public void Activity(SessionContext context)
        {
            _lastActivity = DateTime.Now;
            Console.WriteLine($"[STATE] ActiveState: Actividad registrada a las {_lastActivity:

        }

        2 referencias
        public void Timeout(SessionContext context)
        {
            Console.WriteLine("[STATE] ActiveState: Sesión expirada por inactividad");
            context.SetState(new ExpiredState());
        }

        2 referencias
        public void Logout(SessionContext context)
        {
            Console.WriteLine("[STATE] ActiveState: Cerrando sesión...");
            context.SetState(new IdleState());
        }
    }

    /// <summary>
    /// Estado: Sesión expirada
    /// </summary>
    1 referencia
    public class ExpiredState : ISessionState
    {
        2 referencias
        public void Login(SessionContext context)
        {
            Console.WriteLine("[STATE] ExpiredState: Reiniciando sesión...");
            context.SetState(new ActiveState());
        }

        2 referencias
        public void Activity(SessionContext context)
        {
            Console.WriteLine("[STATE] ExpiredState: Sesión expirada, debe iniciar sesión nuevamente");
        }

        2 referencias
        public void Timeout(SessionContext context)
        {
            Console.WriteLine("[STATE] ExpiredState: La sesión ya está expirada");
        }

        2 referencias
        public void Logout(SessionContext context)
        {
            Console.WriteLine("[STATE] ExpiredState: Limpiando sesión expirada...");
            context.SetState(new IdleState());
        }
    }
```



Administrador: Sesión activa por 5m 29s



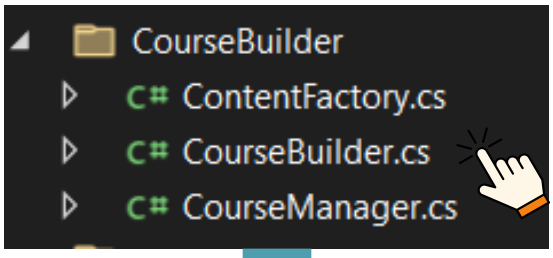
Profesor: Sesión activa por 0m 2s



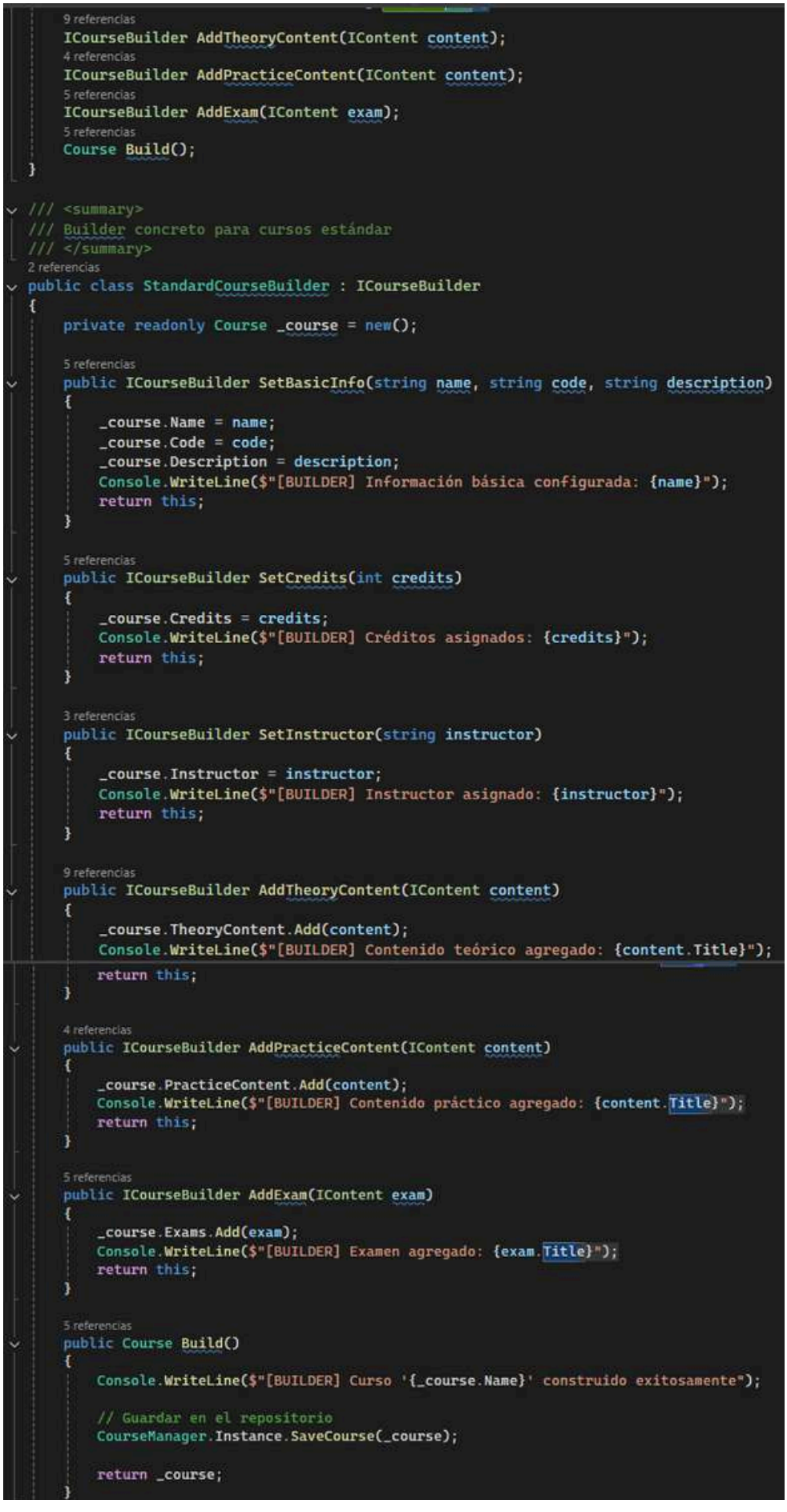
Estudiante: Sesión activa por 0m 0s

2. Gestión de Cursos y Contenidos

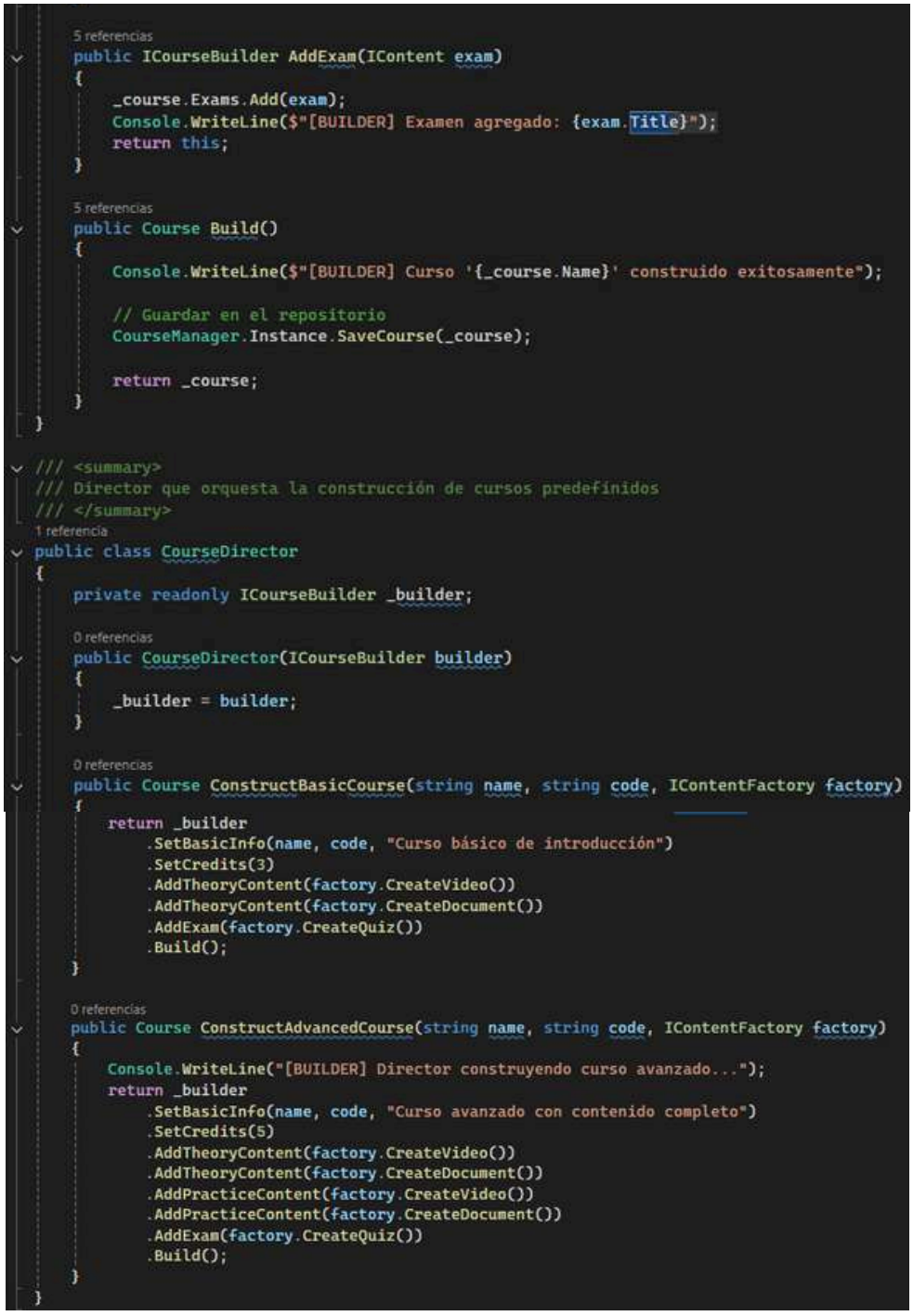
Builder: **CourseBuilder** permite construir objetos **Course** complejos paso a paso (info básica, contenido teórico, práctico, exámenes).




```
namespace PlataformaAcademicaModular.CourseBuilder;

/// <summary>
/// Clase que representa un curso completo
/// </summary>
20 referencias
public class Course
{
    33 referencias
    public string Name { get; set; } = string.Empty;
    24 referencias
    public string Code { get; set; } = string.Empty;
    4 referencias
    public string Description { get; set; } = string.Empty;
    5 referencias
    public List<IContent> TheoryContent { get; set; } = new();
    5 referencias
    public List<IContent> PracticeContent { get; set; } = new();
    3 referencias
    public List<IContent> Exams { get; set; } = new();
    6 referencias
    public int Credits { get; set; }
    10 referencias
    public string Instructor { get; set; } = string.Empty;

    2 referencias
    public void DisplayCourse()
    {
        Console.WriteLine($"\n Curso: {Name} ({Code})");
        Console.WriteLine($"   Descripción: {Description}");
        Console.WriteLine($"   Créditos: {Credits} | Instructor: {Instructor}");
        Console.WriteLine($"   Contenido Teórico: {TheoryContent.Count} elementos");
        Console.WriteLine($"   Contenido Práctico: {PracticeContent.Count} elementos");
        Console.WriteLine($"   Exámenes: {Exams.Count} elementos");
    }
}

/// <summary>
/// PATRÓN BUILDER: Construcción paso a paso de cursos complejos
/// Permite crear cursos con diferentes configuraciones de manera fluida
/// </summary>
15 referencias
public interface ICourseBuilder
{
    5 referencias
    ICourseBuilder SetBasicInfo(string name, string code, string description);
    5 referencias
    ICourseBuilder SetCredits(int credits);
    3 referencias
    ICourseBuilder SetInstructor(string instructor);
```



```
    9 referencias
    ICourseBuilder AddTheoryContent(IContent content);
    4 referencias
    ICourseBuilder AddPracticeContent(IContent content);
    5 referencias
    ICourseBuilder AddExam(IContent exam);
    5 referencias
    Course Build();
}

/// <summary>
/// Builder concreto para cursos estándar
/// </summary>
2 referencias
public class StandardCourseBuilder : ICourseBuilder
{
    private readonly Course _course = new();

    5 referencias
    public ICourseBuilder SetBasicInfo(string name, string code, string description)
    {
        _course.Name = name;
        _course.Code = code;
        _course.Description = description;
        Console.WriteLine($"[BUILDER] Información básica configurada: {name}");
        return this;
    }

    5 referencias
    public ICourseBuilder SetCredits(int credits)
    {
        _course.Credits = credits;
        Console.WriteLine($"[BUILDER] Créditos asignados: {credits}");
        return this;
    }

    3 referencias
    public ICourseBuilder SetInstructor(string instructor)
    {
        _course.Instructor = instructor;
        Console.WriteLine($"[BUILDER] Instructor asignado: {instructor}");
        return this;
    }

    9 referencias
    public ICourseBuilder AddTheoryContent(IContent content)
    {
        _course.TheoryContent.Add(content);
        Console.WriteLine($"[BUILDER] Contenido teórico agregado: {content.Title}");
        return this;
    }

    4 referencias
    public ICourseBuilder AddPracticeContent(IContent content)
    {
        _course.PracticeContent.Add(content);
        Console.WriteLine($"[BUILDER] Contenido práctico agregado: {content.Title}");
        return this;
    }

    5 referencias
    public ICourseBuilder AddExam(IContent exam)
    {
        _course.Exams.Add(exam);
        Console.WriteLine($"[BUILDER] Examen agregado: {exam.Title}");
        return this;
    }

    5 referencias
    public Course Build()
    {
        Console.WriteLine($"[BUILDER] Curso '{_course.Name}' construido exitosamente");

        // Guardar en el repositorio
        CourseManager.Instance.SaveCourse(_course);

        return _course;
    }
```

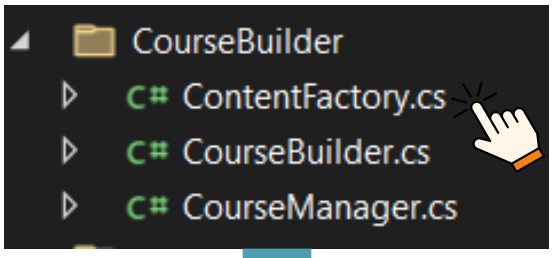


```
    5 referencias
    public ICourseBuilder AddExam(IContent exam)
    {
        _course.Exams.Add(exam);
        Console.WriteLine($"[BUILDER] Examen agregado: {exam.Title}");
        return this;
    }

    5 referencias
    public Course Build()
    {
        Console.WriteLine($"[BUILDER] Curso '{_course.Name}' construido exitosamente");

        // Guardar en el repositorio
        CourseManager.Instance.SaveCourse(_course);

        return _course;
    }
}

/// <summary>
/// Director que orquesta la construcción de cursos predefinidos
/// </summary>
1 referencia
public class CourseDirector
{
    private readonly ICourseBuilder _builder;

    0 referencias
    public CourseDirector(ICourseBuilder builder)
    {
        _builder = builder;
    }

    0 referencias
    public Course ConstructBasicCourse(string name, string code, IContentFactory factory)
    {
        return _builder
            .SetBasicInfo(name, code, "Curso básico de introducción")
            .SetCredits(3)
            .AddTheoryContent(factory.CreateVideo())
            .AddTheoryContent(factory.CreateDocument())
            .AddExam(factory.CreateQuiz())
            .Build();
    }

    0 referencias
    public Course ConstructAdvancedCourse(string name, string code, IContentFactory factory)
    {
        Console.WriteLine("[BUILDER] Director construyendo curso avanzado...");
        return _builder
            .SetBasicInfo(name, code, "Curso avanzado con contenido completo")
            .SetCredits(5)
            .AddTheoryContent(factory.CreateVideo())
            .AddTheoryContent(factory.CreateDocument())
            .AddPracticeContent(factory.CreateVideo())
            .AddPracticeContent(factory.CreateDocument())
            .AddExam(factory.CreateQuiz())
            .Build();
    }
}
```

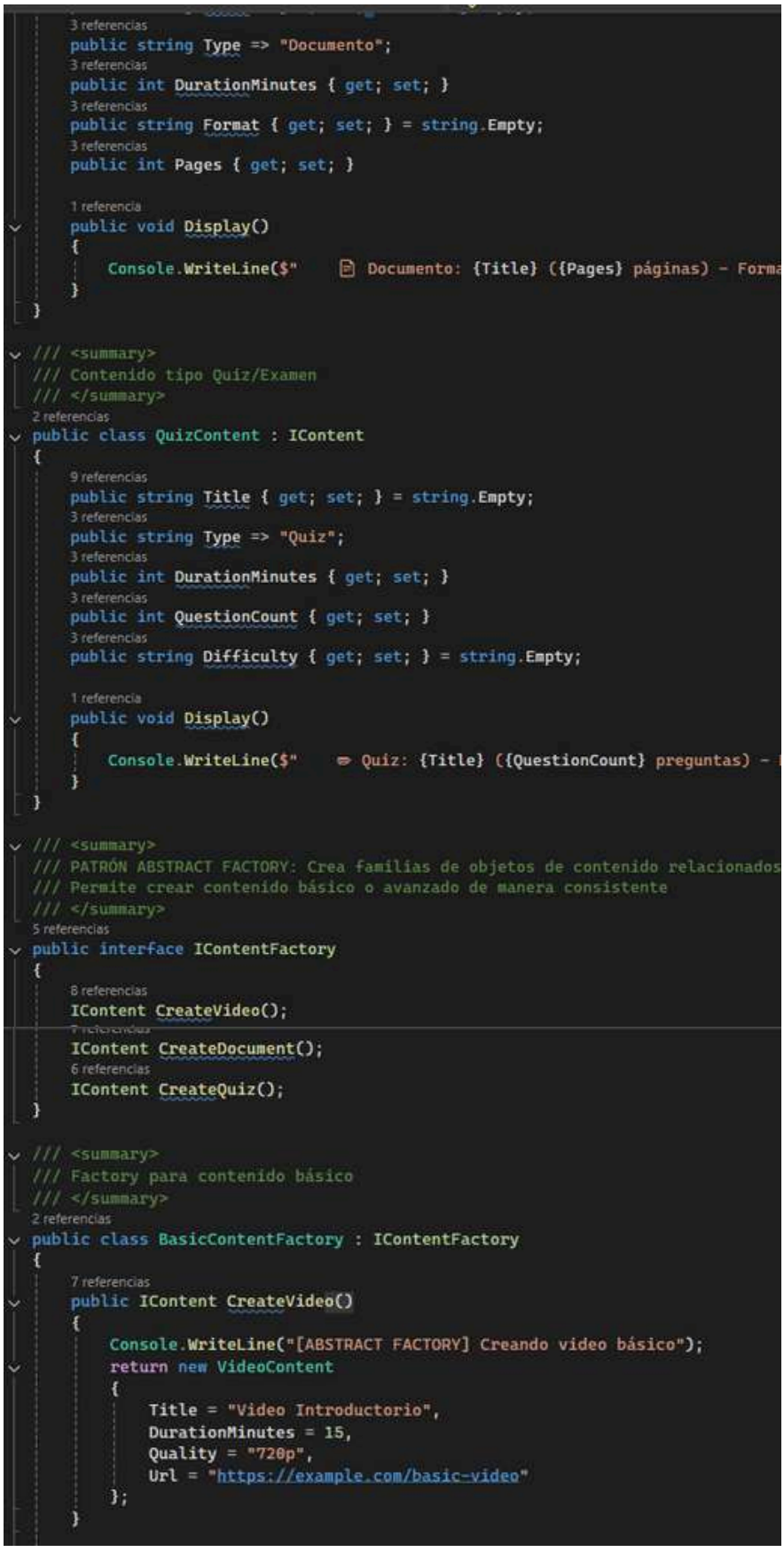
Curso avanzado creado paso a paso con Builder.

Se configuran créditos, instructor y contenidos, y se notifica al sistema al finalizar

Abstract Factory: **ContentFactory** crea familias de objetos de contenido (Videos, Documentos) asegurando consistencia.




```
namespace PlataformaAcademicaModular.CourseBuilder;

/// <summary>
/// Interfaz base para contenido educativo
/// </summary>
21 referencias
public interface IContent
{
    17 referencias
    string Title { get; set; }
    5 referencias
    string Type { get; }
    10 referencias
    int DurationMinutes { get; set; }
    3 referencias
    void Display();
}

/// <summary>
/// Contenido tipo Video
/// </summary>
2 referencias
public class VideoContent : IContent
{
    9 referencias
    public string Title { get; set; } = string.Empty;
    3 referencias
    public string Type => "Video";
    4 referencias
    public int DurationMinutes { get; set; }
    3 referencias
    public string Quality { get; set; } = string.Empty;
    2 referencias
    public string Url { get; set; } = string.Empty;

    1 referencia
    public void Display()
    {
        Console.WriteLine($"    Video: {Title} ({DurationMinutes} min) - Calidad: {Quality}");
    }
}

/// <summary>
/// Contenido tipo Documento
/// </summary>
2 referencias
public class DocumentContent : IContent
{
```

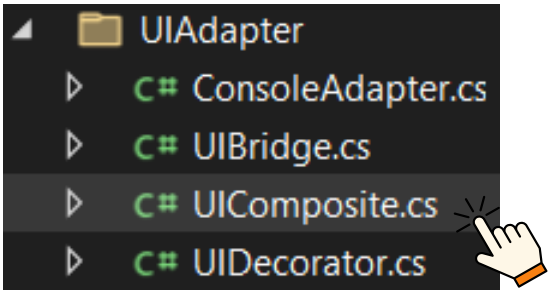


```
    3 referencias
    public string Type => "Documento";
    3 referencias
    public int DurationMinutes { get; set; }
    3 referencias
    public string Format { get; set; } = string.Empty;
    3 referencias
    public int Pages { get; set; }

    1 referencia
    public void Display()
    {
        Console.WriteLine($"    Documento: {Title} ({Pages} páginas) - Forma
    }
}

/// <summary>
/// Contenido tipo Quiz/Examen
/// </summary>
2 referencias
public class QuizContent : IContent
{
    9 referencias
    public string Title { get; set; } = string.Empty;
    3 referencias
    public string Type => "Quiz";
    3 referencias
    public int DurationMinutes { get; set; }
    3 referencias
    public int QuestionCount { get; set; }
    3 referencias
    public string Difficulty { get; set; } = string.Empty;

    1 referencia
    public void Display()
    {
        Console.WriteLine($"    Quiz: {Title} ({QuestionCount} preguntas) -
    }
}

/// <summary>
/// PATRÓN ABSTRACT FACTORY: Crea familias de objetos de contenido relacionados
/// Permite crear contenido básico o avanzado de manera consistente
/// </summary>
5 referencias
public interface IContentFactory
{
    8 referencias
    IContent CreateVideo();
    IContent CreateDocument();
    6 referencias
    IContent CreateQuiz();
}

/// <summary>
/// Factory para contenido básico
/// </summary>
2 referencias
public class BasicContentFactory : IContentFactory
{
    7 referencias
    public IContent CreateVideo()
    {
        Console.WriteLine("[ABSTRACT FACTORY] Creando video básico");
        return new VideoContent
        {
            Title = "Video Introductorio",
            DurationMinutes = 15,
            Quality = "720p",
            Url = "https://example.com/basic-video"
        };
    }
```

```
    6 referencias
    public IContent CreateDocument()
    {
        Console.WriteLine("[ABSTRACT FACTORY] Creando documento básico");
        return new DocumentContent
        {
            Title = "Guía de Introducción",
            DurationMinutes = 20,
            Format = "PDF",
            Pages = 10
        };
    }

    5 referencias
    public IContent CreateQuiz()
    {
        Console.WriteLine("[ABSTRACT FACTORY] Creando quiz básico");
        return new QuizContent
        {
            Title = "Evaluación Básica",
            DurationMinutes = 30,
            QuestionCount = 10,
            Difficulty = "Fácil"
        };
    }
}

/// <summary>
/// Factory para contenido avanzado
/// </summary>
1 referencia
public class AdvancedContentFactory : IContentFactory
{
    6 referencias
    public IContent CreateVideo()
    {
        Console.WriteLine("[ABSTRACT FACTORY] Creando video avanzado");
        return new VideoContent
        {
            Title = "Masterclass Avanzada",
            DurationMinutes = 45,
            Quality = "4K",
            Url = "https://example.com/advanced-video"
        };
    }

    5 referencias
    public IContent CreateDocument()
    {
        Console.WriteLine("[ABSTRACT FACTORY] Creando documento avanzado");
        return new DocumentContent
        {
            Title = "Manual Técnico Completo",
            DurationMinutes = 60,
            Format = "PDF Interactivo",
            Pages = 50
        };
    }

    4 referencias
    public IContent CreateQuiz()
    {
        Console.WriteLine("[ABSTRACT FACTORY] Creando quiz avanzado");
        return new QuizContent
        {
            Title = "Examen Certificación",
            DurationMinutes = 90,
            QuestionCount = 50,
            Difficulty = "Difícil"
        };
    }
}
```

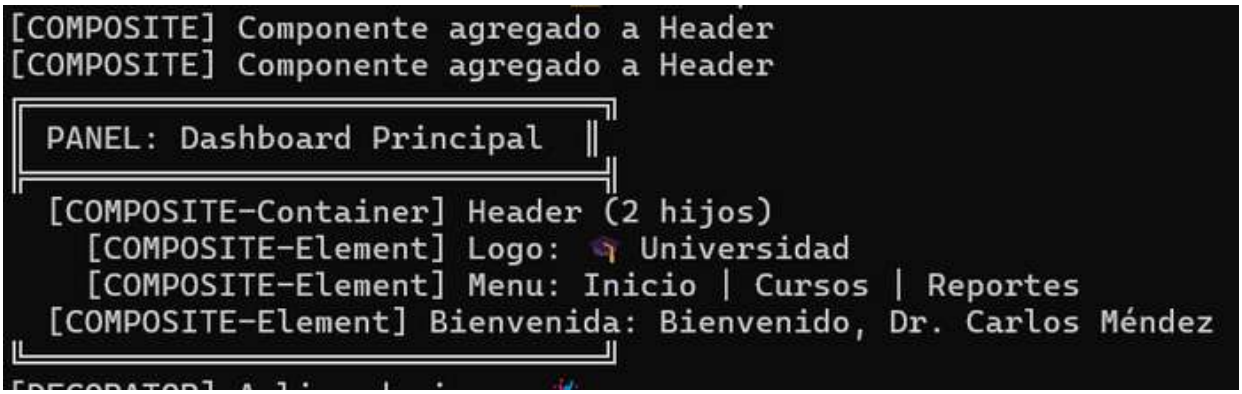
Curso avanzado construido con Builder y Abstract Factory; notificado al sistema mediante Observer.

Composite: **UIContainer** maneja la jerarquía de elementos del curso como una estructura de árbol (secciones, temas, subtemas).



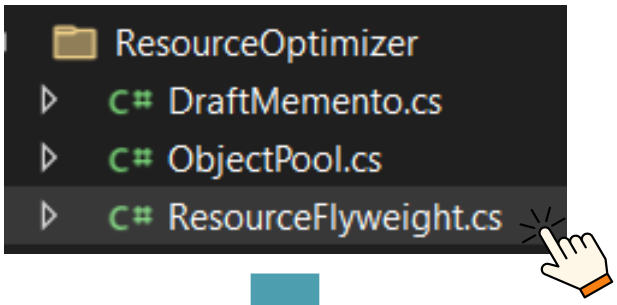

```
namespace PlataformaAcademicaModular.UIAdapter;

/// <summary>
/// PATRÓN COMPOSITE: Compone objetos en estructuras de árbol
/// Permite tratar objetos individuales y composiciones de manera uniforme
/// </summary>
12 referencias
public abstract class UIComponent
{
    protected string _name;

    2 referencias
    protected UIComponent(string name)
    {
        _name = name;
    }

    6 referencias
    public abstract void Render(int depth = 0);

    4 referencias
    public virtual void Add(UIComponent component)
    {
        throw new NotSupportedException();
    }

    1 referencia
    public virtual void Remove(UIComponent component)
    {
        throw new NotSupportedException();
    }
}

/// <summary>
/// Hoja: Elemento simple de UI
/// </summary>
3 referencias
public class UIElement : UIComponent
{
    private readonly string _content;

    2 referencias
    public UIElement(string name, string content) : base(name)
    {
        _content = content;
    }

    3 referencias
    public override void Render(int depth = 0)
```

```
    {
        var indent = new string(' ', depth * 2);
        Console.WriteLine($"{indent}[COMPOSITE-Element] {_name}: {_content}");
    }
}

/// <summary>
/// Compuesto: Contenedor de elementos UI
/// </summary>
4 referencias
public class UIContainer : UIComponent
{
    private readonly List<UIComponent> _children = new();

    2 referencias
    public UIContainer(string name) : base(name) { }

    4 referencias
    public override void Add(UIComponent component)
    {
        _children.Add(component);
        Console.WriteLine($"[COMPOSITE] Componente agregado a {_name}");
    }

    1 referencia
    public override void Remove(UIComponent component)
    {
        _children.Remove(component);
        Console.WriteLine($"[COMPOSITE] Componente removido de {_name}");
    }

    5 referencias
    public override void Render(int depth = 0)
    {
        var indent = new string(' ', depth * 2);
        Console.WriteLine($"{indent}[COMPOSITE-Container] {_name} ({_children.Count} hi

        foreach (var child in _children)
        {
            child.Render(depth + 1);
        }
    }
}

/// <summary>
/// Ejemplo de uso: Panel complejo
/// </summary>
1 referencia
public class UIPanel : UIContainer
{
    0 referencias
    public UIPanel(string name) : base(name) { }

    5 referencias
    public override void Render(int depth = 0)
    {
        var indent = new string(' ', depth * 2);
        Console.WriteLine($"{indent}╔══════════════════════════════╗");
        Console.WriteLine($"{indent}║ PANEL: {_name.PadRight(20)} ");
        Console.WriteLine($"{indent}╚══════════════════════════════╝");

        foreach (var child in _children)
        {
            child.Render(depth + 1);
        }

        Console.WriteLine($"{indent}└──────────────────────────────┘");
    }

    private readonly List<UIComponent> _children = new();

    4 referencias
    public override void Add(UIComponent component)
    {
        _children.Add(component);
    }
}
```



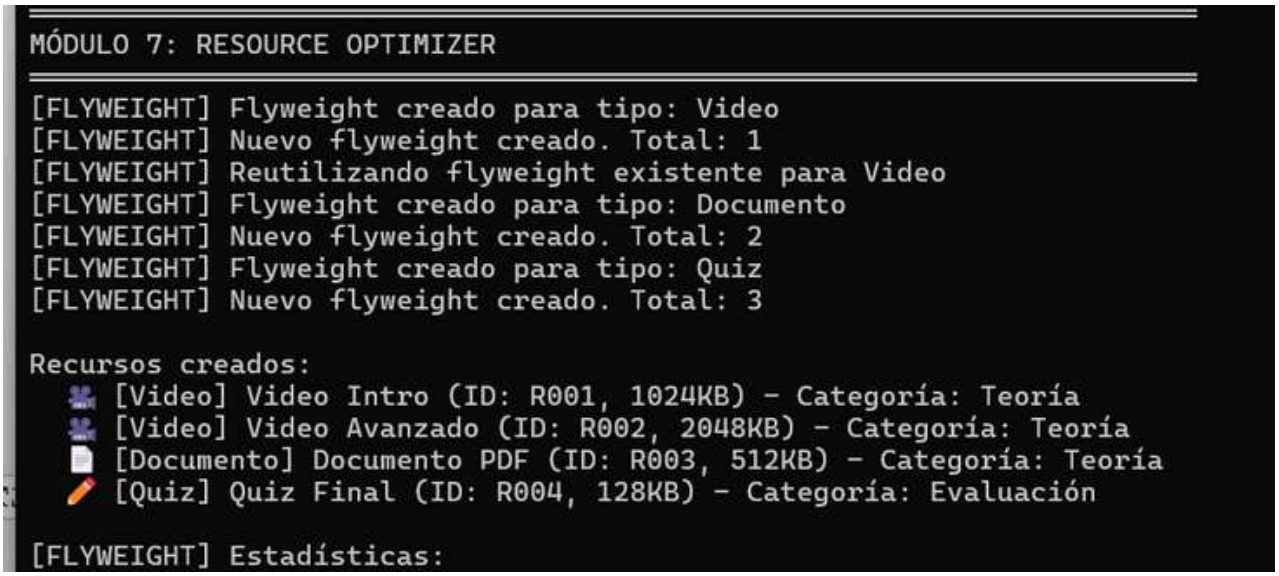
Renderiza su nombre y luego recorre a sus hijos:

Flyweight: **ResourceFlyweight** optimiza la memoria compartiendo recursos comunes (como iconos o metadatos de archivos) entre múltiples cursos.



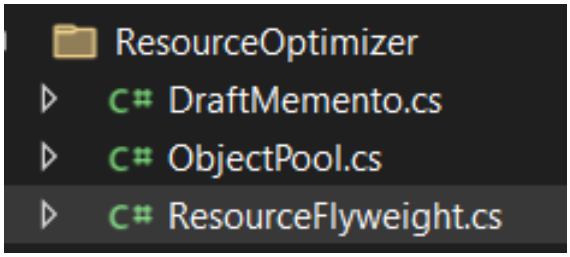

```
namespace PlataformaAcademicaModular.ResourceOptimizer;

/// <summary>
/// Datos compartidos (intrínsecos) de recursos
/// </summary>
4 referencias
public class SharedResourceData
{
    3 referencias
    public string Type { get; set; } = string.Empty;
    2 referencias
    public string Icon { get; set; } = string.Empty;
    2 referencias
    public string Category { get; set; } = string.Empty;

    1 referencia
    public SharedResourceData(string type, string icon, string category)
    {
        Type = type;
        Icon = icon;
        Category = category;
    }
}

/// <summary>
/// PATRÓN FLYWEIGHT: Comparte eficientemente objetos que se usan en grandes cantidades
/// Minimiza el uso de memoria compartiendo datos comunes
/// </summary>
6 referencias
public class ResourceFlyweight
{
    private readonly SharedResourceData _sharedData;

    1 referencia
    public ResourceFlyweight(SharedResourceData sharedData)
    {
        _sharedData = sharedData;
        Console.WriteLine($"[FLYWEIGHT] Flyweight creado para tipo: {sharedData.Type}");
    }

    1 referencia
    public void Display(string uniqueId, string name, int size)
    {
        Console.WriteLine($"  {_sharedData.Icon} [{_sharedData.Type}] {name} (ID: {uniqueId}, {size}KB) - Categoría: {_sharedData.Category}");
    }
}
```

```
/// <summary>
/// Fábrica de Flyweights
/// </summary>
0 referencias
public class ResourceFlyweightFactory
{
    private readonly Dictionary<string, ResourceFlyweight> _flyweights = new();
    private int _createdCount = 0;

    0 referencias
    public ResourceFlyweight GetFlyweight(string type, string icon, string category)
    {
        string key = $"{type}_{category}";

        if (!_flyweights.ContainsKey(key))
        {
            var sharedData = new SharedResourceData(type, icon, category);
            _flyweights[key] = new ResourceFlyweight(sharedData);
            _createdCount++;
            Console.WriteLine($"[FLYWEIGHT] Nuevo flyweight creado. Total: {_createdCount}");
        }
        else
        {
            Console.WriteLine($"[FLYWEIGHT] Reutilizando flyweight existente para {type}");
        }

        return _flyweights[key];
    }

    0 referencias
    public void ShowStatistics()
    {
        Console.WriteLine($"\n[FLYWEIGHT] Estadísticas:");
        Console.WriteLine($"  Total de flyweights únicos: {_flyweights.Count}");
        Console.WriteLine($"  Memoria ahorrada: ~{(_createdCount - _flyweights.Count) * 100}KB (estimado)");
    }
}

/// <summary>
/// Recurso con datos únicos (extrínsecos)
/// </summary>
1 referencia
public class Resource
{
    private readonly string _uniqueId;
    private readonly string _name;
    private readonly int _size;
    private readonly ResourceFlyweight _flyweight;

    0 referencias
    public Resource(string uniqueId, string name, int size, ResourceFlyweight flyweight)
    {
        _uniqueId = uniqueId;
        _name = name;
        _size = size;
        _flyweight = flyweight;
    }

    0 referencias
    public void Display()
    {
        _flyweight.Display(_uniqueId, _name, _size);
    }
}
```



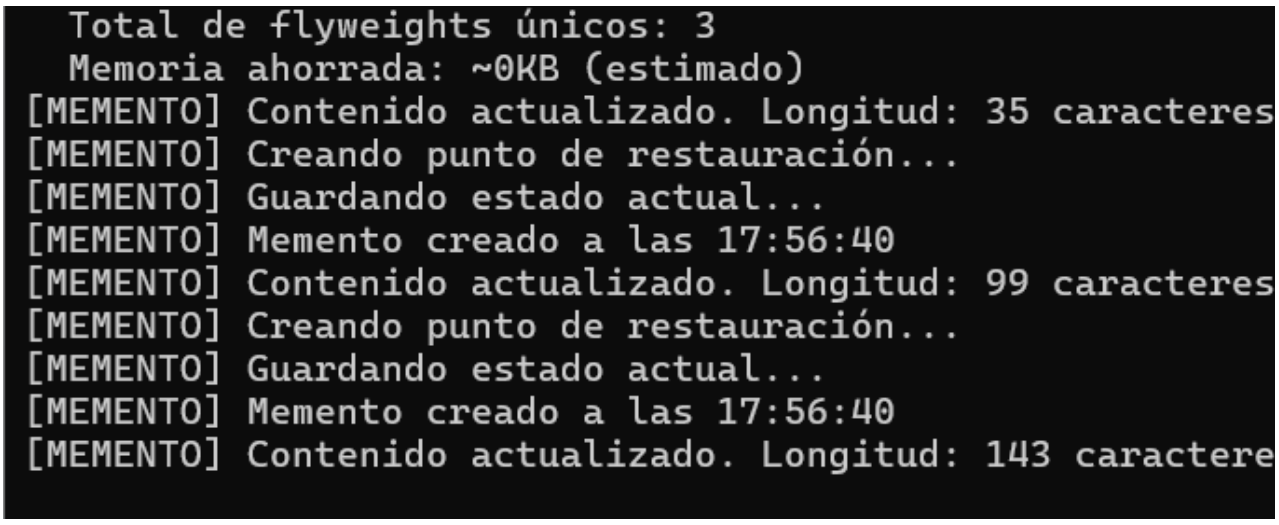
muestra cómo el sistema crea y reutiliza recursos (videos, documentos y quizzes) usando el patrón Flyweight para ahorrar memoria.”

Memento: **DraftMemento** permite guardar y restaurar versiones de borradores de cursos, facilitando la funcionalidad de "deshacer".




```
namespace PlataformaAcademicaModular.ResourceOptimizer;

/// <summary>
/// PATRÓN MEMENTO: Captura y externaliza el estado interno de un objeto
/// Permite restaurar el objeto a ese estado más tarde
/// </summary>
5 referencias
public class DraftMemento
{
    private readonly string _content;
    private readonly DateTime _timestamp;
    private readonly string _author;

    1 referencia
    public DraftMemento(string content, string author)
    {
        _content = content;
        _author = author;
        _timestamp = DateTime.Now;
        Console.WriteLine($"[MEMENTO] Memento creado a las {_timestamp:HH:mm:ss}");
    }

    1 referencia
    public string GetContent() => _content;
    1 referencia
    public DateTime GetTimestamp() => _timestamp;
    0 referencias
    public string GetAuthor() => _author;

    1 referencia
    public void ShowInfo()
    {
        Console.WriteLine($"  Guardado: {_timestamp:dd/MM/yyyy HH:mm:ss} por {_author}");
        Console.WriteLine($"  Contenido: {_content.Substring(0, Math.Min(50, _content.Length))}...");
    }
}

/// <summary>
/// Originador: Borrador de curso
/// </summary>
3 referencias
public class CourseDraft
{
    private string _content;
    private string _author;

    0 referencias
    public CourseDraft(string author)
    {
```

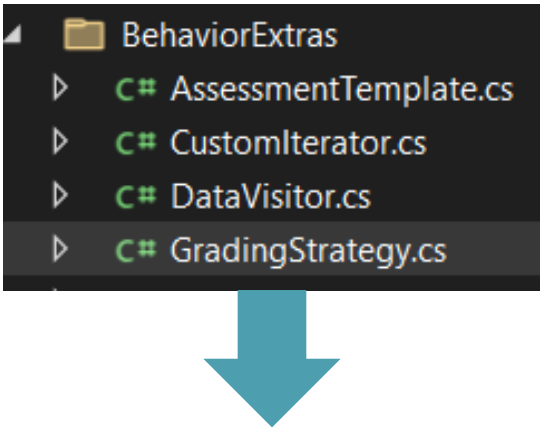
```
    {
        _author = author;
        _content = string.Empty;
    }

    0 referencias
    public void Write(string text)
    {
        _content += text;
        Console.WriteLine($"[MEMENTO] Contenido actualizado. Longitud: {_content.Length} caracteres");
    }

    0 referencias
    public void SetContent(string content)
    {
        _content = content;
        Console.WriteLine($"[MEMENTO] Contenido establecido directamente");
    }

    1 referencia
    public DraftMemento Save()
    {
        Console.WriteLine("[MEMENTO] Guardando estado actual...");
        return new DraftMemento(_content, _author);
    }

    1 referencia
    public void Restore(DraftMemento memento)
    {
        _content = memento.GetContent();
        Console.WriteLine($"[MEMENTO] Estado restaurado desde {memento.GetTimestamp():HH:mm:ss}");
    }

    0 referencias
    public void Display()
    {
        Console.WriteLine($"\n Borrador de {_author}:");
        Console.WriteLine($"   {_content}");
    }
}

/// <summary>
/// Cuidador: Gestiona el historial de mementos
/// </summary>
1 referencia
public class DraftHistory
{
    private readonly Stack<DraftMemento> _history = new();
    private readonly CourseDraft _draft;

    0 referencias
    public DraftHistory(CourseDraft draft)
    {
        _draft = draft;
    }

    0 referencias
    public void Backup()
    {
        Console.WriteLine("[MEMENTO] Creando punto de restauración...");
        _history.Push(_draft.Save());
    }

    0 referencias
    public void Undo()
    {
        if (_history.Count > 0)
        {
            var memento = _history.Pop();
            Console.WriteLine("[MEMENTO] Deshaciendo cambios...");
            _draft.Restore(memento);
        }
        else
        {
            Console.WriteLine("[MEMENTO] No hay estados anteriores para restaurar");
        }
    }

    0 referencias
    public void ShowHistory()
    {
        Console.WriteLine($"\n[MEMENTO] Historial ({_history.Count} versiones guardadas):");
        int version = _history.Count;
        foreach (var memento in _history)
        {
            Console.WriteLine($"\n  Versión {version--}:");
            memento.ShowInfo();
        }
    }
}
```



El sistema guarda puntos de restauración de borradores de curso, permitiendo deshacer cambios y recuperar estados anteriores.

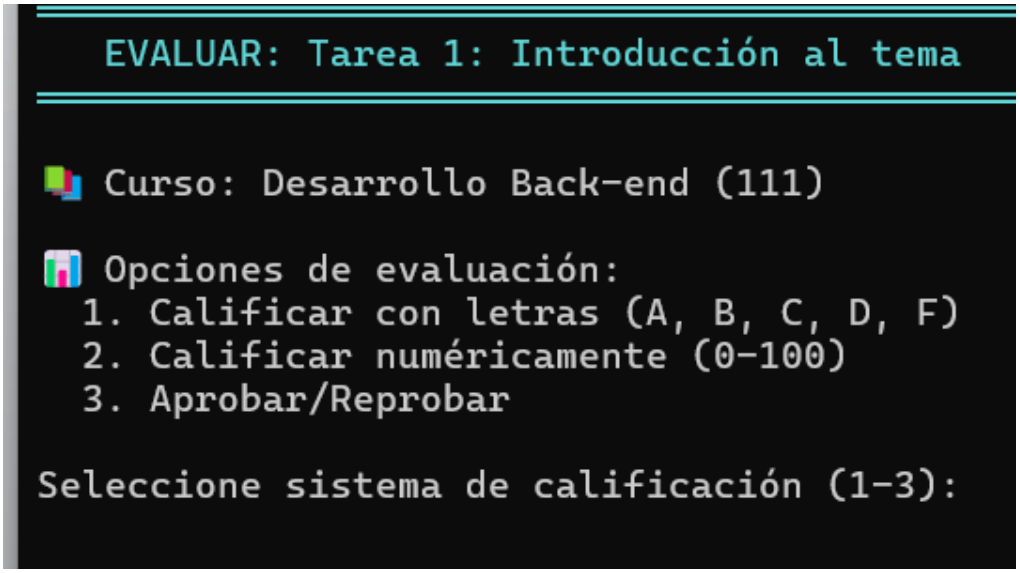
3. Lógica de Negocio y Evaluaciones

Strategy: GradingStrategy permite intercambiar algoritmos de calificación en tiempo de ejecución (Letras A–F, Numérico 0–100, Aprobado/Reprobado).

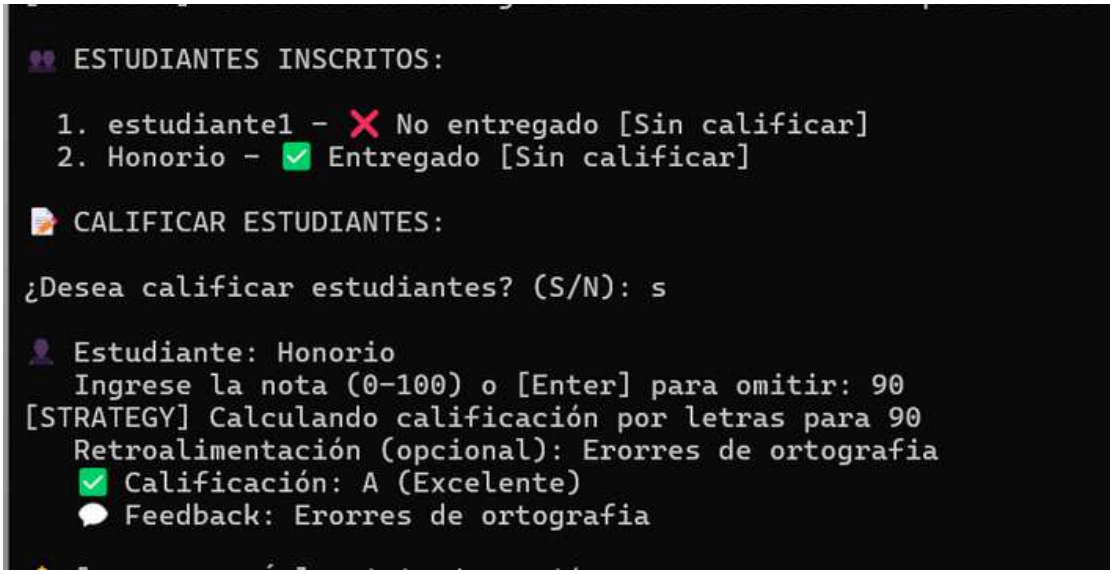



```
namespace PlataformaAcademicaModular.BehaviorExtras;

/// <summary>
/// PATRÓN STRATEGY: Define una familia de algoritmos de calificación
/// Los hace intercambiables y permite que varien independientemente
/// </summary>
7 referencias
public interface IGradingStrategy
{
    4 referencias
    string CalculateGrade(double score);
    6 referencias
    string GetStrategyName();
}

/// <summary>
/// Estrategia: Calificación por letras (A-F)
/// </summary>
5 referencias
public class LetterGradingStrategy : IGradingStrategy
{
    4 referencias
    public string GetStrategyName() => "Calificación por Letras";

    2 referencias
    public string CalculateGrade(double score)
    {
        Console.WriteLine($"[STRATEGY] Calculando calificación por letras para {score}");

        return score switch
        {
            >= 90 => "A (Excelente)",
            >= 80 => "B (Muy Bueno)",
            >= 70 => "C (Bueno)",
            >= 60 => "D (Suficiente)",
            _ => "F (Reprobado)"
        };
    }
}

/// <summary>
/// Estrategia: Calificación por porcentaje
/// </summary>
0 referencias
public class PercentageGradingStrategy : IGradingStrategy
{
    4 referencias
    public string GetStrategyName() => "Calificación por Porcentaje";
```

```
        2 referencias
        public string CalculateGrade(double score)
        {
            Console.WriteLine($"[STRATEGY] Calculando calificación por porcentaje para {score}");
            return $"{score}% - {GetPerformanceLevel(score)}";
        }

        1 referencia
        private string GetPerformanceLevel(double score)
        {
            return score switch
            {
                >= 85 => "Sobresaliente",
                >= 70 => "Notable",
                >= 60 => "Aprobado",
                _ => "Reprobado"
            };
        }
    }

    /// <summary>
    /// Estrategia: Calificación Aprobado/Reprobado
    /// </summary>
    3 referencias
    public class PassFailGradingStrategy : IGradingStrategy
    {
        private readonly double _passingScore;

        2 referencias
        public PassFailGradingStrategy(double passingScore = 60)
        {
            _passingScore = passingScore;
        }

        4 referencias
        public string GetStrategyName() => "Calificación Aprobado/Reprobado";

        2 referencias
        public string CalculateGrade(double score)
        {
            Console.WriteLine($"[STRATEGY] Calculando aprobado/reprobado para {score}");
            return score >= _passingScore ? "✅ APROBADO" : "❌ REPROBADO";
        }
    }
    /// <summary>
    /// Contexto que utiliza las estrategias de calificación
    /// </summary>
    4 referencias
    public class GradeCalculator
    {
        private IGradingStrategy _strategy;

        3 referencias
        public GradeCalculator(IGradingStrategy strategy)
        {
            _strategy = strategy;
            Console.WriteLine($"[STRATEGY] Calculadora configurada con: {strategy.GetStrategyName()}");
        }

        1 referencia
        public void SetStrategy(IGradingStrategy strategy)
        {
            _strategy = strategy;
            Console.WriteLine($"[STRATEGY] Estrategia cambiada a: {strategy.GetStrategyName()}");
        }

        4 referencias
        public string Grade(double score)
        {
            return _strategy.CalculateGrade(score);
        }
    }
```

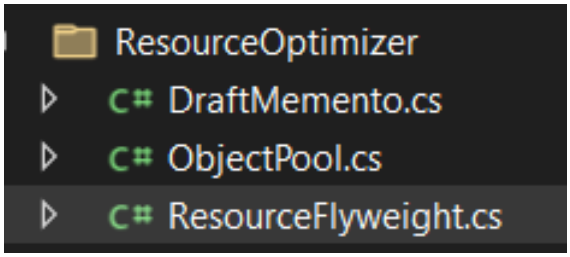


El sistema permite elegir entre letras, porcentaje o aprobado/reprobado antes de evaluar al estudiante.



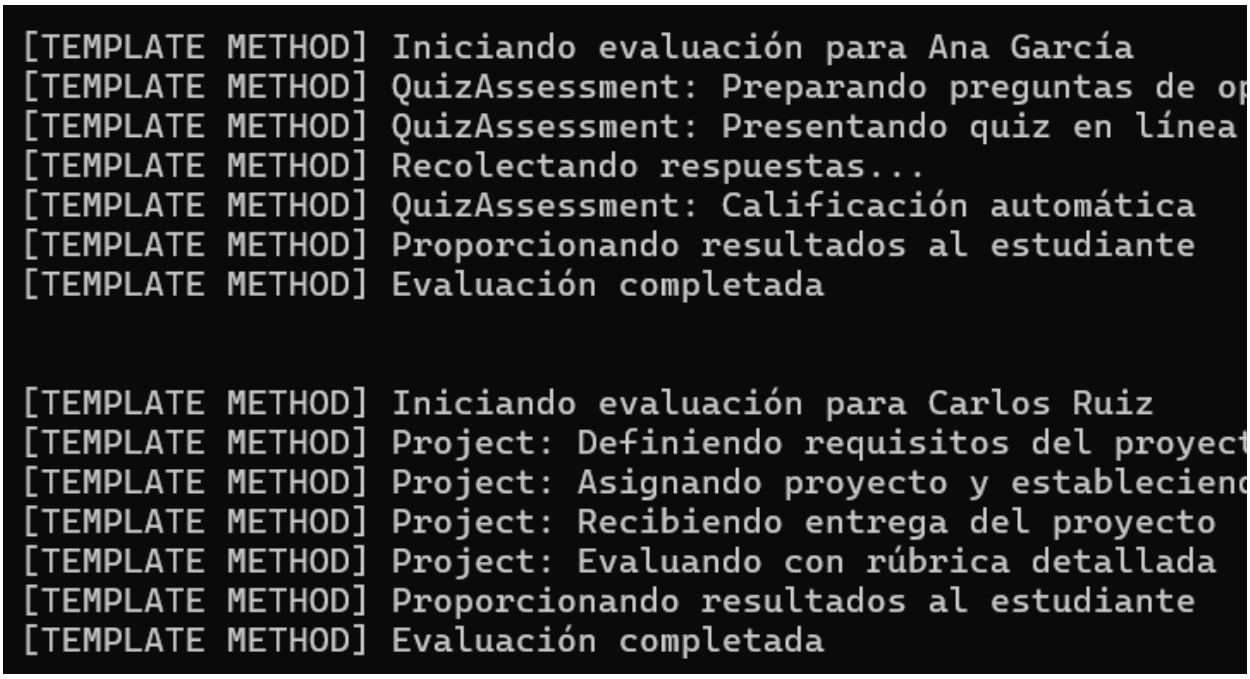
Se calcula la calificación por letras y se registra retroalimentación personalizada para el estudiante.

Template Method: **AssessmentTemplate** define el esqueleto del algoritmo de evaluación, permitiendo que las subclases redefinan pasos específicos sin cambiar la estructura.



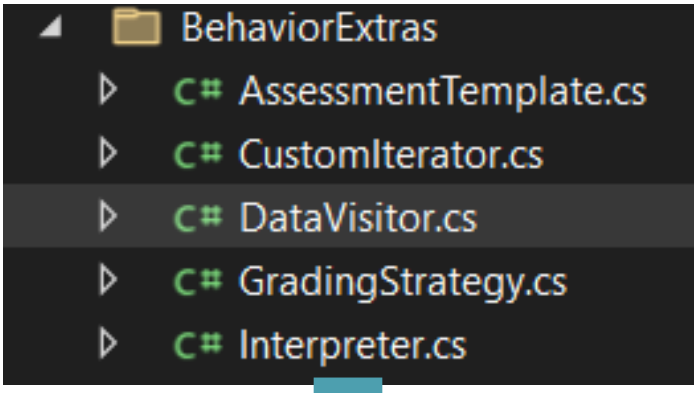

```
namespace PlataformaAcademicaModular.BehaviorExtras;

/// <summary>
/// PATRÓN TEMPLATE METHOD: Define el esqueleto del proceso de evaluación
/// Las subclases implementan pasos específicos
/// </summary>
3 referencias
public abstract class AssessmentTemplate
{
    // Template Method - algoritmo general
    0 referencias
    public void ConductAssessment(string studentName)
    {
        Console.WriteLine($"\n[TEMPLATE METHOD] Iniciando evaluación para {studentName}");

        PrepareAssessment();
        AdministerTest();
        CollectAnswers();
        GradeAssessment();
        ProvideResults();

        Console.WriteLine("[TEMPLATE METHOD] Evaluación completada\n");
    }

    // Pasos abstractos - deben ser implementados
    4 referencias
    protected abstract void PrepareAssessment();
    4 referencias
    protected abstract void AdministerTest();
    4 referencias
    protected abstract void GradeAssessment();

    // Pasos con implementación por defecto
    2 referencias
    protected virtual void CollectAnswers()
    {
        Console.WriteLine("[TEMPLATE METHOD] Recolectando respuestas...");
    }

    3 referencias
    protected virtual void ProvideResults()
    {
        Console.WriteLine("[TEMPLATE METHOD] Proporcionando resultados al estudiante");
    }
}
```

```
0 referencias
public class QuizAssessment : AssessmentTemplate
{
    2 referencias
    protected override void PrepareAssessment()
    {
        Console.WriteLine("[TEMPLATE METHOD] QuizAssessment: Preparando preguntas de opción múltipl
    }

    2 referencias
    protected override void AdministerTest()
    {
        Console.WriteLine("[TEMPLATE METHOD] QuizAssessment: Presentando quiz en línea");
    }

    2 referencias
    protected override void GradeAssessment()
    {
        Console.WriteLine("[TEMPLATE METHOD] QuizAssessment: Calificación automática");
    }
}

/// <summary>
/// Evaluación tipo Examen escrito
/// </summary>
0 referencias
public class WrittenExamAssessment : AssessmentTemplate
{
    2 referencias
    protected override void PrepareAssessment()
    {
        Console.WriteLine("[TEMPLATE METHOD] WrittenExam: Preparando preguntas abiertas");
    }

    2 referencias
    protected override void AdministerTest()
    {
        Console.WriteLine("[TEMPLATE METHOD] WrittenExam: Distribuyendo examen físico");
    }

    2 referencias
    protected override void GradeAssessment()
    {
        Console.WriteLine("[TEMPLATE METHOD] WrittenExam: Calificación manual por profesor");
    }

    3 referencias
    protected override void ProvideResults()
    {
        base.ProvideResults();
        Console.WriteLine("[TEMPLATE METHOD] WrittenExam: Enviando retroalimentación detallada");
    }
}

/// <summary>
/// Evaluación tipo Proyecto
/// </summary>
0 referencias
public class ProjectAssessment : AssessmentTemplate
{
    2 referencias
    protected override void PrepareAssessment()
    {
        Console.WriteLine("[TEMPLATE METHOD] Project: Definiendo requisitos del proyecto");
    }

    2 referencias
    protected override void AdministerTest()
    {
        Console.WriteLine("[TEMPLATE METHOD] Project: Asignando proyecto y estableciendo fecha lim
    }

    2 referencias
    protected override void CollectAnswers()
    {
        Console.WriteLine("[TEMPLATE METHOD] Project: Recibiendo entrega del proyecto");
```



Se aplica el flujo fijo de evaluación tipo quiz, con calificación automática y entrega de resultados.

Visitor: **DataVisitor** separa los algoritmos de reporte de la estructura de objetos, permitiendo generar estadísticas complejas sin modificar las clases de datos.




```
namespace PlataformaAcademicaModular.BehaviorExtras;

/// <summary>
/// Elementos de datos visitables
/// </summary>
2 referencias
public interface IDataElement
{
    3 referencias
    void Accept(IDataVisitor visitor);
}

/// <summary>
/// Elemento: Registro de estudiante
/// </summary>
5 referencias
public class StudentRecord : IDataElement
{
    8 referencias
    public string Name { get; set; } = string.Empty;
    7 referencias
    public List<double> Grades { get; set; } = new();
    4 referencias
    public int Absences { get; set; }

    2 referencias
    public void Accept(IDataVisitor visitor)
    {
        visitor.VisitStudentRecord(this);
    }
}
```

```
/// <summary>
/// Elemento: Registro de curso
/// </summary>
4 referencias
public class CourseRecord : IDataElement
{
    7 referencias
    public string CourseName { get; set; } = string.Empty;
    3 referencias
    public int EnrolledStudents { get; set; }
    4 referencias
    public double AverageGrade { get; set; }

    1 referencia
    public void Accept(IDataVisitor visitor)
    {
        visitor.VisitCourseRecord(this);
    }
}
public interface IDataVisitor
{
    4 referencias
    void VisitStudentRecord(StudentRecord record);
    4 referencias
    void VisitCourseRecord(CourseRecord record);
}

/// <summary>
/// Visitor: Calculador de estadisticas
/// </summary>
1 referencia
public class StatisticsCalculatorVisitor : IDataVisitor
{
    2 referencias
    public double TotalAverage { get; private set; }
    3 referencias
    public int TotalStudents { get; private set; }

    2 referencias
    public void VisitStudentRecord(StudentRecord record)
    {
        if (record.Grades.Count > 0)
        {
            var avg = record.Grades.Average();
            TotalAverage += avg;
            TotalStudents++;
            Console.WriteLine($"[VISITOR] Estadisticas de {record.Name}: Promedio {avg:F
        }
    }

    2 referencias
    public void VisitCourseRecord(CourseRecord record)
    {
        Console.WriteLine($"[VISITOR] Estadisticas del curso {record.CourseName}:");
        Console.WriteLine($"  Estudiantes inscritos: {record.EnrolledStudents}");
        Console.WriteLine($"  Promedio del curso: {record.AverageGrade:F2}");
    }

    0 referencias
    public double GetOverallAverage()
    {
        return TotalStudents > 0 ? TotalAverage / TotalStudents : 0;
    }
}
```

```
        return Errors.Count == 0;
    }
}

/// <summary>
/// Visitor: Generador de reportes
/// </summary>
0 referencias
public class ReportGeneratorVisitor : IDataVisitor
{
    private readonly System.Text.StringBuilder _report = new();

    2 referencias
    public void VisitStudentRecord(StudentRecord record)
    {
        _report.AppendLine($"📊 Estudiante: {record.Name}");
        _report.AppendLine($"   Calificaciones: {string.Join(", ", record.Grades.Select(g => g.ToString("F1")))}");
        _report.AppendLine($"   Promedio: {(record.Grades.Count > 0 ? record.Grades.Average() : 0):F2}");
        _report.AppendLine($"   Ausencias: {record.Absences}");
        Console.WriteLine($"[VISITOR] Reporte generado para {record.Name}");
    }

    2 referencias
    public void VisitCourseRecord(CourseRecord record)
    {
        _report.AppendLine($"📚 Curso: {record.CourseName}");
        _report.AppendLine($"   Estudiantes: {record.EnrolledStudents}");
        _report.AppendLine($"   Promedio: {record.AverageGrade:F2}");
        Console.WriteLine($"[VISITOR] Reporte generado para {record.CourseName}");
    }

    0 referencias
    public string GetReport()
    {
        return _report.ToString();
    }
}
```

```
[VISITOR] Estadísticas de Ana García: Promedio 88.88, Ausencias 2
[VISITOR] Estadísticas del curso Patrones de Diseño en C#:
  Estudiantes inscritos: 45
  Promedio del curso: 87.50
[VISITOR] Validando registro de Ana García
[VISITOR] Validando registro del curso Patrones de Diseño en C#

[VISITOR] Validación: ✅ Datos válidos
```

Se aplican visitantes para calcular estadísticas y validar registros sin modificar las clases de datos

Iterator: **CustomIterator** proporciona una forma secuencial de acceder a las colecciones de estudiantes o contenidos sin exponer su representación subyacente.

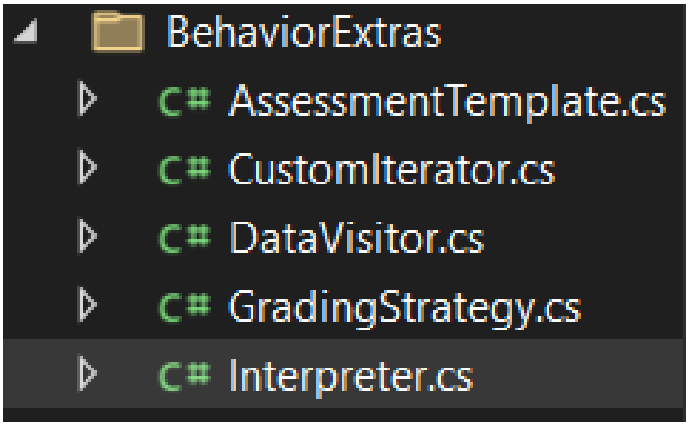
```
namespace PlataformaAcademicaModular.BehaviorExtras;

/// <summary>
/// Colección de estudiantes
/// </summary>
1 referencia
public class StudentCollection
{
    private readonly List<string> _students = new();

    3 referencias
    public void AddStudent(string name)
    {
        _students.Add(name);
    }

    1 referencia
    public ICustomIterator CreateIterator()
    {
        return new StudentIterator(_students);
    }

    0 referencias
    public int Count => _students.Count;
}

/// <summary>
/// PATRÓN ITERATOR: Proporciona una forma de acceder secuencialmente a elementos
/// Sin exponer la representación subyacente
/// </summary>
3 referencias
public interface ICustomIterator
{
    5 referencias
    bool HasNext();
    3 referencias
    string Next();
    2 referencias
    void Reset();
}

/// <summary>
/// Iterador concreto para estudiantes
/// </summary>
2 referencias
public class StudentIterator : ICustomIterator
{
    private readonly List<string> _students;
    private int _position = 0;

    1 referencia
```

```
    1 referencia
    public StudentIterator(List<string> students)
    {
        _students = new List<string>(students);
        Console.WriteLine($"[ITERATOR] Iterador creado para {_students.Count} estudiantes");
    }

    3 referencias
    public bool HasNext()
    {
        return _position < _students.Count;
    }

    2 referencias
    public string Next()
    {
        if (!HasNext())
        {
            throw new InvalidOperationException("No hay más elementos");
        }

        var student = _students[_position];
        _position++;
        Console.WriteLine($"[ITERATOR] Elemento {_position}/{_students.Count}: {student}");
        return student;
    }

    1 referencia
    public void Reset()
    {
        _position = 0;
        Console.WriteLine("[ITERATOR] Iterador reiniciado");
    }
}

/// <summary>
/// Iterador con filtro
/// </summary>
1 referencia
public class FilteredIterator : ICustomIterator
{
    private readonly List<string> _filteredStudents;
    private int _position = 0;

    0 referencias
    public FilteredIterator(List<string> students, Func<string, bool> filter)
    {
        _filteredStudents = students.Where(filter).ToList();
        Console.WriteLine($"[ITERATOR] Iterador filtrado creado: {_filteredStudents.Count} elementos");
```

```
MÓDULO 8: BEHAVIOR EXTRAS

Iterando sobre estudiantes:
[ITERATOR] Iterador creado para 4 estudiantes
[ITERATOR] Elemento 1/4: Ana García
[ITERATOR] Elemento 2/4: Carlos Ruiz
[ITERATOR] Elemento 3/4: María Fernández
[ITERATOR] Elemento 4/4: Juan Pérez
```

El sistema accede secuencialmente a los estudiantes sin exponer la estructura interna de la colección.

Interpreter: **AccessRuleInterpreter** evalúa reglas de acceso complejas definidas en un lenguaje simple para determinar permisos dinámicos.

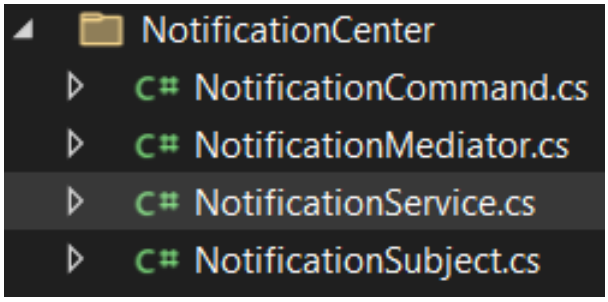



```
namespace PlataformaAcademicaModular.BehaviorExtras;

/// <summary>
/// PATRÓN INTERPRETER: Interpreta y evalúa expresiones en un lenguaje específico
/// Define una gramática y un intérprete para evaluarla
/// </summary>
16 referencias
public interface IExpression
{
    10 referencias
    bool Interpret(Context context);
}

/// <summary>
/// Contexto que contiene información para la interpretación
/// </summary>
7 referencias
public class Context
{
    2 referencias
    public Dictionary<string, bool> Variables { get; } = new();

    2 referencias
    public void SetVariable(string name, bool value)
    {
        Variables[name] = value;
        Console.WriteLine($"[INTERPRETER] Variable '{name}' = {value}");
    }

    1 referencia
    public bool GetVariable(string name)
    {
        return Variables.TryGetValue(name, out bool value) && value;
    }
}
```

```
/// <summary>
/// Expresión terminal: Variable
/// </summary>
9 referencias
public class VariableExpression : IExpression
{
    private readonly string _variableName;

    8 referencias
    public VariableExpression(string variableName)
    {
        _variableName = variableName;
    }

    7 referencias
    7 referencias
    public bool Interpret(Context context)
    {
        bool result = context.GetVariable(_variableName);
        Console.WriteLine($"[INTERPRETER] Evaluando variable '{_variableName}' = {result}");
        return result;
    }
}

/// <summary>
/// Expresión no terminal: AND lógico
/// </summary>
3 referencias
public class AndExpression : IExpression
{
    private readonly IExpression _left;
    private readonly IExpression _right;

    2 referencias
    public AndExpression(IExpression left, IExpression right)
    {
        _left = left;
        _right = right;
    }

    7 referencias
    public bool Interpret(Context context)
    {
        bool result = _left.Interpret(context) && _right.Interpret(context);
        Console.WriteLine($"[INTERPRETER] AND = {result}");
        return result;
    }
}

/// <summary>
/// Expresión no terminal: OR lógico
/// </summary>
3 referencias
public class OrExpression : IExpression
{
    private readonly IExpression _left;
    private readonly IExpression _right;

    2 referencias
    public OrExpression(IExpression left, IExpression right)
    {
        _left = left;
        _right = right;
    }
```

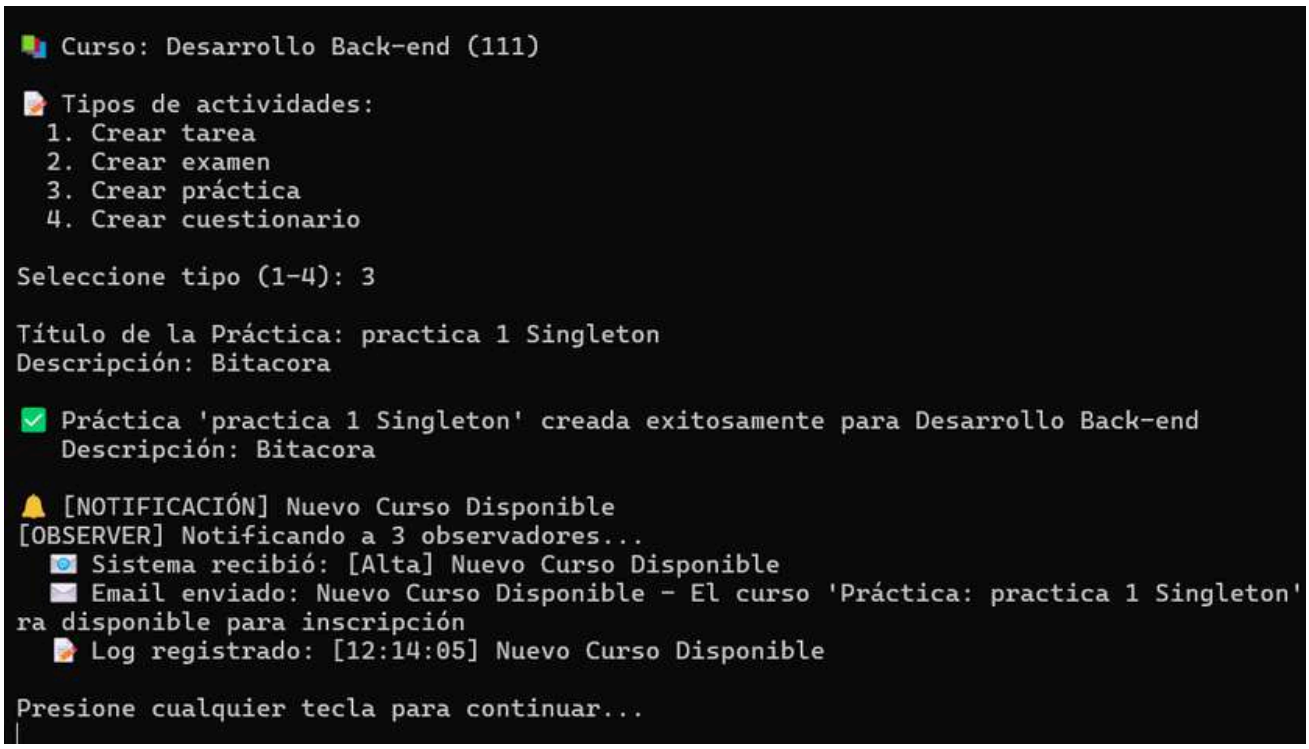
4. Interfaz y Comunicación

Observer: **NotificationService** notifica automáticamente a los estudiantes cuando se crea una actividad o se publica una calificación.



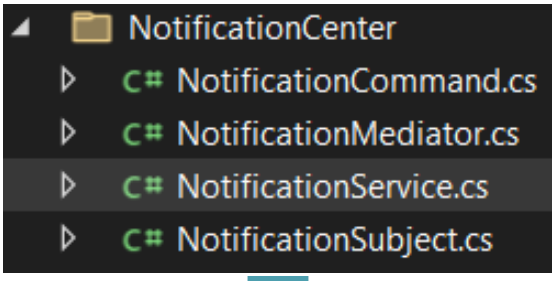

```
18 referencias
public sealed class NotificationService
{
    private static readonly Lazy<NotificationService> _instance = new(() => new NotificationService());
    private readonly NotificationSubject _subject;
    private readonly List<Notification> _notificationHistory = new();

    1 referencia
    private NotificationService()
    {
        _subject = new NotificationSubject();

        // Suscribir observadores por defecto
        _subject.Attach(new UserObserver("Sistema"));
        _subject.Attach(new EmailObserver());
        _subject.Attach(new LoggerObserver());

        Console.WriteLine("🔔 [NOTIFICATION SERVICE] Servicio de notificaciones inicializado");
    }

    13 referencias
    public static NotificationService Instance => _instance.Value;

    /// <summary>
    /// Notifica cuando se registra un nuevo usuario
    /// </summary>
    1 referencia
    public void NotifyUserRegistered(string username, string role)
    {
        var notification = new Notification
        {
            Title = "Nuevo Usuario Registrado",
            Message = $"El usuario '{username}' se ha registrado como {role}",
            Timestamp = DateTime.Now,
            Priority = "Normal"
        };

        SendNotification(notification);
    }
```

```
    /// <summary>
    /// Notifica cuando se crea un nuevo curso
    /// </summary>
    4 referencias
    public void NotifyCourseCreated(string courseName, string courseCode)
    {
        var notification = new Notification
        {
            Title = "Nuevo Curso Disponible",
            Message = $"El curso '{courseName}' ({courseCode}) está ahora disponible para inscripción",
            Timestamp = DateTime.Now,
            Priority = "Alta"
        };

        SendNotification(notification);
    }

    /// <summary>
    /// Notifica inicio de sesión
    /// </summary>
    5 referencias
    public void NotifyUserLogin(string username, string role)
    {
        var notification = new Notification
        {
            Title = "Inicio de Sesión",
            Message = $"{username} ({role}) ha iniciado sesión",
            Timestamp = DateTime.Now,
            Priority = "Baja"
        };

        SendNotification(notification);
    }

    /// <summary>
    /// Envía una notificación y la guarda en el historial
    /// </summary>
    3 referencias
    private void SendNotification(Notification notification)
    {
        Console.WriteLine($"\n🔔 [NOTIFICACIÓN] {notification.Title}");
        _subject.Notify(notification);
        _notificationHistory.Add(notification);
    }

    /// <summary>
    /// Muestra el historial de notificaciones
    /// </summary>
    1 referencia
    public void DisplayNotificationHistory()
    {
        if (_notificationHistory.Count == 0)
        {
            Console.WriteLine("\n📭 No hay notificaciones");
            return;
        }

        Console.WriteLine($"\n📜 HISTORIAL DE NOTIFICACIONES ({_notificationHistory.Count}):");
        Console.WriteLine(new string('─', 70));

        for (int i = _notificationHistory.Count - 1; i >= 0 && i >= _notificationHistory.Count - 10; i--)
        {
            var notif = _notificationHistory[i];
            Console.WriteLine($"  [{notif.Timestamp:HH:mm:ss}] [{notif.Priority}] {notif.Title}");
            Console.WriteLine($"      {notif.Message}");
            Console.WriteLine();
        }
    }

    /// <summary>
    /// Obtiene el conteo de notificaciones
    /// </summary>
    2 referencias
    public int GetNotificationCount() => _notificationHistory.Count;
}
```



Al crear una práctica, el sistema dispara una notificación que es recibida por tres observadores: sistema, correo y registro.

Command: **NotificationCommand** encapsula solicitudes de notificación como objetos, permitiendo encolarlas o registrarlas.



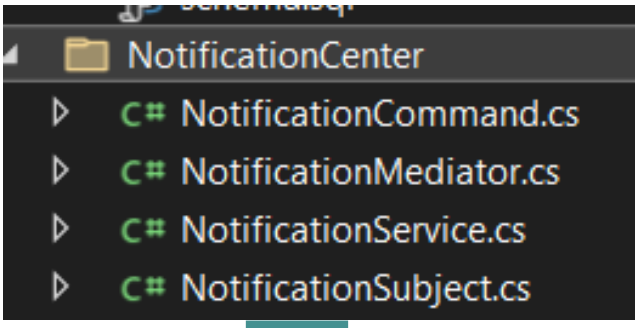

```
namespace PlataformaAcademicaModular.NotificationCenter;

/// <summary>
/// PATRÓN COMMAND: Encapsula una solicitud como un objeto
/// Permite parametrizar clientes con diferentes solicitudes, encolar y deshacer operaciones
/// </summary>
public interface INotificationCommand
{
    void Execute();
    4 referencias
    void Undo();
    10 referencias
    string Description { get; }
}

/// <summary>
/// Comando concreto: Enviar notificación
/// </summary>
1 referencia
public class SendNotificationCommand : INotificationCommand
{
    private readonly NotificationSubject _subject;
    private readonly Notification _notification;
    private bool _executed;

    4 referencias
    public string Description => $"Enviar: {_notification.Title}";

    0 referencias
    public SendNotificationCommand(NotificationSubject subject, Notification notification)
    {
        _subject = subject;
        _notification = notification;
    }

    2 referencias
    public void Execute()
    {
        Console.WriteLine($"[COMMAND] Ejecutando: {Description}");
        _subject.Notify(_notification);
        _executed = true;
    }

    2 referencias
    public void Undo()
    {
        if (_executed)
        {
```

```
            Console.WriteLine($"[COMMAND] Deshaciendo: {Description}");
            // En un sistema real, aquí se revertiría la notificación
            _executed = false;
        }
    }
}

/// <summary>
/// Comando concreto: Suscribir observador
/// </summary>
1 referencia
public class SubscribeCommand : INotificationCommand
{
    private readonly NotificationSubject _subject;
    private readonly INotificationObserver _observer;

    4 referencias
    public string Description => $"Suscribir: {_observer.ObserverName}";

    0 referencias
    public SubscribeCommand(NotificationSubject subject, INotificationObserver observer)
    {
        _subject = subject;
        _observer = observer;
    }

    2 referencias
    public void Execute()
    {
        Console.WriteLine($"[COMMAND] Ejecutando: {Description}");
        _subject.Attach(_observer);
    }

    2 referencias
    public void Undo()
    {
        Console.WriteLine($"[COMMAND] Deshaciendo: {Description}");
        _subject.Detach(_observer);
    }
}

/// <summary>
/// Comando concreto: Desuscribir observador
/// </summary>
1 referencia
public class UnsubscribeCommand : INotificationCommand
{
    private readonly NotificationSubject _subject;
    private readonly INotificationObserver _observer;
```

```
MÓDULO 4: NOTIFICATION CENTER
═══════════════════════════════════════════════════════════

[OBSERVER] Ana García suscrito a notificaciones
[OBSERVER] Sistema de Email suscrito a notificaciones
[OBSERVER] Sistema de Logs suscrito a notificaciones
[COMMAND] Ejecutando: Enviar: Nuevo Curso Disponible
[OBSERVER] Notificando a 3 observadores...
  📧 Ana García recibió: [Alta] Nuevo Curso Disponible
  ✉️ Email enviado: Nuevo Curso Disponible - El curso 'Patrones de Diseñ
  📝 Log registrado: [17:56:40] Nuevo Curso Disponible
```

Cada acción (suscribir, enviar, registrar) se encapsula como comando, permitiendo ejecución controlada y trazable.

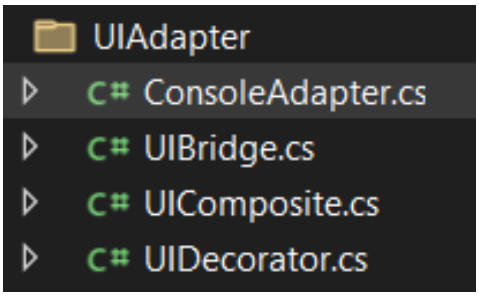
Mediator: **NotificationMediator** centraliza la comunicación compleja entre diferentes módulos del sistema, reduciendo dependencias directas.




```
namespace PlataformaAcademicaModular.NotificationCenter;

/// <summary>
/// Componente base del sistema de notificaciones
/// </summary>
2 referencias
public abstract class NotificationComponent
{
    protected INotificationMediator? Mediator;

    2 referencias
    public void SetMediator(INotificationMediator mediator)
    {
        Mediator = mediator;
    }
}

/// <summary>
/// PATRÓN MEDIATOR: Centraliza la comunicación entre componentes
/// Reduce el acoplamiento entre objetos que interactúan
/// </summary>
3 referencias
public interface INotificationMediator
{
    3 referencias
    void Notify(object sender, string eventType, object? data = null);
}

/// <summary>
/// Mediador concreto para el sistema de notificaciones
/// </summary>
1 referencia
public class NotificationMediator : INotificationMediator
{
    private readonly NotificationSubject _subject;
    private readonly NotificationQueue _queue;
    private readonly NotificationLogger _logger;

    0 referencias
    public NotificationMediator(NotificationSubject subject, NotificationQueue queue, NotificationLogger logger)
    {
        _subject = subject;
        _queue = queue;
        _logger = logger;

        // Solo los componentes que heredan de NotificationComponent tienen SetMediator
        _queue.SetMediator(this);
        _logger.SetMediator(this);
    }
```

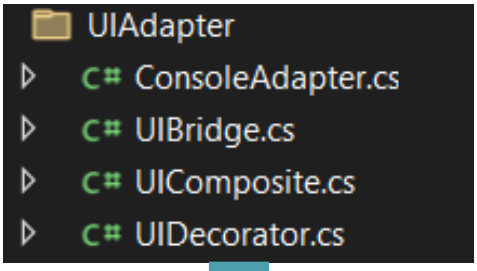
```
3 referencias
public void Notify(object sender, string eventType, object? data = null)
{
    Console.WriteLine($"[MEDIATOR] Evento '{eventType}' recibido de {sender.GetType().Name}");

    switch (eventType)
    {
        case "NotificationCreated":
            if (data is Notification notification)
            {
                _logger.Log($"Nueva notificación: {notification.Title}");
                _queue.Enqueue(notification);
                _subject.Notify(notification);
            }
            break;

        case "NotificationProcessed":
            _logger.Log("Notificación procesada exitosamente");
            break;

        case "QueueEmpty":
            _logger.Log("Cola de notificaciones vacía");
            break;
    }
}

/// <summary>
/// Componente: Cola de notificaciones
/// </summary>
2 referencias
public class NotificationQueue : NotificationComponent
{
    private readonly Queue<Notification> _queue = new();

    1 referencia
    public void Enqueue(Notification notification)
    {
        _queue.Enqueue(notification);
        Console.WriteLine($"[MEDIATOR] Notificación encolada. Total en cola: {_queue.Count}");
    }

    0 referencias
    public Notification? Dequeue()
    {
        if (_queue.Count > 0)
        {
            var notification = _queue.Dequeue();
            Mediator?.Notify(this, "NotificationProcessed");
            return notification;
        }
        Mediator?.Notify(this, "QueueEmpty");
        return null;
    }
}

/// <summary>
/// Componente: Logger de notificaciones
/// </summary>
2 referencias
public class NotificationLogger : NotificationComponent
{
    private readonly List<string> _logs = new();

    3 referencias
    public void Log(string message)
    {
        var logEntry = $"[{DateTime.Now:HH:mm:ss}] {message}";
        _logs.Add(logEntry);
        Console.WriteLine($"[MEDIATOR] Log: {message}");
    }

    0 referencias
    public IReadOnlyList<string> GetLogs() => _logs.AsReadOnly();
}
```

Adapter: **ConsoleAdapter** permite que el sistema interactúe con diferentes interfaces de salida (consola actual, futura API web) sin cambios en la lógica.




```
/// <summary>
/// PATRÓN ADAPTER: Convierte la interfaz de una clase en otra que los clientes esperan
/// Permite que clases con interfaces incompatibles trabajen juntas
/// </summary>
2 referencias
public class ConsoleAdapter : IModernUI
{
    private readonly LegacyConsoleSystem _legacySystem;

    1 referencia
    public ConsoleAdapter(LegacyConsoleSystem legacySystem)
    {
        _legacySystem = legacySystem;
        Console.WriteLine("[ADAPTER] Adaptador de consola creado");
    }

    2 referencias
    public void RenderElement(string content, string style)
    {
        Console.WriteLine($"[ADAPTER] Adaptando renderizado con estilo '{style}'");

        switch (style.ToLower())
        {
            case "error":
                _legacySystem.PrintWithColor(content, ConsoleColor.Red);
                break;
            case "success":
                _legacySystem.PrintWithColor(content, ConsoleColor.Green);
                break;
            case "warning":
                _legacySystem.PrintWithColor(content, ConsoleColor.Yellow);
                break;
            case "info":
                _legacySystem.PrintWithColor(content, ConsoleColor.Cyan);
                break;
            default:
                _legacySystem.PrintText(content);
                break;
        }
    }
}
```

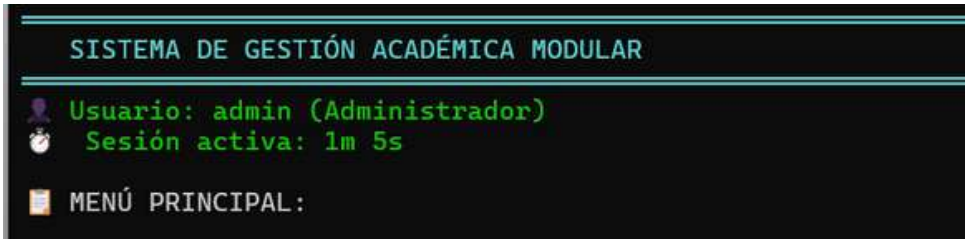
Decorator: **UIDecorator** añade responsabilidades visuales (bordes, colores) a los componentes de la interfaz de forma dinámica.



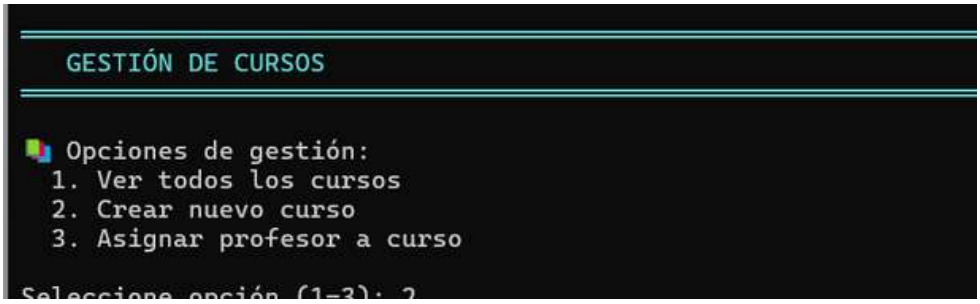

```
/// <summary>
/// Componente base de UI
/// </summary>
public interface IUIElement
{
    12 referencias
    void Display();
}

/// <summary>
/// Componente concreto: Texto simple
/// </summary>
2 referencias
public class SimpleText : IUIElement
{
    private readonly string _text;

    1 referencia
    public SimpleText(string text)
    {
        _text = text;
    }

    7 referencias
    public void Display()
    {
        Console.WriteLine(_text);
    }
}

/// <summary>
/// PATRÓN DECORATOR: Añade responsabilidades adicionales a un objeto dinámicamente
/// Proporciona una alternativa flexible a la herencia para extender funcionalidad
/// </summary>
9 referencias
public abstract class UIDecorator : IUIElement
{
    protected IUIElement _component;

    4 referencias
    protected UIDecorator(IUIElement component)
    {
        _component = component;
    }

    11 referencias
    public virtual void Display()
```

```
        {
            _component.Display();
        }
    }

    /// <summary>
    /// Decorador concreto: Borde
    /// </summary>
    2 referencias
    public class BorderDecorator : UIDecorator
    {
        private readonly string _borderChar;

        1 referencia
        public BorderDecorator(IUIElement component, string borderChar = "=") : base(component)
        {
            _borderChar = borderChar;
        }

        8 referencias
        public override void Display()
        {
            Console.WriteLine($"[DECORATOR] Aplicando borde");
            Console.WriteLine(new string(_borderChar[0], 50));
            _component.Display();
            Console.WriteLine(new string(_borderChar[0], 50));
        }
    }

    /// <summary>
    /// Decorador concreto: Color
    /// </summary>
    2 referencias
    public class ColorDecorator : UIDecorator
    {
        private readonly ConsoleColor _color;

        1 referencia
        public ColorDecorator(IUIElement component, ConsoleColor color) : base(component)
        {
            _color = color;
        }

        8 referencias
        public override void Display()
        {
            Console.WriteLine($"[DECORATOR] Aplicando color {_color}");
            var oldColor = Console.ForegroundColor;
            Console.ForegroundColor = _color;
            _component.Display();
    /// <summary>
    /// Decorador concreto: Padding
    /// </summary>
    1 referencia
    public class PaddingDecorator : UIDecorator
    {
        private readonly int _padding;

        0 referencias
        public PaddingDecorator(IUIElement component, int padding) : base(component)
        {
            _padding = padding;
        }

        8 referencias
        public override void Display()
        {
            Console.WriteLine($"[DECORATOR] Aplicando padding de {_padding} espacios");
            var indent = new string(' ', _padding);
            Console.Write(indent);
            _component.Display();
        }
    }

    /// <summary>
    /// Decorador concreto: Icono
    /// </summary>
    1 referencia
    public class IconDecorator : UIDecorator
    {
        private readonly string _icon;

        0 referencias
        public IconDecorator(IUIElement component, string icon) : base(component)
        {
            _icon = icon;
        }
```

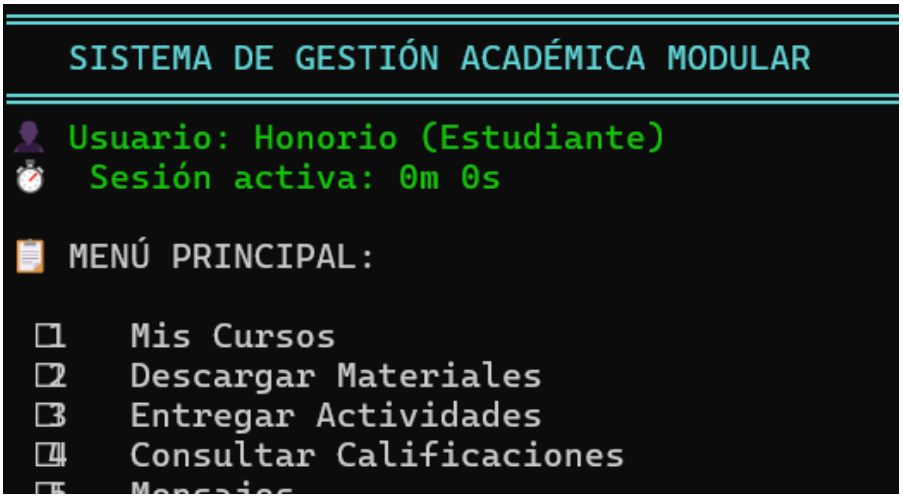
d



Al crear una práctica, el sistema dispara una notificación que es recibida por tres observadores (sistema, correo y registro), garantizando comunicación desacoplada.

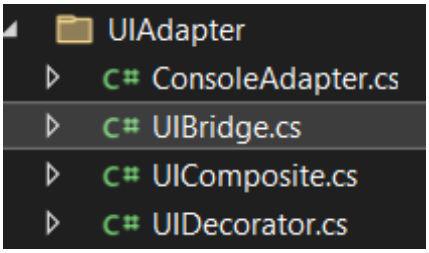


El envío de una notificación se ejecuta como comando, y los observadores suscritos reciben el evento de manera ordenada y registrada en logs.



El sistema adapta la salida visual según el rol del usuario (administrador o estudiante), permitiendo que diferentes perfiles interactúen con la misma lógica sin modificar la base.

Bridge: **UIBridge** desacopla la abstracción de la interfaz de su implementación, permitiendo que ambas evolucionen independientemente (ej. cambiar el motor de renderizado).




```
namespace PlataformaAcademicaModular.UIAdapter;

/// <summary>
/// Abstracción de UI
/// </summary>
7 referencias
public abstract class UIAbstraction
{
    protected IUIRenderer _renderer;

    3 referencias
    protected UIAbstraction(IUIRenderer renderer)
    {
        _renderer = renderer;
    }

    3 referencias
    public abstract void Display(string content);
}

/// <summary>
/// PATRÓN BRIDGE: Separa una abstracción de su implementación
/// Permite que ambas varíen independientemente
/// </summary>
7 referencias
public interface IUIRenderer
{
    5 referencias
    void Render(string content, string format);
}

/// <summary>
/// Implementación concreta: Renderizador de texto
/// </summary>
0 referencias
public class TextRenderer : IUIRenderer
{
    4 referencias
    public void Render(string content, string format)
    {
        Console.WriteLine($"[BRIDGE-TextRenderer] {format}: {content}");
    }
}
```

```
        4 referencias
        public void Render(string content, string format)
        {
            Console.WriteLine($"[BRIDGE-HtmlRenderer] <{format}>{content}</{format}>");
        }
    }

    /// <summary>
    /// Abstracción refinada: Mensaje simple
    /// </summary>
    1 referencia
    public class SimpleMessage : UIAbstraction
    {
        0 referencias
        public SimpleMessage(IUIRenderer renderer) : base(renderer) { }

        1 referencia
        public override void Display(string content)
        {
            Console.WriteLine("[BRIDGE] Mostrando mensaje simple");
            _renderer.Render(content, "p");
        }
    }

    /// <summary>
    /// Abstracción refinada: Alerta
    /// </summary>
    1 referencia
    public class AlertMessage : UIAbstraction
    {
        0 referencias
        public AlertMessage(IUIRenderer renderer) : base(renderer) { }

        1 referencia
        public override void Display(string content)
        {
            Console.WriteLine("[BRIDGE] Mostrando alerta");
            _renderer.Render($"⚠ {content}", "alert");
        }
    }

    /// <summary>
    /// Abstracción refinada: Título
    /// </summary>
    1 referencia
    public class TitleMessage : UIAbstraction
    {
        0 referencias
        public TitleMessage(IUIRenderer renderer) : base(renderer) { }

        1 referencia
        public override void Display(string content)
        {
            Console.WriteLine("[BRIDGE] Mostrando título");
            _renderer.Render(content.ToUpper(), "h1");
        }
    }
```

CONCLUSIONES

La implementación del Sistema de Gestión Académica Modular ha demostrado exitosamente la viabilidad y los beneficios tangibles de aplicar los 23 patrones de diseño en un desarrollo de software moderno con .NET 8.

En primer lugar, la modularidad y el desacoplamiento se evidenciaron drásticamente. Por ejemplo, gracias al patrón Strategy, fue posible añadir nuevos formatos de exportación de reportes sin tocar una sola línea de la lógica de generación de datos. Del mismo modo, el patrón Observer permitió desconectar el núcleo de la lógica de cursos del sistema de notificaciones; el creador del curso no necesita saber quién recibe el mensaje, solo emite el evento.

En segundo lugar, se validó la flexibilidad y mantenibilidad. Patrones como Decorator en la capa de UI permitieron combinaciones visuales infinitas (bordes, colores, iconos) sin crear una explosión de subclases. El uso de Builder simplificó la creación de objetos complejos (Cursos con múltiple contenido) que de otra forma hubieran requerido constructores telescópicos inmanejables.

Finalmente, aunque la aplicación estricta de tantos patrones puede introducir una sobrecarga de clases ("over-engineering") en sistemas triviales, en un sistema complejo como este Plataforma Académica resulta en una arquitectura limpia y comprensible. Cada clase tiene una responsabilidad única y clara. Este proyecto funciona no solo como un prototipo operativo, sino como un catálogo vivo de mejores prácticas de arquitectura de software, cumpliendo así con todos los objetivos planteados.

https://github.com/saiko3000/sistemagestiondecursos.git