TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería en Sistemas Computacionales

MATERIA:
Patrones de diseño

TÍTULO ACTIVIDAD:
Examen

UNIDAD A EVALUAR:
Unidad 4 y 5

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

Solano Cortez Ivan Israel 22210786

NOMBRE DEL MAESTRO (A):
Maribel Guerrero Luis

Introducción

Este proyecto asignado anteriormente implementa un *Sistema Educativo* en C# donde se registran estudiantes, tanto del sistema actual como del sistema antiguo.

Para lograr un diseño flexible, escalable y fácil de mantener, se aplicaron varios **patrones de diseño** que separan responsabilidades, permiten la extensión del programa sin modificar código existente y facilitan la integración con sistemas heredados.

Patrones de diseño utilizados

Factory Method (Creacional)

Para centralizar la creación de distintos tipos de estudiantes sin que el Program.cs tenga que conocer detalles de implementación. Hace el sistema extensible y elimina la creación manual de objetos.

Clases:

EstudianteManagerFactory, EstudianteBase, EvaluacionExtraDecorator, SistemaAdapter

Decorator (Estructural)

Para agregar módulos extra y validación a un estudiante sin modificar la clase principal, permitiendo añadir funcionalidades en capas.

Clases:

EstudianteDecorator, EvaluacionExtraDecorator, ValidacionDecorator

Adapter (Estructural)

Para permitir que el sistema actual interactúe con el sistema antiguo, que funciona de forma distinta.

Se adaptó su método RegistrarAlumno al modelo IEstudiante.

Clases:

SistemaAdapter, ISistemaAntiguo, SistemaAntiguo

State (Comportamiento)

Para que el estado del estudiante (Aprobado, En riesgo, Reprobado) dependa de su promedio final y pueda cambiar dinámicamente.

Clases:

IEstadoEstudiante, EstadoAprobado, EstadoEnRiesgo, EstadoReprobado

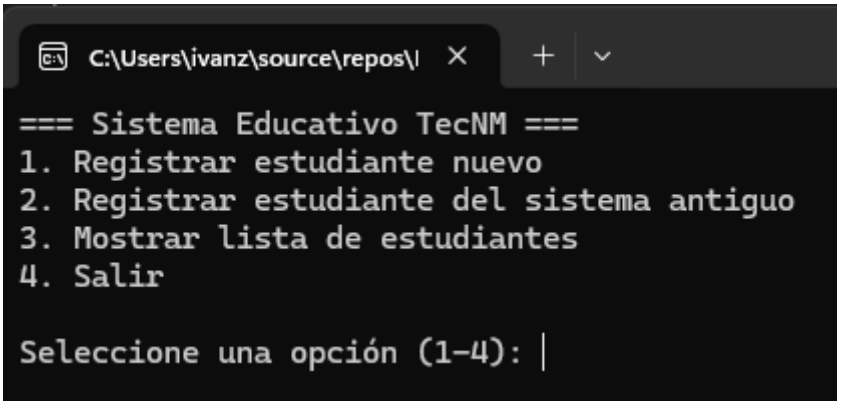
Arquitectura en Capas (From Mud to Structure)

Aunque sea una aplicación de consola, el proyecto está dividido en capas lógicas:

- Presentación → Program.cs
- Lógica de Negocio → Decorators, Factory, States
- Integración → Adapter
- Modelo → IEstudiante, EstudianteBase

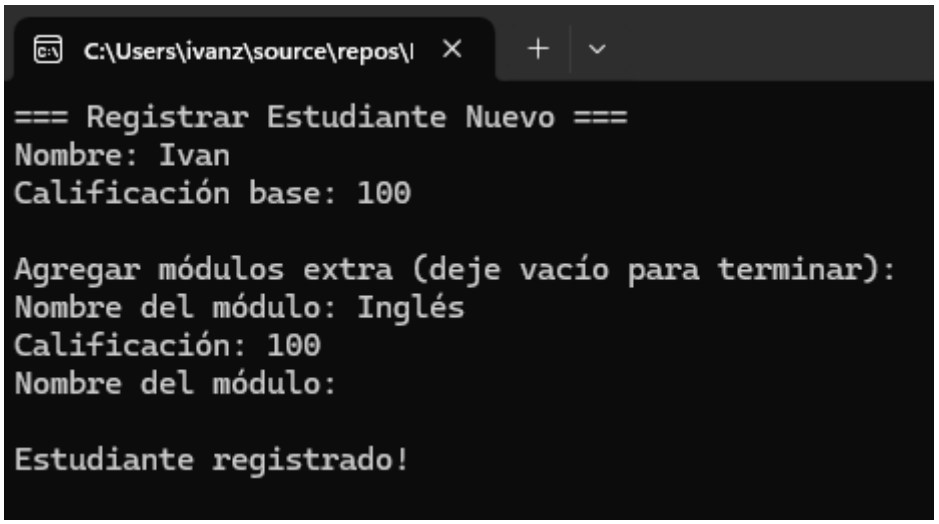
Lo que mantiene un código ordenado, escalable y fácil de revisar.

Programa en ejecución



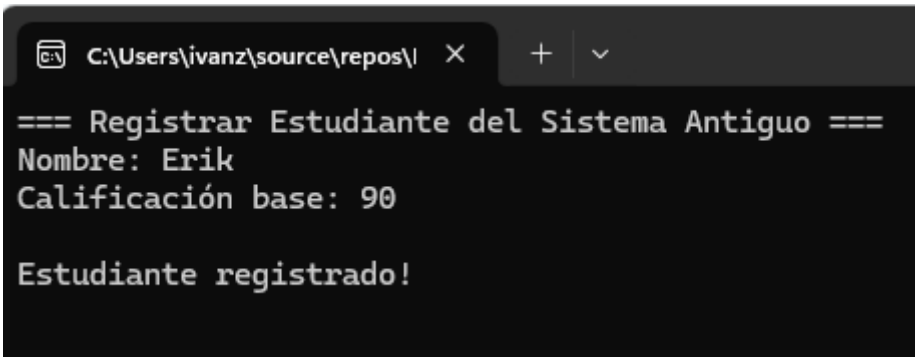
Como primera impresión al ejecutar el programa, aparece un menú con 4 opciones dependiendo que se quiera realizar.

Al seleccionar la primera opción, nos llevará a la sección para registrar un nuevo alumno, el cual utiliza **Factory Method** para no tener que estar creando alumnos de una manera poco eficiente y que consuma más memoria de lo que debería aparte que se hace uso del **patrón de diseño Decorator** que se encarga de poder agregar módulos extra con nombre y calificaciones propias.



Y de esa manera quedaría registrado el alumno en el sistema actual utilizando el **patrón de diseño Factory Method**.

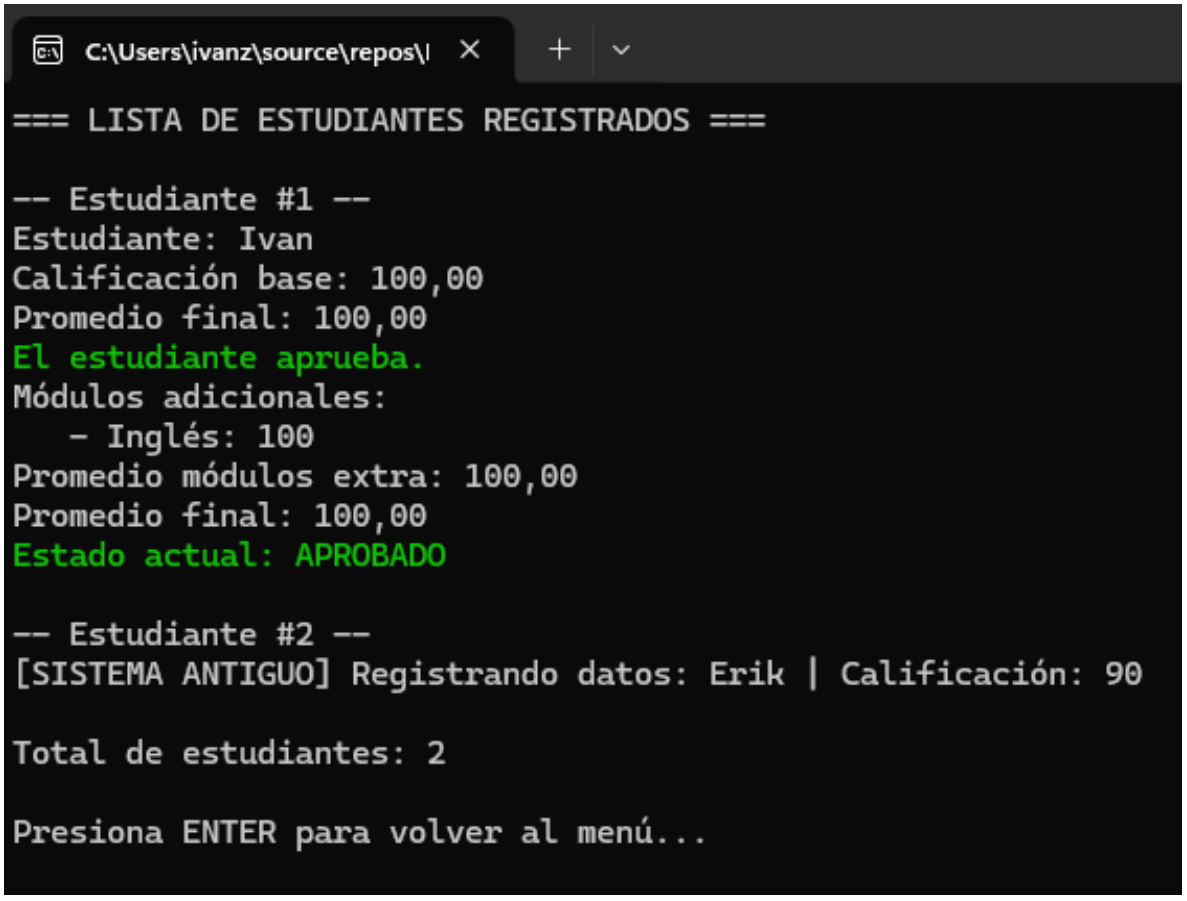
Ahora en la siguiente opción del menú principal (2), el programa nos deja tener compatibilidad con un sistema antiguo para poder ingresar alumnos, esto gracias al **patrón de diseño Adapter** ya que permite que no funcionen ambos sistemas en uno solo.



Y la última opción que nos permite visualizar todos los alumnos ingresados a ambos sistemas, en este apartado se hace uso de todos los patrones en s'i, ya que muestra toda la información detallada, pero se resalta el uso del **patrón de diseño State,** ya que es el que hace la validación de si un alumno aprueba o no y da la descripción final en esta lista de alumnos registrados.

Elección del estado dentro de "ValidacionDecorador.cs"

```
if (promedioFinal >= 70)
    estado = new EstadoAprobado();
else if (promedioFinal >= 60)
    estado = new EstadoEnRiesgo();
else
    estado = new EstadoReprobado();
estado.MostrarEstado();
```



Y por último la opción de salir que simplemente cierra el programa.

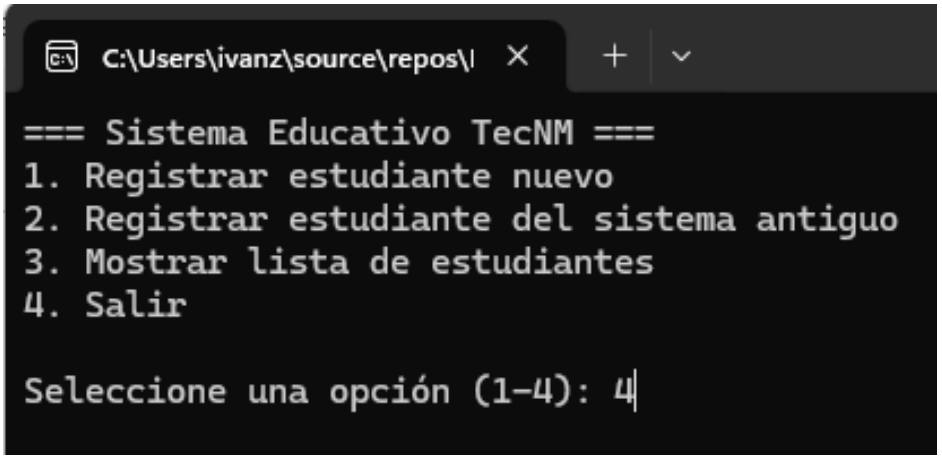
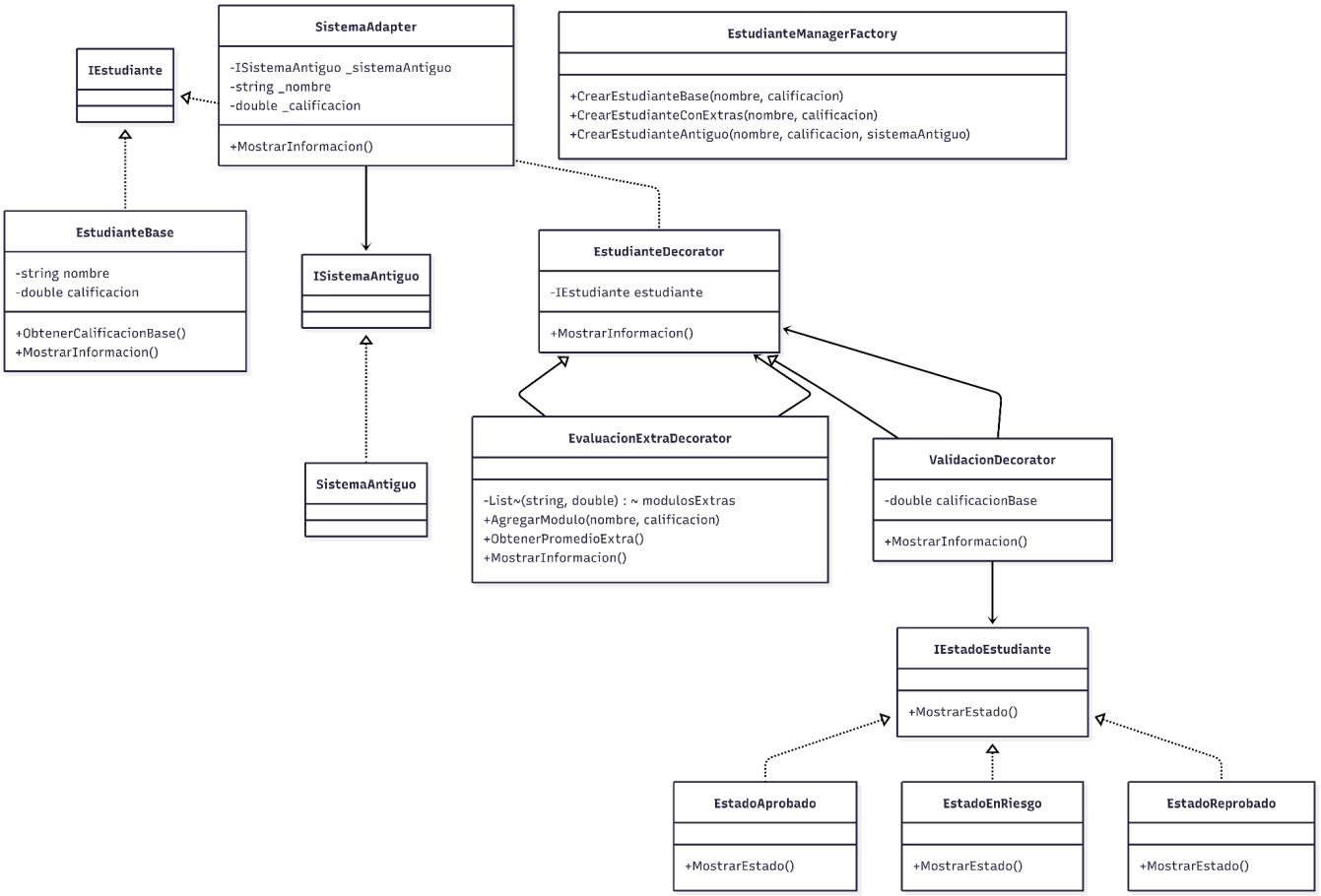


Diagrama UML



Conclusión

La implementación de este sistema educativo permitió integrar varios patrones de diseño que hicieron el proyecto más flexible, escalable y fácil de mantener. Gracias al uso de Factory Method, Decorator, Adapter y State, el sistema puede crear distintos tipos de estudiantes, extender sus funcionalidades sin modificar código existente, comunicarse con un sistema antiguo y manejar estados dinámicos según su desempeño. En conjunto, estos patrones dieron estructura profesional al proyecto y demostraron cómo un buen diseño mejora la calidad y la organización del software.