TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería Informática

MATERIA:
Patrones de diseño de software

TÍTULO ACTIVIDAD:

Examen unidad 3

UNIDAD A EVALUAR:
3

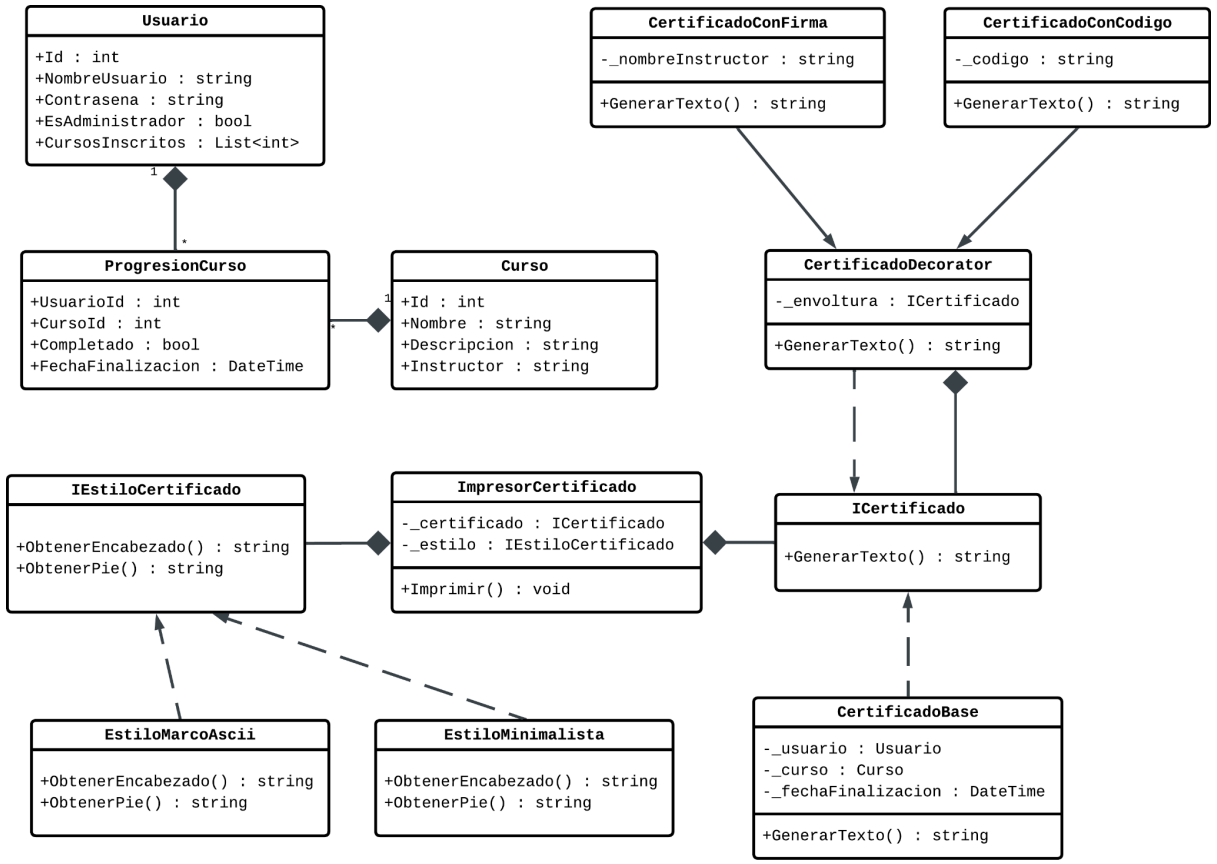
NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

Herrera Sandoval Jose Miguel 21212344

NOMBRE DEL MAESTRO (A):

Maribel Guerrero Luis

Diagrama UML



Código

Clase Curso

```
namespace PatronesU3Examenv1
{
    8 referencias
    class Curso
    {
        16 referencias
        public int Id { get; set; }
        11 referencias
        public string Nombre { get; set; }
        8 referencias
        public string Instructor { get; set; }
    }
}
```

Clase Usuario

```
namespace PatronesU3Examenv1
{

    class Usuario
    {

        public int Id { get; set; }

        public string NombreUsuario { get; set; }

        public string Contraseña { get; set; }

        public bool EsAdministrador { get; set; }
        7 referencias
        public List<int> CursosInscritos { get; set; } = new List<int>();
    }
}
```

Clase ProgresoCurso

```
namespace PatronesU3Examenv1
{

    class ProgresoCurso
    {

        public int UsuarioId { get; set; }

        public int CursoId { get; set; }

        public bool Completado { get; set; }

        public DateTime? FechaFinalizacion { get; set; }
    }
}
```

Interface ICertificado

```
namespace PatronesU3Examenv1
{
    9 referencias
    interface ICertificado
    {
        8 referencias
        string GenerarTexto();
    }
}
```

Clase CertificadoDecorator

```
namespace PatronesU3Examenv1
{
    5 referencias
    abstract class CertificadoDecorator : ICertificado
    {
        protected readonly ICertificado _envoltura;

        2 referencias
        protected CertificadoDecorator(ICertificado certificado)
        {
            _envoltura = certificado;
        }

        7 referencias
        public virtual string GenerarTexto()
        {
            return _envoltura.GenerarTexto();
        }
    }
}
```

Clase CertificadoBase

```
namespace PatronesU3Examenv1
{

    class CertificadoBase : ICertificado
    {
        private readonly Usuario _usuario;
        private readonly Curso _curso;
        private readonly DateTime _fechaFinalizacion;


        public CertificadoBase(Usuario usuario, Curso curso, DateTime fechaFinalizacion)
        {
            _usuario = usuario;
            _curso = curso;
            _fechaFinalizacion = fechaFinalizacion;
        }


        public virtual string GenerarTexto()
        {
            return
                $"CERTIFICADO DE FINALIZACIÓN\n" +
                $"Alumno: {_usuario.NombreUsuario}\n" +
                $"Curso: {_curso.Nombre}\n" +
                $"Área: Tecnologías de la Información\n" +
                $"Fecha de finalización: {_fechaFinalizacion:dd/MM/yyyy}\n";
        }
    }
}
```

Clase CertificadoConFirma

```
namespace PatronesU3Examenv1
{
    2 referencias
    class CertificadoConFirma : CertificadoDecorator
    {
        private readonly string _nombreInstructor;

        1 referencia
        public CertificadoConFirma(ICertificado certificado, string nombreInstructor)
            : base(certificado)
        {
            _nombreInstructor = nombreInstructor;
        }

        6 referencias
        public override string GenerarTexto()
        {
            string baseTexto = base.GenerarTexto();
            return baseTexto + $"\nFirma del instructor: {_nombreInstructor}\n";
        }
    }
}
```

Clase CertificadoConCodigo

```
namespace PatronesU3Examenv1
{
    2 referencias
    class CertificadoConCodigo : CertificadoDecorator
    {
        private readonly string _codigo;

        1 referencia
        public CertificadoConCodigo(ICertificado certificado, string codigo)
            : base(certificado)
        {
            _codigo = codigo;
        }

        6 referencias
        public override string GenerarTexto()
        {
            string baseTexto = base.GenerarTexto();
            return baseTexto + $"Código interno: {_codigo}\n";
        }
    }
}
```

Interface IEstiloCertificado

```
namespace PatronesU3Examenv1
{
    5 referencias
    interface IEstiloCertificado
    {
        3 referencias
        string ObtenerEncabezado();
        3 referencias
        string ObtenerPie();
    }
}
```

Clase EstiloAscii

```
namespace PatronesU3Examenv1
{

    class EstiloAscii : IEstiloCertificado
    {

        public string ObtenerEncabezado()
        {
            return "+-----------------------------------------------+\n" +
                   "|                CERTIFICADO TI                 |\n" +
                   "+-----------------------------------------------+";
        }


        public string ObtenerPie()
        {
            return "+-----------------------------------------------+\n" +
                   "|         Plataforma de Cursos en TI            |\n" +
                   "+-----------------------------------------------+";
        }
    }
}
```

Clase EstiloMinimalista

```
namespace PatronesU3Examenv1
{

    class EstiloMinimalista : IEstiloCertificado
    {

        public string ObtenerEncabezado()
        {
            return "================ CERTIFICADO ================";
        }

        2 referencias
        public string ObtenerPie()
        {
            return "=============================================";
        }
    }
}
```

Clase ImprimirCertificado

```
namespace PatronesU3Examenv1
{
    2 referencias
    class ImprimirCertificado
    {
        private readonly ICertificado _certificado;
        private readonly IEstiloCertificado _estilo;

        1 referencia
        public ImprimirCertificado(ICertificado certificado, IEstiloCertificado estilo)
        {
            _certificado = certificado;
            _estilo = estilo;
        }

        1 referencia
        public void Imprimir()
        {
            Console.WriteLine();
            Console.WriteLine(_estilo.ObtenerEncabezado());
            Console.WriteLine();
            Console.WriteLine(_certificado.GenerarTexto());
            Console.WriteLine(_estilo.ObtenerPie());
            Console.WriteLine();
        }
    }
}
```

Clase Program

```
namespace PatronesU3Examenv1
{
    0 referencias
    class Program
    {
        static List<Usuario> Usuarios = new List<Usuario>();
        static List<Curso> Cursos = new List<Curso>();
        static List<ProgresoCurso> Progresos = new List<ProgresoCurso>();

        static int ultimoIdUsuario = 0;
        static int ultimoIdCurso = 0;

        0 referencias
        static void Main(string[] args)
        {
            InicializarDatosDemo();
            MenuPrincipal();
        }

        1 referencia
        static void InicializarDatosDemo()
        {
            Usuarios.Add(new Usuario
            {
                Id = ++ultimoIdUsuario,
                NombreUsuario = "admin",
                Contraseña = "admin123",
                EsAdministrador = true
            });

            Cursos.Add(new Curso
            {
                Id = ++ultimoIdCurso,
                Nombre = "Fundamentos de Programación en Python",
                Instructor = "Ing. Gonzalez"
            });

            Cursos.Add(new Curso
            {
                Id = ++ultimoIdCurso,
                Nombre = "Introducción a Redes de Computadoras",
                Instructor = "Mtro. Hinojoza"
            });

            Cursos.Add(new Curso
            {
                Id = ++ultimoIdCurso,
                Nombre = "Bases de Datos MySQL",
                Instructor = "Lic. Ramirez"
            });
        }
```

```
static void MenuPrincipal()
{
    while (true)
    {
        Console.Clear();
        Console.WriteLine("=== PLATAFORMA DE CURSOS EN TI ===");
        Console.WriteLine("1. Registrarse");
        Console.WriteLine("2. Iniciar sesión");
        Console.WriteLine("3. Salir");
        Console.Write("Selecciona una opción: ");
        string opcion = Console.ReadLine();

        switch (opcion)
        {
            case "1":
                RegistrarUsuario();
                break;
            case "2":
                IniciarSesion();
                break;
            case "3":
                Console.WriteLine("Saliendo del sistema.");
                return;
            default:
                Console.WriteLine("Opción no válida. Presiona una tecla para continuar.");
                Console.ReadKey();
                break;
        }
    }
}
```

```
1 referencia
static void RegistrarUsuario()
{
    Console.Clear();
    Console.WriteLine("=== REGISTRO DE USUARIO ===");
    Console.Write("Nombre de usuario: ");
    string nombre = Console.ReadLine();

    if (Usuarios.Any(u => u.NombreUsuario.Equals(nombre, StringComparison.OrdinalIgnoreCase)))
    {
        Console.WriteLine("Ese nombre de usuario ya existe.");
        Console.ReadKey();
        return;
    }

    Console.Write("Contraseña: ");
    string contraseña = Console.ReadLine();

    var nuevo = new Usuario
    {
        Id = ++ultimoIdUsuario,
        NombreUsuario = nombre,
        Contraseña = contraseña,
        EsAdministrador = false
    };

    Usuarios.Add(nuevo);
    Console.WriteLine("Usuario registrado con éxito.");
    Console.ReadKey();
}
```

```
1 referencia
static void IniciarSesion()
{
    Console.Clear();
    Console.WriteLine("=== INICIO DE SESIÓN ===");
    Console.Write("Nombre de usuario: ");
    string nombre = Console.ReadLine();
    Console.Write("Contraseña: ");
    string contraseña = Console.ReadLine();

    var usuario = Usuarios.FirstOrDefault(
        u => u.NombreUsuario.Equals(nombre, StringComparison.OrdinalIgnoreCase)
             && u.Contraseña == contraseña);

    if (usuario == null)
    {
        Console.WriteLine("Credenciales incorrectas.");
        Console.WriteLine("Presiona una tecla para continuar.");
        Console.ReadKey();
        return;
    }

    if (usuario.EsAdministrador)
        MenuAdministrador(usuario);
    else
        MenuAlumno(usuario);
}
```

```
static void MenuAlumno(Usuario usuario)
{
    while (true)
    {
        Console.Clear();
        Console.WriteLine($"=== BIENVENIDO ({usuario.NombreUsuario}) ===");
        Console.WriteLine("1. Ver cursos disponibles");
        Console.WriteLine("2. Inscribirse a un curso");
        Console.WriteLine("3. Ver cursos inscritos");
        Console.WriteLine("4. Marcar curso como terminado");
        Console.WriteLine("5. Generar certificado");
        Console.WriteLine("6. Cerrar sesión");
        Console.Write("Opción: ");
        string opcion = Console.ReadLine();

        switch (opcion)
        {
            case "1":
                VerCursos();
                break;
            case "2":
                InscribirseCurso(usuario);
                break;
            case "3":
                VerCursosInscritos(usuario);
                break;
            case "4":
                MarcarCursoTerminado(usuario);
                break;
            case "5":
                GenerarCertificado(usuario);
                break;
            case "6":
                return;
            default:
                Console.WriteLine("Opción no válida.");
                Console.ReadKey();
                break;
        }
    }
}
```

```
4 referencias
static void VerCursos()
{
    Console.Clear();
    Console.WriteLine("=== CURSOS DISPONIBLES ===");
    foreach (var c in Cursos)
    {
        Console.WriteLine($"{c.Id}. {c.Nombre} (Instructor: {c.Instructor})");
    }
    Console.WriteLine("\nPresiona una tecla para continuar.");
    Console.ReadKey();
}

1 referencia
static void InscribirseCurso(Usuario usuario)
{
    VerCursos();
    Console.Write("Id del curso al que deseas inscribirte: ");
    if (!int.TryParse(Console.ReadLine(), out int idCurso))
    {
        Console.WriteLine("Id no válido.");
        Console.ReadKey();
        return;
    }

    var curso = Cursos.FirstOrDefault(c => c.Id == idCurso);
    if (curso == null)
    {
        Console.WriteLine("Curso no encontrado.");
        Console.ReadKey();
        return;
    }

    if (usuario.CursosInscritos.Contains(idCurso))
    {
        Console.WriteLine("Ya estás inscrito en ese curso.");
        Console.ReadKey();
        return;
    }

    usuario.CursosInscritos.Add(idCurso);
    Console.WriteLine("Inscripción realizada con éxito.");
    Console.ReadKey();
```

```
2 referencias
static void VerCursosInscritos(Usuario usuario)
{
    Console.Clear();
    Console.WriteLine("=== TUS CURSOS INSCRITOS ===");
    if (!usuario.CursosInscritos.Any())
    {
        Console.WriteLine("No estás inscrito en ningún curso.");
    }
    else
    {
        foreach (var id in usuario.CursosInscritos)
        {
            var c = Cursos.FirstOrDefault(x => x.Id == id);
            if (c != null)
            {
                bool completado = Progresos.Any(p => p.UsuarioId == usuario.Id && p.CursoId == c.Id && p.Completado);
                string estado = completado ? "Terminado" : "En progreso";
                Console.WriteLine($"{c.Id}. {c.Nombre} (Estado: {estado})");
            }
        }
    }
    Console.WriteLine("\nPresiona una tecla para continuar.");
    Console.ReadKey();
}
```

```
static void MarcarCursoTerminado(Usuario usuario)
{
    VerCursosInscritos(usuario);
    Console.Write("\nId del curso que deseas marcar como terminado: ");
    if (!int.TryParse(Console.ReadLine(), out int idCurso))
    {
        Console.WriteLine("Id no válido.");
        Console.ReadKey();
        return;
    }

    if (!usuario.CursosInscritos.Contains(idCurso))
    {
        Console.WriteLine("No estás inscrito en ese curso.");
        Console.ReadKey();
        return;
    }

    var progreso = Progresos.FirstOrDefault(p => p.UsuarioId == usuario.Id && p.CursoId == idCurso);
    if (progreso == null)
    {
        progreso = new ProgresoCurso
        {
            UsuarioId = usuario.Id,
            CursoId = idCurso,
            Completado = true,
            FechaFinalizacion = DateTime.Now
        };
        Progresos.Add(progreso);
    }
    else
    {
        progreso.Completado = true;
        progreso.FechaFinalizacion = DateTime.Now;
    }

    Console.WriteLine("Curso marcado como terminado.");
    Console.ReadKey();
}
```

```
static void GenerarCertificado(Usuario usuario)
{
    var completados = Progresos
        .Where(p => p.UsuarioId == usuario.Id && p.Completado)
        .ToList();

    if (!completados.Any())
    {
        Console.WriteLine("No tienes cursos terminados para generar certificado.");
        Console.ReadKey();
        return;
    }

    Console.Clear();
    Console.WriteLine("=== CURSOS TERMINADOS ===");
    foreach (var p in completados)
    {
        var c = Cursos.FirstOrDefault(x => x.Id == p.CursoId);
        if (c != null)
        {
            Console.WriteLine($"{c.Id}. {c.Nombre} (Finalizado: {p.FechaFinalizacion:dd/MM/yyyy})");
        }
    }

    Console.Write("\nIngresa el Id del curso para generar certificado: ");
    if (!int.TryParse(Console.ReadLine(), out int idCurso))
    {
        Console.WriteLine("Id no válido.");
        Console.ReadKey();
        return;
    }

    var prog = completados.FirstOrDefault(p => p.CursoId == idCurso);
    var curso = Cursos.FirstOrDefault(c => c.Id == idCurso);

    if (prog == null || curso == null)
    {
        Console.WriteLine("Curso no encontrado o no está terminado.");
        Console.ReadKey();
        return;
    }
```

```
    ICertificado certificado = new CertificadoBase(usuario, curso, prog.FechaFinalizacion ?? DateTime.Now);

    certificado = new CertificadoConFirma(certificado, curso.Instructor);
    certificado = new CertificadoConCodigo(certificado, $"CUR-{curso.Id}-USR-{usuario.Id}");

    Console.Clear();
    Console.WriteLine("Elige estilo de certificado:");
    Console.WriteLine("1. Estilo minimalista");
    Console.WriteLine("2. Estilo con marco ASCII");
    Console.Write("Opción: ");
    string opEstilo = Console.ReadLine();

    IEstiloCertificado estilo;
    if (opEstilo == "2")
        estilo = new EstiloAscii();
    else
        estilo = new EstiloMinimalista();

    var impresor = new ImprimirCertificado(certificado, estilo);
    impresor.Imprimir();

    Console.WriteLine("Certificado generado.");
    Console.ReadKey();
}
```

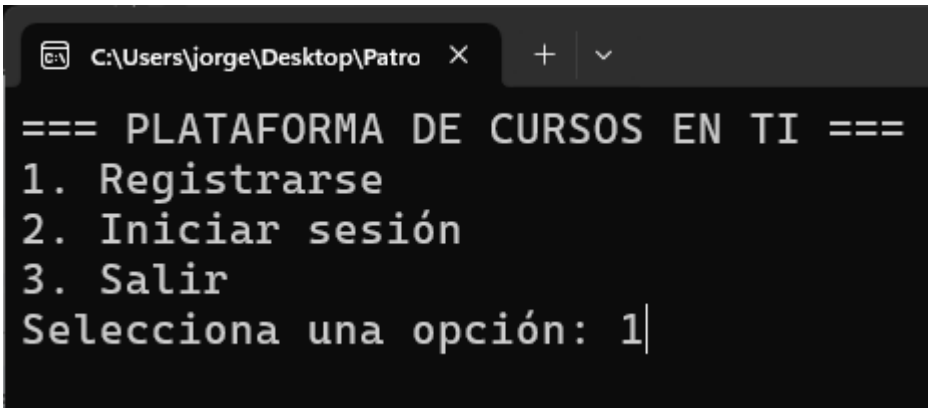
```
static void MenuAdministrador(Usuario admin)
{
    while (true)
    {
        Console.Clear();
        Console.WriteLine($"=== MENÚ ADMINISTRADOR ===");
        Console.WriteLine("1. Ver cursos");
        Console.WriteLine("2. Agregar curso");
        Console.WriteLine("3. Modificar curso");
        Console.WriteLine("4. Ver usuarios inscritos");
        Console.WriteLine("5. Cerrar sesión");
        Console.Write("Opción: ");
        string opcion = Console.ReadLine();

        switch (opcion)
        {
            case "1":
                VerCursos();
                break;
            case "2":
                AgregarCurso();
                break;
            case "3":
                ModificarCurso();
                break;
            case "4":
                VerUsuariosInscritos();
                break;
            case "5":
                return;
            default:
                Console.WriteLine("Opción no válida.");
                Console.ReadKey();
                break;
        }
    }
}
```

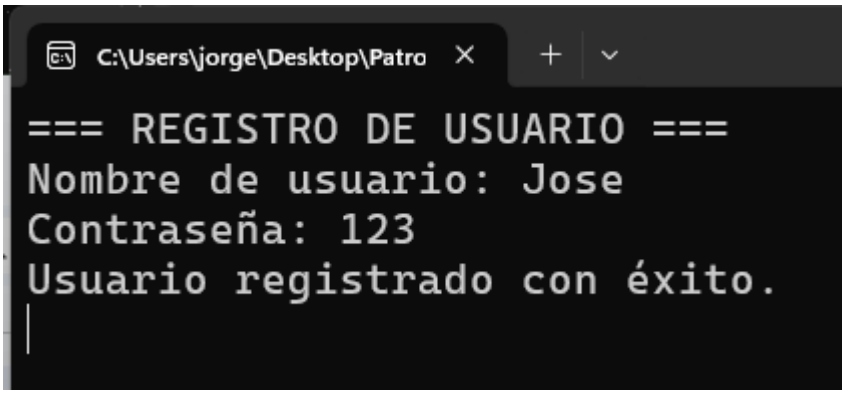
```
static void AgregarCurso()
{
    Console.Clear();
    Console.WriteLine("=== AGREGAR CURSO ===");
    Console.Write("Nombre del curso: ");
    string nombre = Console.ReadLine();

    if (string.IsNullOrWhiteSpace(nombre))
    {
        Console.WriteLine("El nombre no puede estar vacío. Regresando al menú...");
        Console.ReadKey();
        return;
    }

    Console.Write("Nombre del instructor: ");
    string instructor = Console.ReadLine();

    if (string.IsNullOrWhiteSpace(instructor))
    {
        Console.WriteLine("El instructor no puede estar vacío. Regresando al menú...");
        Console.ReadKey();
        return;
    }

    var curso = new Curso
    {
        Id = ++ultimoIdCurso,
        Nombre = nombre,
        Instructor = instructor
    };

    Cursos.Add(curso);
    Console.WriteLine("Curso agregado con éxito.");
    Console.ReadKey();
}
```

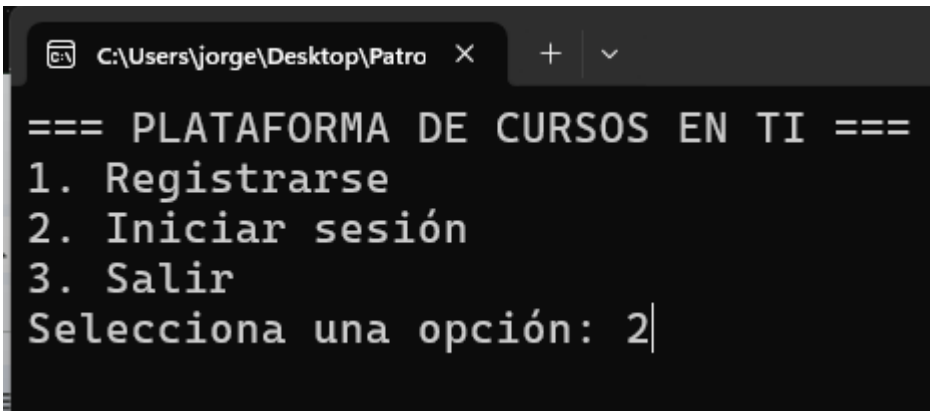
```
static void ModificarCurso()
{
    VerCursos();
    Console.Write("\nIngresa el Id del curso a modificar: ");
    if (!int.TryParse(Console.ReadLine(), out int idCurso))
    {
        Console.WriteLine("Id no válido.");
        Console.ReadKey();
        return;
    }

    var curso = Cursos.FirstOrDefault(c => c.Id == idCurso);
    if (curso == null)
    {
        Console.WriteLine("Curso no encontrado.");
        Console.ReadKey();
        return;
    }

    Console.WriteLine($"Curso actual: {curso.Nombre} (Instructor: {curso.Instructor})");
    Console.Write("Nuevo nombre (deja vacío para conservar): ");
    string nuevoNombre = Console.ReadLine();
    Console.Write("Nuevo instructor (deja vacío para conservar): ");
    string nuevoInstructor = Console.ReadLine();

    if (!string.IsNullOrWhiteSpace(nuevoNombre))
        curso.Nombre = nuevoNombre;
    if (!string.IsNullOrWhiteSpace(nuevoInstructor))
        curso.Instructor = nuevoInstructor;

    Console.WriteLine("Curso modificado.");
    Console.ReadKey();
}
```

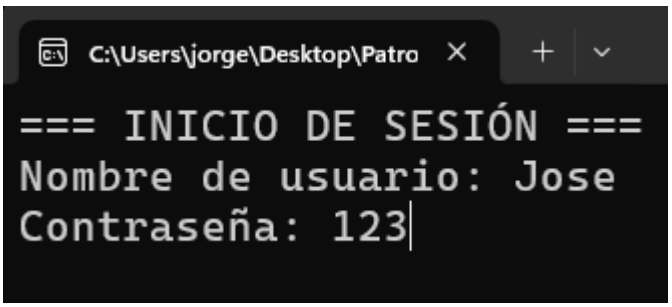
```
        1 referencia
        static void VerUsuariosInscritos()
        {
            Console.Clear();
            Console.WriteLine("=== USUARIOS E INSCRIPCIONES ===");
            foreach (var usuario in Usuarios)
            {
                if (usuario.EsAdministrador) continue;

                Console.WriteLine($"Usuario: {usuario.NombreUsuario}");
                if (!usuario.CursosInscritos.Any())
                {
                    Console.WriteLine("  Sin cursos inscritos.");
                }
                else
                {
                    foreach (var idCurso in usuario.CursosInscritos)
                    {
                        var curso = Cursos.FirstOrDefault(c => c.Id == idCurso);
                        if (curso != null)
                        {
                            bool completado = Progresos.Any(p => p.UsuarioId == usuario.Id && p.CursoId == curso.Id && p.Completado);
                            string estado = completado ? "Terminado" : "En progreso";
                            Console.WriteLine($"  - {curso.Nombre} ({estado})");
                        }
                    }
                }
                Console.WriteLine();
            }
            Console.WriteLine("Presiona una tecla para continuar.");
            Console.ReadKey();
        }
    }
}
```

Capturas de pantalla (Ejecución)

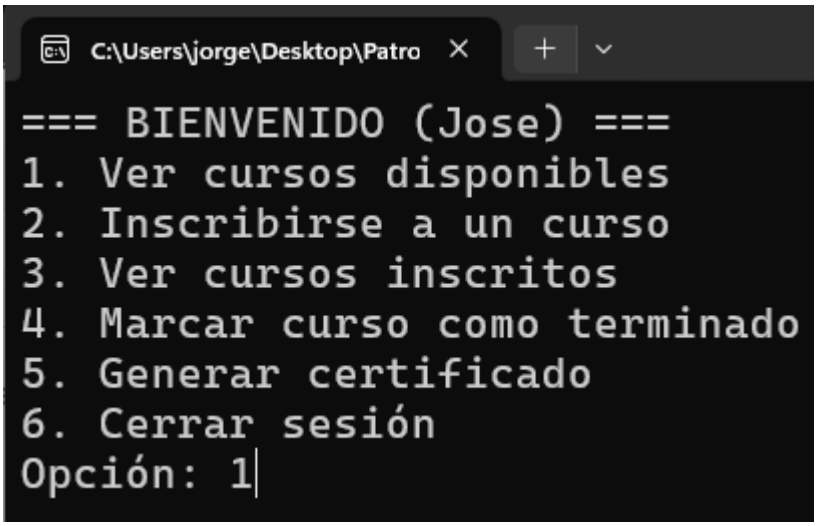




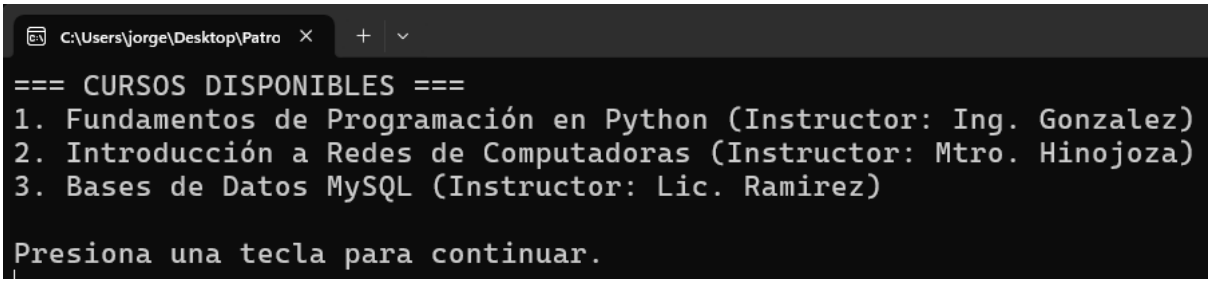




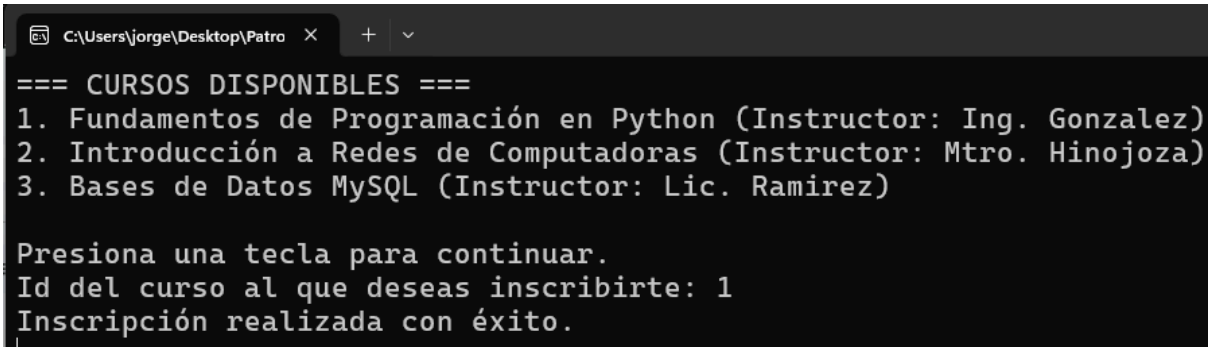
MENÚ USUARIO



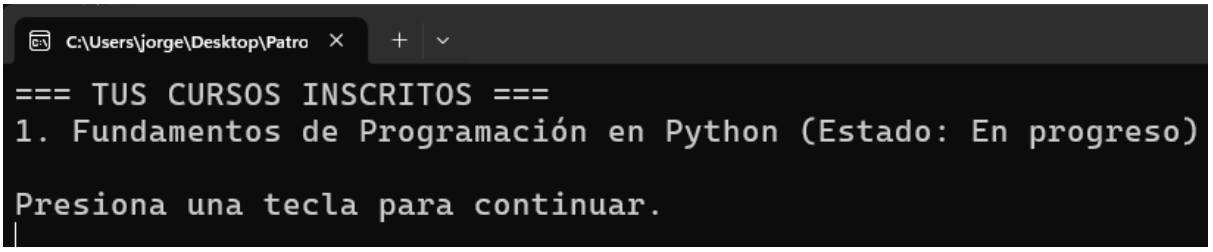
Opcion 1



Opcion 2



Opcion 3



Opcion 4

```
C:\Users\jorge\Desktop\Patro
=== TUS CURSOS INSCRITOS ===
1. Fundamentos de Programación en Python (Estado: En progreso)

Presiona una tecla para continuar.

Id del curso que deseas marcar como terminado: 1
Curso marcado como terminado.
```

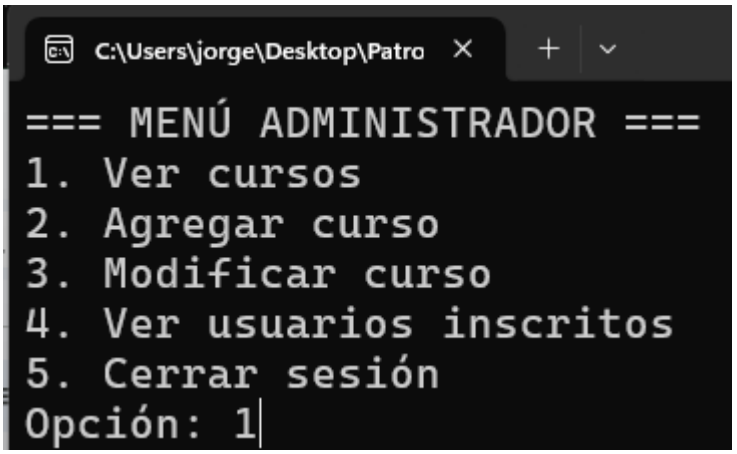
Opcion 5

```
C:\Users\jorge\Desktop\Patro
=== CURSOS TERMINADOS ===
1. Fundamentos de Programación en Python (Finalizado: 13/11/2025)

Ingresa el Id del curso para generar certificado: 1
```

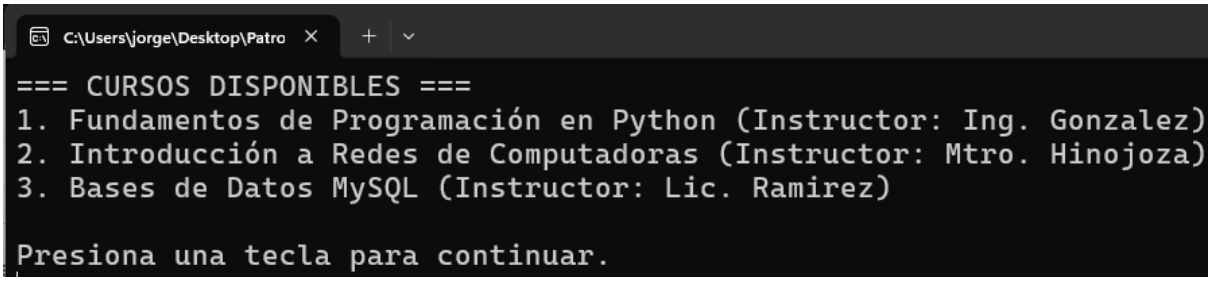
```
C:\Users\jorge\Desktop\Patro
Elige estilo de certificado:
1. Estilo minimalista
2. Estilo con marco ASCII
Opción: 1

================ CERTIFICADO =================

CERTIFICADO DE FINALIZACIÓN
Alumno: Jose
Curso: Fundamentos de Programación en Python
Área: Tecnologías de la Información
Fecha de finalización: 13/11/2025

Firma del instructor: Ing. Gonzalez
Código interno: CUR-1-USR-2

=============================================

Certificado generado.
```


C:\Users\jorge\Desktop\Patro
Elige estilo de certificado:
1. Estilo minimalista
2. Estilo con marco ASCII
Opción: 2
+--+
| CERTIFICADO TI |
+--+
CERTIFICADO DE FINALIZACIÓN
Alumno: Jose
Curso: Fundamentos de Programación en Python
Área: Tecnologías de la Información
Fecha de finalización: 13/11/2025
Firma del instructor: Ing. Gonzalez
Código interno: CUR-1-USR-2
+--+
| Plataforma de Cursos en TI |
+--+
Certificado generado.

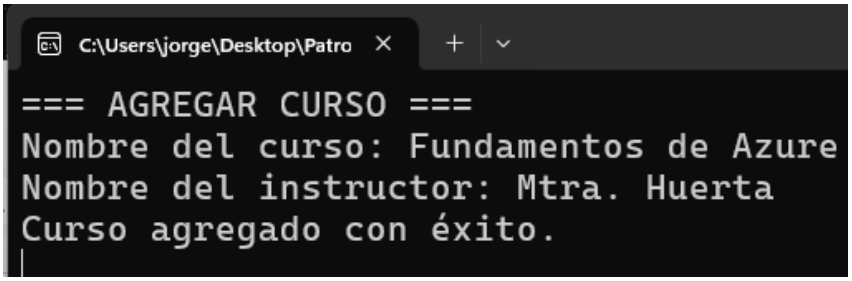
MENÚ ADMINISTRADOR



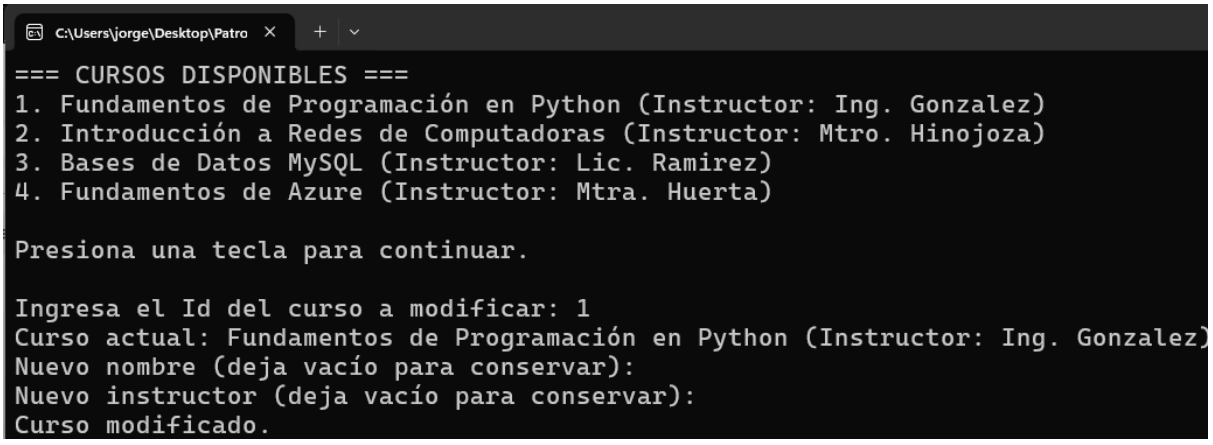
Opcion 1



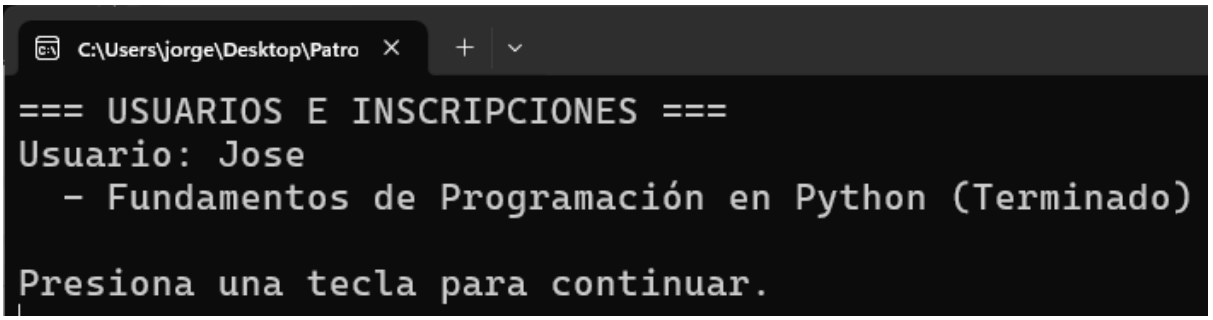
Opcion 2



Opcion 3



Opcion 4


C:\Users\jorge\Desktop\Patro
=== USUARIOS E INSCRIPCIONES ===
Usuario: Jose
- Fundamentos de Programación en Python (Terminado)
Presiona una tecla para continuar.

Conclusión

El patrón Decorator en conjunto con el patrón Bridge permite una buena colaboración o “Sinergia” para mejorar la presentación y la mejora continua de un programa, ya que permiten añadir grandes funcionalidades a cosas que ya estaban por defecto, agregando un plus sin alterar el código. Para este proyecto se implementó dentro de la generación de certificados, que al tener datos base como nombre de alumno, curso y fecha, con Decorator se agrega una firma y código interno que identifica el certificado como único, y para Bridge, se dividen dos estilos de certificado a conveniencia del usuario y sus requerimientos.