Actividades 1 y 2 C#

Aprendices:

Ana María Carmona Urrego

Mateo Usuga Álvarez

Valentina Correa Hoyos

Centro de Tecnología de la manufactura avanzada. SENA

3144585 análisis y desarrollo de software

Instructor:

Luis Fernando Sanchez

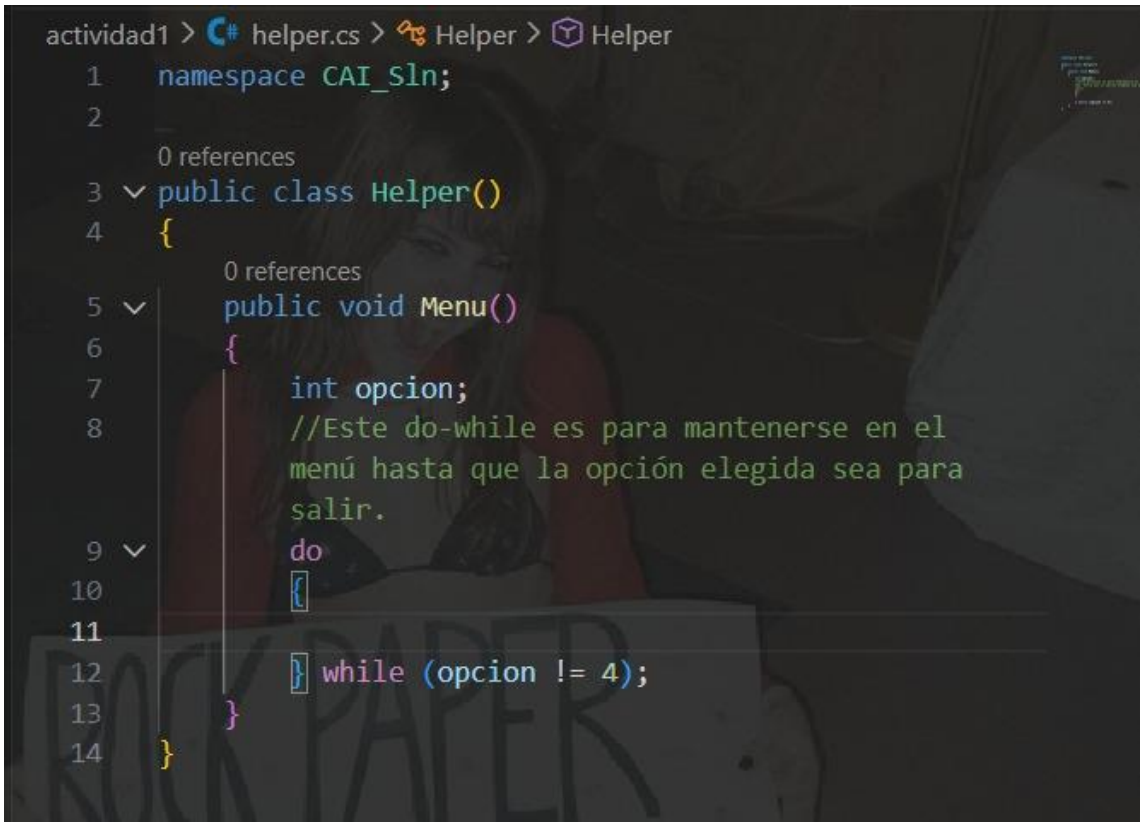
2025

Introducción

Este documento presenta dos aplicaciones desarrolladas en C#. La primera es una aplicación de consola con menús interactivos y submenús para operaciones básicas con números, textos y arreglos. La segunda es una aplicación web que gestiona matrículas de estudiantes usando ASP.NET Core, almacenando los datos en JSON.

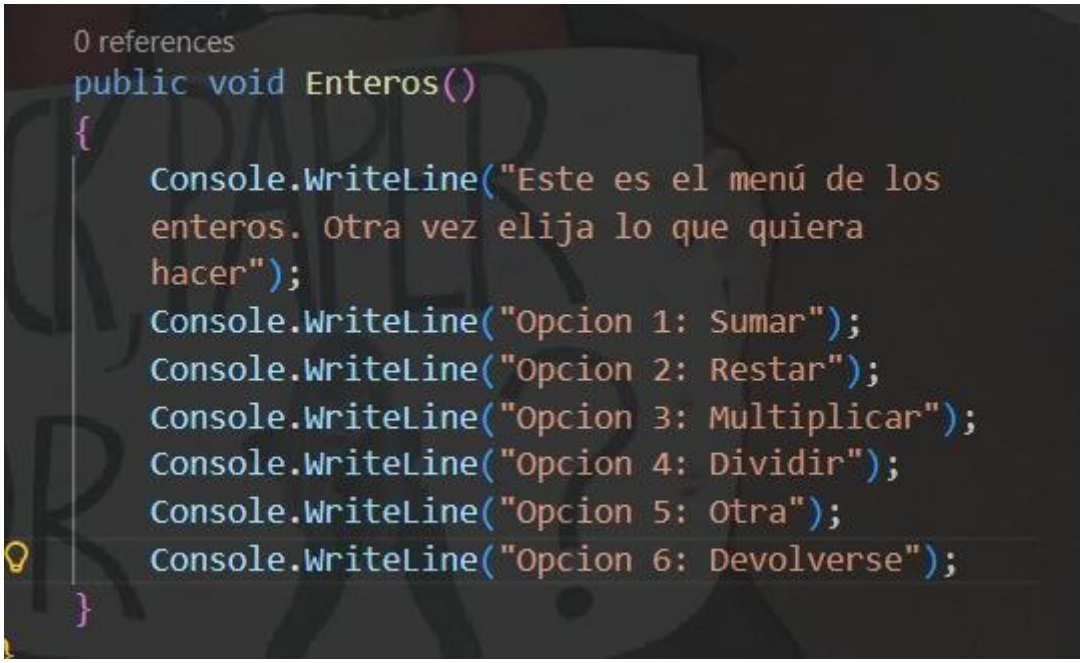
Ejercicio práctico 1:

Creación de una aplicación de consola con C#:

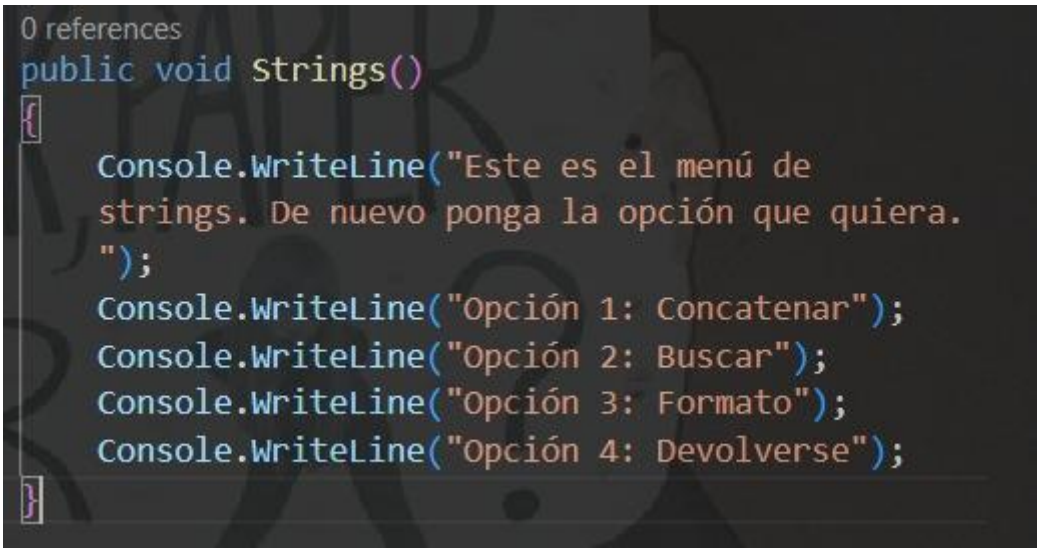


En esta parte del código, se desarrolla el helper, definiendo su respectivo namespace y la clase public que lo contiene. Después de eso, inicia la construcción del menú principal utilizando un ciclo do-while, el cual permite que el programa se

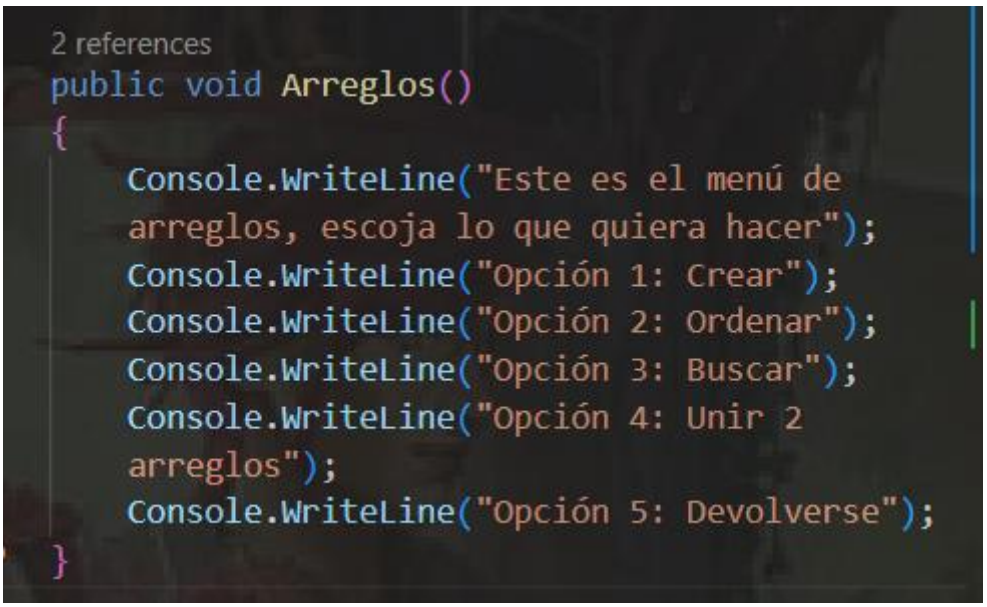
mantenga en ejecución de manera continua hasta que el usuario decida finalizar el proceso seleccionando la opción 4.



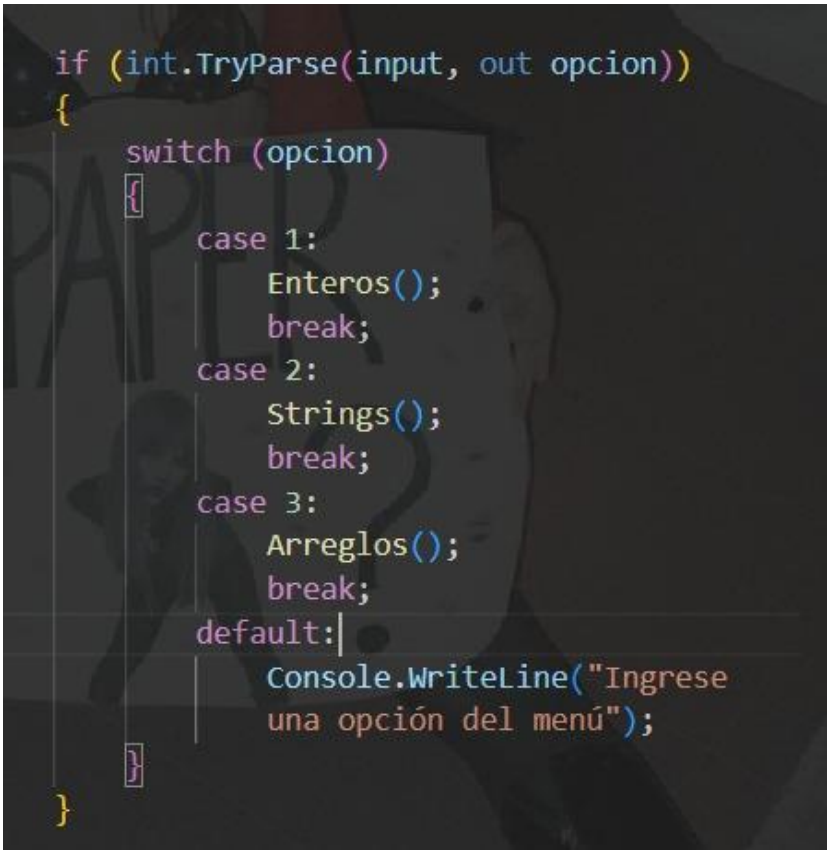
En esta parte se empieza a crear el submenú correspondiente con números enteros, en la pantalla se muestra en pantalla todas las opciones disponibles que el usuario puede seleccionar, como sumar, restar, multiplicar, dividir, realizar otra operación o regresar al menú anterior.



Aquí se crea el submenú de strings. Se muestra un mensaje introductorio y luego se imprimen en pantalla las opciones disponibles: concatenar, buscar, dar formato o devolverse al menú anterior. Este submenú sirve para que el usuario pueda escoger qué operación quiere hacer con cadenas de texto (strings).



Aquí se arma el submenú de arreglos. Se muestra un mensaje introductorio y luego aparecen las opciones que el usuario puede escoger, como crear un arreglo, ordenarlo, buscar un elemento, unir dos arreglos o devolverse al menú anterior.



```
if (int.TryParse(input, out opcion))
{
    switch (opcion)
    {
        case 1:
            Enteros();
            break;
        case 2:
            Strings();
            break;
        case 3:
            Arreglos();
            break;
        default:
            Console.WriteLine("Ingrese
            una opción del menú");
    }
}
```

Aquí se usa un `switch` para dirigir al usuario al submenú que corresponda. Según la opción que escriba, se llama al menú de enteros, al de strings o al de arreglos.

Ahora hay que hacer que los submenús funcionen. Para esto se utiliza un do-while para que en cada uno no se salga del submenú hasta que elija “devolverse”. Además, se utiliza un if para validar que el número ingresado sea un entero, al igual que en el menú principal; dentro de este if colocamos el switch con la opción elegida y el respectivo método que lo soluciona.

```
public void Enteros()

    //Este do-while de nuevo es pa que no se
    salga del submenú hasta que elija
    devolverse.
    do
    {
        Console.WriteLine("Este es el menú de
        los enteros. Otra vez elija lo que
        quiera hacer");
        Console.WriteLine("Opcion 1: Sumar");
        Console.WriteLine("Opcion 2: Restar");
        Console.WriteLine("Opcion 3:
        Multiplicar");
        Console.WriteLine("Opcion 4: Dividir");
        Console.WriteLine("Opcion 5: Otra");
        Console.WriteLine("Opcion 6:
        Devolverse");

        string input = Console.ReadLine();


        if (int.TryParse(inputEnteros, out
        opcionEnteros))
        {
            switch (opcionEnteros)
            {
                case 1:
                    Sumar();
                    break;
                case 2:
                    Restar();
                    break;
```

Método sumar:

```
void Sumar()
{
    Console.Write("Ingrese el primer
    número: ");
    int a = int.Parse(Console.ReadLine());

    Console.Write("Ingrese el segundo
    número: ");
    int b = int.Parse(Console.ReadLine());

    Console.WriteLine($"Resultado: {a + b}
    ");
}
```

Método restar:

```
void Restar()
{
    Console.Write("Ingrese el primer
    número: ");
    int a = int.Parse(Console.ReadLine());

    Console.Write("Ingrese el segundo
    número: ");
    int b = int.Parse(Console.ReadLine());

    Console.WriteLine($"Resultado: {a - b}
    ");
}
```

Método multiplicar:

```
void Multiplicar()
{
    Console.Write("Ingrese el primer
    número: ");
    int a = int.Parse(Console.ReadLine());

    Console.Write("Ingrese el segundo
    número: ");
    int b = int.Parse(Console.ReadLine());

    Console.WriteLine($"Resultado: {a * b}
    ");
}
```

Método dividir:

```
void Dividir()
{
    Console.Write("Ingrese el primer
    número (dividendo): ");
    int dividendo;

    Console.Write("Ingrese el segundo
    número (divisor): ");
    int divisor;

    if (divisor == 0)
    {
        Console.WriteLine("No se puede
        dividir entre cero.");
        return;
    } else
    {
        int resultado = dividendo /
        divisor;

        Console.WriteLine($"Resultado:
        {dividendo} / {divisor} =
        {resultado}");
    }
}
```

Para hacer funcionar el submenú de Strings hacemos también un do-while para que no se salga del submenú hasta que el usuario lo decida, un if para que si no es un entero lo que se coloca se reinicie el método del submenú, un switch para ver cúal es la opción escogida y sus respectivos métodos.

```
1 reference
public void Strings()
{
    do{
    Console.WriteLine("Este es el menú de
    strings. De nuevo ponga la opción que
    quiera.");
    Console.WriteLine("Opción 1: Concatenar");
    Console.WriteLine("Opción 2: Buscar");
    Console.WriteLine("Opción 3: Formato");
    Console.WriteLine("Opción 4: Devolverse");

    if (!int.TryParse(Console.ReadLine(), out
    opcion))
    {
        Console.WriteLine("Ingrese un número
        válido.");
        continue;
    }

    switch (opcion)
    {
        case 1:
            Concatenar();
            break;

        case 2:
            Buscar();
            break;

        case 3:
```

```
        case 3:
            Formato();
            break;

        case 4:
            Console.WriteLine("Volviendo al
            menú principal...");
            break;

        default:
            Console.WriteLine("Opción no
            válida.");
            break;
    }
```

Método concatenar:

```
void Concatenar()
{
    Console.Write("Ingrese el primer texto: ");
    string t1 = Console.ReadLine();

    Console.Write("Ingrese el segundo texto:
    ");
    string t2 = Console.ReadLine();

    string resultado = t1 + " " + t2;

    Console.WriteLine($"Resultado: {resultado}
    ");
}
```

Método buscar:

```
void Buscar()
{
    Console.Write("Ingrese un texto donde
    buscar: ");
    string texto = Console.ReadLine();

    Console.Write("Ingrese la palabra a
    buscar: ");
    string palabra = Console.ReadLine();

    if (texto.Contains(palabra))
        Console.WriteLine($"La palabra '
        {palabra}' SÍ se encontró en el texto.
        ");
    else
        Console.WriteLine($"La palabra '
        {palabra}' NO se encontró.");
```

Método Formato:

```
void Formato()
{
    Console.Write("Ingrese un texto: ");
    string texto = Console.ReadLine();

    Console.WriteLine("\nFormato aplicado:");
    Console.WriteLine($"En mayúsculas: {texto.
    ToUpper()}");
    Console.WriteLine($"En minúsculas: {texto.
    ToLower()}");
    Console.WriteLine($"Longitud del texto:
    {texto.Length} caracteres");
}
```

Para el menú de Arreglos utilizamos la misma estrategia anterior en los otros dos submenús para leer la opción, validarla, redirigir al usuario a los métodos elegidos y seguir en el submenú hasta que se elija la opción contraria.

```
public void Arreglos()
{
    int opcionArreglos = 0;
    do
    {
        Console.WriteLine("Este es el menú de
        arreglos, escoja lo que quiera hacer");
        Console.WriteLine("Opción 1: Crear");
        Console.WriteLine("Opción 2: Ordenar");
        Console.WriteLine("Opción 3: Buscar");
        Console.WriteLine("Opción 4: Unir 2
        arreglos");
        Console.WriteLine("Opción 5:
        Devolverse");

        if (!int.TryParse(Console.ReadLine(),
        out opcionArreglos))
        {
            Console.WriteLine("Ingrese un
            número válido.");
            continue;
        }

        switch (opcionArreglos)
        {
            case 1:
                CrearArreglo();
                break;

            case 2:
                OrdenarArreglo();
                break;
```

```
        case 3:
            BuscarEnArreglo();
            break;

        case 4:
            UnirArreglos();
            break;

        case 5:
            Console.WriteLine("Volviendo
            al menú principal...");
            break;

        default:
            Console.WriteLine("Opción no
            válida.");
            break;
    }
} while (opcionArreglos != 5);
```

Ahora haré dos métodos auxiliares para no tener que repetir mucho código en los próximos métodos:

Método auxiliar para leer enteros: se utiliza para leer de qué tamaño será el arreglo que el usuario quiere ingresar y leer los enteros que habrá en el arreglo.

```
int LeerEntero()
{
    int numero;
    while (!int.TryParse(Console.ReadLine
    (), out numero))
    {
        Console.Write("Entrada inválida.
        Ingrese un número entero: ");
    }
    return numero;
}
```

Método auxiliar para mostrar arreglos:

```
void MostrarArreglo(int[] arr)
{
    Console.WriteLine("[" + string.Join(",
    ", arr) + "]");
}
```

Ahora hay que crear un método para crear los arreglos, que también se podría considerar auxiliar debido a que se utiliza en todas las opciones del submenú:

```
int[] CrearArreglo()
{
    Console.Write("¿Cuántos elementos
    tendrá el arreglo?: ");
    int tam = LeerEntero();

    int[] arreglo = new int[tam];

    for (int i = 0; i < tam; i++)
    {
        Console.Write($"Elemento {i + 1}:
        ");
        arreglo[i] = LeerEntero();
        }

    Console.WriteLine("\nArreglo creado:");
    MostrarArreglo(arreglo);

    return arreglo;
}
```

Método para ordenar arreglos:

```
void OrdenarArreglo()
{
    int[] arr = CrearArreglo();

    Array.Sort(arr);

    Console.WriteLine("\nArreglo
    ordenado:");
    MostrarArreglo(arr);
}
```

Método para buscar en el arreglo:

```
void BuscarEnArreglo()
{
    int[] arr = CrearArreglo();

    Console.Write("¿Qué número desea
    buscar?: ");
    int valor = LeerEntero();

    bool encontrado = false;

    for (int i = 0; i < arr.Length; i++)
    {
        if (arr[i] == valor)
        {
            Console.WriteLine($"El valor
            {valor} está en la posición
            {i}.");
            encontrado = true;
        break;
        }
    }

    if (!encontrado)
    Console.WriteLine("El valor no se
    encontró en el arreglo.");
}
```

Método para unir 2 arreglos:

```
void UnirArreglos()
{
    Console.WriteLine("\nPrimero cree el
    arreglo 1:");
    int[] arr1 = CrearArreglo();

    Console.WriteLine("\nAhora cree el
    arreglo 2:");
    int[] arr2 = CrearArreglo();

    int[] resultado = new int[arr1.Length
    + arr2.Length];

    arr1.CopyTo(resultado, 0);
    arr2.CopyTo(resultado, arr1.Length);

    Console.WriteLine("\nArreglos
    unidos:");
    MostrarArreglo(resultado);
}
```

Ejercicio práctico 2:

Aplicación web con C#:

Controller:

```
using Microsoft.AspNetCore.Mvc;
using WebMatricula.Models;
using WebMatricula.Services;

namespace WebMatricula.Controllers
{
    public class MatriculaController : Controller
    {
        // Servicio para manejar operaciones con archivos JSON
        private readonly JsonFileService _jsonService;

        // Constructor con inyección de dependencia del servicio
        public MatriculaController(JsonFileService jsonService)
        {
            _jsonService = jsonService;
        }

        // Acción para mostrar la lista de todas las matrículas
        public IActionResult Index()
        {
            // Obtener todas las matrículas del servicio JSON
            var matriculas = _jsonService.LeerMatriculas();
            // Pasar la lista de matrículas a la vista
            return View(matriculas);
        }

        // Acción GET para mostrar el formulario de creación de matrícula
        public IActionResult Create()
        {
            // Retorna una vista vacía con el formulario
            return View();
```

```
            // Retorna una vista vacía con el formulario
            return View();
        }

        // Acción POST para procesar el formulario de creación
        [HttpPost]
        public IActionResult Create(Estudiante estudiante)
        {
            // Validar si los datos del modelo son válidos
            if (ModelState.IsValid)
            {
                // Obtener todas las matrículas existentes
                var matriculas = _jsonService.LeerMatriculas();

                // Generar ID automático:
                // Si hay matrículas existentes, toma el ID máximo y le suma 1
                // Si no hay matrículas, asigna ID = 1
                estudiante.Id = matriculas.Count > 0 ? matriculas.Max(e => e.Id) + 1 : 1;

                // Asignar fecha y hora actual del sistema
                estudiante.FeIngreso = DateTime.Now;

                // Generar número de matrícula único con formato: MAT-ID-FECHA
                estudiante.NumeroMatricula = $"MAT-{estudiante.Id}-{DateTime.Now:yyyyMMdd}";

                // Guardar la nueva matrícula en el archivo JSON
                _jsonService.AgregarMatricula(estudiante);

                // Mostrar mensaje de éxito (se usa TempData para persistir entre redirecciones)
                // Mostrar mensaje de éxito (se usa TempData para persistir entre redirecciones)
                TempData["Mensaje"] = "Matrícula registrada exitosamente!";

                // Redirigir al listado de matrículas
                return RedirectToAction("Index");
            }

            // Si el modelo no es válido, volver a mostrar el formulario con los errores
            return View(estudiante);
        }
    }
}
```

Este controller, se encarga del proceso de registrar nuevas matrículas de estudiantes.

Su función principal es recibir los datos del formulario, validarlos y generar automáticamente la información necesaria antes de guardarla.

Cuando un usuario accede al sistema, puede ver la lista completa de estudiantes matriculados. Al registrar una nueva matrícula, el controlador recibe los datos del formulario, los valida y automáticamente genera información adicional como un ID único, la fecha de ingreso y un número de matrícula especial. Finalmente, guarda toda esta información en un archivo JSON y confirma la operación exitosa al usuario.

Estudiante.cs:

```
using System;

namespace WebMatricula.Models
{
    // La clase Estudiante hereda de la clase Persona (hereda todas sus propiedades y métodos)
    public class Estudiante : Persona
    {
        // Propiedad específica del estudiante: curso al que está matriculado
        public string Curso { get; set; }

        // Número único de matrícula generado automáticamente
        public string NumeroMatricula { get; set; }

        // Constructor por defecto (sin parámetros) - Requerido para ASP.NET MVC
        public Estudiante() { }

        // Constructor completo con todos los parámetros para inicializar un estudiante
        public Estudiante(int id, string nombres, string apellidos, string direccion,
                          string email, int telefono, DateTime feNacimiento, DateTime feIngreso,
                          string curso = "")  // Parámetro opcional con valor por defecto
        {
            // Propiedades heredadas de la clase Persona
            Id = id;
            Nombres = nombres;
            Apellidos = apellidos;
            Direccion = direccion;
            Email = email;
            Telefono = telefono;
            FeNacimiento = feNacimiento;
            FeIngreso = feIngreso;

            // Propiedades específicas de Estudiante
            Curso = curso;

            // Genera automáticamente el número de matrícula con formato: MAT-ID-FECHA
            // Ejemplo: MAT-1-20231215
            NumeroMatricula = $"MAT-{Id}-{DateTime.Now:yyyyMMdd}";
        }

        // Método que retorna información formateada del estudiante
        public string GetInfoEstudiante()
        {
            return $"Estudiante: {Nombres} {Apellidos} - Matrícula: {NumeroMatricula} - Curso: {Curso}";
        }
    }
}
```

La clase Estudiante hereda todas las propiedades básicas de una clase Persona (como identificación, nombres, apellidos, dirección, contacto y fechas importantes) y añade propiedades específicas como el curso asignado y un número de matrícula que se genera automáticamente combinando un prefijo "MAT", el ID del estudiante y la fecha actual. Incluye dos constructores: uno vacío para que el framework ASP.NET MVC pueda instanciarla al procesar formularios web, y otro completo que permite inicializar

todos los datos del estudiante incluyendo la generación automática del número de matrícula, además de un método que devuelve la información del estudiante formateada en una cadena de texto legible.

Persona.cs:

```
using System;

namespace WebMatricula.Models
{
    // Clase base que representa a una persona en el sistema
    public class Persona
    {
        public int Id { get; set; }                    // Identificador único
        public string Nombres { get; set; }            // Nombres de la persona
        public string Apellidos { get; set; }          // Apellidos de la persona
        public string Direccion { get; set; }          // Dirección de residencia
        public string Email { get; set; }              // Correo electrónico
        public int Telefono { get; set; }              // Número de teléfono
        public DateTime FeNacimiento { get; set; }     // Fecha de nacimiento
        public DateTime FeIngreso { get; set; }        // Fecha de ingreso al sistema
    }
}
```

La clase Persona define la plantilla base para cualquier persona en el sistema, conteniendo las propiedades fundamentales que comparten todos los tipos de usuarios: identificación única (Id), nombres y apellidos, datos de contacto (dirección, email, teléfono) e información temporal (fecha de nacimiento y fecha de ingreso al sistema), sirviendo como clase padre para otras clases más específicas como Estudiante que heredarán todas estas características comunes.

JsonFileService.cs:

```
using System.Text.Json;
using WebMatricula.Models;

namespace WebMatricula.Services
{
    // Servicio para manejar operaciones con archivos JSON de estudiantes
    public class JsonFileService
    {
        private readonly string _filePath = "Data/matriculas.json";

        // Lee todas las matrículas del archivo JSON
        public List<Estudiante> LeerMatriculas()
        {
            // Si el archivo no existe, retorna lista vacía
            if (!File.Exists(_filePath))
                return new List<Estudiante>();

            var json = File.ReadAllText(_filePath);
            return JsonSerializer.Deserialize<List<Estudiante>>(json) ?? new List<Estudiante>();
        }

        // Guarda la lista completa de estudiantes en JSON
        public void GuardarMatriculas(List<Estudiante> matriculas)
        {
            // Crea el directorio si no existe
            var directory = Path.GetDirectoryName(_filePath);
            if (!Directory.Exists(directory))
                Directory.CreateDirectory(directory!);

            // Serializa con formato legible
            var options = new JsonSerializerOptions { WriteIndented = true };
            var json = JsonSerializer.Serialize(matriculas, options);
            File.WriteAllText(_filePath, json);
        }

        // Agrega un nuevo estudiante al archivo
        public void AgregarMatricula(Estudiante estudiante)
        {
            var matriculas = LeerMatriculas();
            matriculas.Add(estudiante);
            GuardarMatriculas(matriculas);
        }
    }
}
```

La clase JsonFileService es el servicio que funciona como "base de datos" del sistema, manejando el almacenamiento y lectura de todas las matrículas en un archivo JSON. Se encarga de tres operaciones principales: leer todas las matrículas existentes verificando si el archivo existe (devolviendo una lista vacía si no), guardar la lista completa de estudiantes en formato JSON legible creando automáticamente el directorio si no existe, y agregar nuevas matrículas leyendo primero la lista actual, añadiendo el

nuevo estudiante y guardando toda la lista actualizada en el archivo Data/matriculas.json.

Create.cshtml:

```
@model Estudiante  //!-- Modelo de datos: Clase Estudiante -->

@{
    ViewData["Title"] = "Nueva Matrícula";  <!-- Título de la página -->
}

<h2>Formulario de Matrícula</h2>

<!-- Formulario que envía datos al método Create del controlador -->
<form asp-action="Create" method="post">

    <!-- Campo: Nombres -->
    <div class="form-group">
        <label asp-for="Nombres"></label>
        <input asp-for="Nombres" class="form-control" required>
        <span asp-validation-for="Nombres" class="text-danger"></span>
    </div>

    <!-- Campo: Apellidos -->
    <div class="form-group">
        <label asp-for="Apellidos"></label>
        <input asp-for="Apellidos" class="form-control" required>
        <span asp-validation-for="Apellidos" class="text-danger"></span>
    </div>

    <!-- Campo: Dirección -->
    <div class="form-group">
        <label asp-for="Direccion"></label>
        <input asp-for="Direccion" class="form-control" required>
        <span asp-validation-for="Direccion" class="text-danger"></span>
    </div>
```

```
        <!-- Campo: Dirección -->
        <div class="form-group">
            <label asp-for="Direccion"></label>
            <input asp-for="Direccion" class="form-control" required>
            <span asp-validation-for="Direccion" class="text-danger"></span>
        </div>

        <!-- Campo: Email con validación de tipo email -->
        <div class="form-group">
            <label asp-for="Email"></label>
            <input asp-for="Email" type="email" class="form-control" required>
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>

        <!-- Campo: Teléfono con validación numérica -->
        <div class="form-group">
            <label asp-for="Telefono"></label>
            <input asp-for="Telefono" type="number" class="form-control" required>
            <span asp-validation-for="Telefono" class="text-danger"></span>
        </div>

        <!-- Campo: Fecha de nacimiento con selector de fecha -->
        <div class="form-group">
            <label asp-for="FeNacimiento"></label>
            <input asp-for="FeNacimiento" type="date" class="form-control" required>
            <span asp-validation-for="FeNacimiento" class="text-danger"></span>
        </div>

        <!-- Campo: Curso con lista desplegable -->
        <div class="form-group">
```

```
        <!-- Campo: Curso con lista desplegable -->
        <div class="form-group">
            <label asp-for="Curso"></label>
            <select asp-for="Curso" class="form-control" required>
                <option value="">Seleccione un curso</option>
                <option value="Matemáticas">Matemáticas</option>
                <option value="Ciencias">Ciencias</option>
                <option value="Literatura">Literatura</option>
                <option value="Historia">Historia</option>
                <option value="Inglés">Inglés</option>
            </select>
            <span asp-validation-for="Curso" class="text-danger"></span>
        </div>

        <!-- Botones de acción -->
        <button type="submit" class="btn btn-success">Guardar Matrícula</button>
        <a asp-action="Index" class="btn btn-secondary">Cancelar</a>
    </form>
```

Este archivo Create.cshtml es la vista que muestra el formulario web para registrar nuevos estudiantes en el sistema. Contiene campos para capturar toda la

información personal del estudiante: nombres, apellidos, dirección, email, teléfono y fecha de nacimiento, además de un selector desplegable para elegir el curso entre opciones predefinidas como Matemáticas, Ciencias, Literatura, Historia e Inglés. El formulario incluye validaciones automáticas para asegurar que los datos sean correctos (como validar que el email tenga formato adecuado y el teléfono sea numérico) y al enviarse, los datos se envían al método Create del controlador Matricula para procesar la matrícula.

Index.cshtml:

```
@model List<Estudiante>  //!-- Modelo: Lista de objetos Estudiante -->

@{
    ViewData["Title"] = "Lista de Matrículas";  <!-- Título de la página -->
}

<h2>Lista de Estudiantes Matriculados</h2>

<!-- Muestra mensaje temporal de éxito -->
@if (TempData["Mensaje"] != null)
{
    <div class="alert alert-success">@TempData["Mensaje"]</div>
}

<!-- Botón para crear nueva matrícula -->
<p>
    <a asp-action="Create" class="btn btn-primary">Nueva Matrícula</a>
</p>

<!-- Tabla para mostrar la lista de estudiantes -->
<table class="table">
    <thead>
        <tr>
            <th>ID</th>
            <th>Nombre Completo</th>
            <th>Email</th>
            <th>Teléfono</th>
            <th>Curso</th>
            <th>Matrícula</th>
        </tr>
    </thead>
```

```
32          <tbody>
33              <!-- Itera sobre cada estudiante en la lista -->
34              @foreach (var estudiante in Model)
35              {
36                  <tr>
37                      <td>@estudiante.Id</td>
38                      <td>@estudiante.Nombres @estudiante.Apellidos</td>
39                      <td>@estudiante.Email</td>
40                      <td>@estudiante.Telefono</td>
41                      <td>@estudiante.Curso</td>
42                      <td>@estudiante.NumeroMatricula</td>
43                  </tr>
44              }
45          </tbody>
46      </table>
```

Esta parte es la vista principal que muestra en una tabla ordenada todos los estudiantes matriculados en el sistema. La página presenta una lista con las columnas: ID, nombre completo, email, teléfono, curso asignado y número de matrícula de cada estudiante, generando automáticamente una fila por cada registro existente. Incluye un botón "Nueva Matrícula" para redirigir al formulario de registro y muestra mensajes de confirmación temporales (como "Matrícula registrada exitosamente") cuando se completa una operación, funcionando como el listado general donde los usuarios pueden ver toda la información de los estudiantes registrados

Program.cs:

```
using WebMatricula.Services;

var builder = WebApplication.CreateBuilder(args);

// Configuración de servicios DI
builder.Services.AddControllersWithViews();  // Soporte para MVC
builder.Services.AddSingleton<JsonFileService>();  // Servicio JSON como singleton

var app = builder.Build();

// Configuración del pipeline HTTP
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");  // Manejo de errores en producción
    app.UseHsts();  // HTTP Strict Transport Security
}

app.UseHttpsRedirection();    // Redirección HTTPS
app.UseStaticFiles();         // Archivos estáticos (CSS, JS, imágenes)
app.UseRouting();             // Sistema de rutas
app.UseAuthorization();       // Autorización (si se usa)

// Configuración de ruta por defecto
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Matricula}/{action=Index}/{id?}");  // Matricula/Index como home

app.Run();  // Inicia la aplicación
```

Esta parte es la configuración principal que inicia y prepara toda la aplicación web. Aquí se registran todos los servicios necesarios (como el manejo de vistas MVC y el servicio de datos JSON), se establecen las reglas de seguridad como la redirección automática a HTTPS y el manejo de errores, y se define que la página de inicio por defecto sea el listado de matrículas, funcionando como el "motor de arranque" que pone en marcha todo el sistema cuando un usuario lo visita.