

ARCHIVOS TEXT



Centro
de Ciencias
BÁSICAS

**Departamento de Ciencias
de la Computación**

Asignatura:

"Lenguajes de Computación"

Profesor:

Rosalinda Avendaño López

Alumnos:

- ▶ Luis Pablo Esparza Terrones
- ▶ Luis Manuel Flores García
- ▶ Adrián de Jesús Frausto Pérez
- ▶ Juan Francisco Gallo Ramírez

**Ingeniería en Computación
Inteligente**

2do Semestre

23 de Mayo de 2023

Índice

Contenido 2

Investigación del tema.	2
Conceptos elementales.	2
Definición de archivo.	2
Tipos de archivos.	3
Ventajas y desventajas de usar archivos text.	4
Clases de manipulación de archivos.	6
Creación, apertura y cierre.	9
Agregar datos.	11
Borrar datos.	12
Cambios de datos.	12
Consulta.	13
Instrucciones del programa ejemplo ABCC (text).	14
Listado del programa ejemplo ABCC (text).	15
Manual de usuario del programa ejemplo ABCC (text).	20
Instrucciones del programa ejercicio ABCC (text).	23
Listado del programa ejercicio ABCC (text).	24
Manual de usuario del programa ejercicio ABCC (text).	30
Ejemplo de programa con Windows Forms.	33
Preguntas del tema.	34

Conclusiones 36

Referencias 38

Contenido

Investigación del tema

1. Conceptos elementales

- ▶ **Archivos:** Un archivo es un contenedor de información.
- ▶ **Datos:** Los elementos individuales de los archivos se llaman datos o campos.
- ▶ **Carpetas:** Una carpeta es un contenedor de archivos, similar a la carpeta que puede haber en un armario archivador.
- ▶ **Objetos:** Como los archivos y las carpetas están representados en el Gestor de archivos en forma de iconos, el término objeto se utiliza para describir a ambos elementos. Los objetos son elementos diferenciados del escritorio que se pueden crear y manipular.
- ▶ **Rutas:** A menudo la ubicación de un archivo se especifica mediante una lista de las carpetas y subcarpetas que conducen al archivo; esta lista se denomina ruta de acceso.
- ▶ **Stream:** La lectura y escritura a un archivo son hechas usando un concepto genérico llamado stream.

2. Definición de archivo

Un **archivo** o fichero informático es una secuencia de bytes almacenados en un dispositivo. Un archivo es identificado por un nombre y la descripción de la carpeta o directorio que lo contiene. A los archivos informáticos se les llama así porque son los equivalentes digitales de los archivos escritos en expedientes, tarjetas, libretas, papel o microfichas del entorno de oficina tradicional.

Los archivos se utilizan cuando se desea almacenar datos de manera persistente, o para guardarlos en memoria secundaria con el fin de no utilizar

memoria primaria, dado que esta última es normalmente más escasa que la anterior. Dependiendo de cada sistema de archivos, los ficheros pueden tener atributos particulares como, por ejemplo, fecha de creación, fecha de última modificación, dueño y permisos de acceso.

El tamaño de un archivo está limitado por una serie de factores, como la capacidad disponible en la memoria secundaria del ordenador y los límites impuestos por el sistema operativo o el sistema de archivos.

El formato es la disposición de los datos dentro del archivo. El formato del archivo se conoce como **tipo de datos**.

3. Tipos de archivos

Podemos dividir los archivos en dos grandes grupos. Éstos son los ejecutables y los no ejecutables o archivos de datos. La diferencia fundamental entre ellos es que los primeros están creados para funcionar por sí mismos y los segundos almacenan información que tendrá que ser utilizada con ayuda de algún programa.

Dentro de los tipos de archivos de datos se pueden crear grupos, especialmente por la temática o clase de información que guarden. Se pueden separar los grupos en archivos de imágenes, de texto, de vídeo, comprimidos, etc.

Los tipos de archivo más comunes son:

- De **texto**: *txt, doc, docx, etc.*
- De **imagen**: *jpg, gif, bmp, png, etc.*
- De **vídeo**: *avi, mp4, mpeg, mwm, etc.*
- De **ejecución** o **del sistema**: *exe, bat, dll, sys, etc.*
- De **audio**: *mp3, wav, wma, etc.*
- De **archivo comprimido**: *zip, rar, tar, etc.*
- De **lectura**: *pdf, epub, azw, ibook, etc.*
- De **imagen de disco**: *iso, mds, img, etc.*

Existen diferentes tipos de **archivos de texto**, dependiendo de la extensión o la convención utilizada para identificar el tipo de contenido. Algunos tipos comunes de archivos de texto incluyen:

- **.txt:** *archivo de texto sin formato.*
- **.csv:** *archivo de valores separados por comas, utilizado para almacenar datos tabulares.*
- **.log:** *archivo de registro que registra eventos o mensajes de un programa o sistema.*
- **.html:** *archivo de texto que contiene código HTML utilizado para crear páginas web.*
- **.xml:** *archivo de texto estructurado que sigue las reglas del lenguaje de marcado XML.*

4. Ventajas y desventajas de usar archivos text

Un **archivo de texto** simple, texto sencillo o texto sin formato también llamado texto plano o texto simple, es un archivo informático que contiene únicamente texto formado solo por caracteres que son legibles por humanos y carece de cualquier tipo de formato tipográfico.

Estos archivos están compuestos de bytes que representan caracteres ordinarios como letras, números y signos de puntuación (incluyendo espacios en blanco), también incluye algunos pocos caracteres de control como tabulaciones, saltos de línea y retornos de carro.

Ventajas

El texto sin formato, a diferencia de otros formatos de datos, ofrece ventajas en términos de flexibilidad y portabilidad. Es compatible con casi todos los programas de aplicación en diferentes sistemas operativos y tipos de CPU, lo que permite manipular la información de manera manual o programática utilizando diversas herramientas de procesamiento de texto.

Esta flexibilidad y portabilidad hacen del texto sin formato el formato ideal para almacenar datos a largo plazo, ya que proporciona un seguro contra la obsolescencia de los programas de aplicación utilizados para crear, leer, modificar y expandir los datos. Los datos en formato legible por humanos tienen más probabilidades de sobrevivir en comparación con otras formas de datos y los programas que los crearon.

El texto sin formato facilita la lectura de archivos de datos en sistemas heredados o la conversión a otros formatos, incluso si se dispone de poca o ninguna información sobre el programa original utilizado para crearlos. En contraste, los formatos binarios propietarios pueden resultar difíciles o incluso imposibles de leer o utilizar.

Es importante tener en cuenta que el texto sin formato no implica necesariamente que no haya estructura. Los lenguajes de programación, SGML (lenguaje de marcado generalizado estándar) y sus descendientes como HTML (lenguaje de marcado de hipertexto) y XML (lenguaje de marcado extensible) son ejemplos de formatos de texto sin formato que tienen estructuras bien definidas. Estos formatos facilitan la legibilidad, reorganización y modificación de texto sin formato para las computadoras, al tiempo que siguen siendo legibles para los humanos.

Desventajas

El uso de texto sin formato para archivos de configuración y almacenamiento de datos varía mucho según el sistema operativo y el programa de aplicación.

Una desventaja que a veces se ha alegado para el texto sin formato es que puede consumir más espacio de almacenamiento (por ejemplo, disco duro o cinta magnética) que un formato binario comprimido. Otra es que podría ser computacionalmente más costoso (es decir, requerir más tiempo de CPU) para interpretar y procesar que los archivos binarios. Sin embargo, ambas supuestas desventajas han perdido importancia como resultado de la rápida reducción en el costo de almacenamiento y el aumento en las velocidades de procesamiento.

Los desarrolladores a veces han expresado su preocupación de que mantener los metadatos (es decir, datos sobre datos, incluida la información de formato) en forma

de texto sin formato podría exponerlos a daños accidentales o maliciosos por parte de los usuarios.

5. Clases de manipulación de archivos

En C#, el espacio de nombre **System.IO** contiene tipos que permiten leer y escribir en los archivos y secuencias de datos, así como tipos que proporcionan compatibilidad básica con los archivos y directorios.

Clases

BinaryReader	Lee tipos de datos primitivos como valores binarios en una codificación específica.
BinaryWriter	Escribe tipos primitivos en formato binario en una secuencia y admite la escritura de cadenas en una codificación específica.
BufferedStream	Agrega una capa de almacenamiento en búfer para las operaciones de lectura y escritura en otra secuencia. Esta clase no puede heredarse.
Directory	Expone métodos estáticos para crear, mover y enumerar archivos en directorios y subdirectorios. Esta clase no puede heredarse.
DirectoryInfo	Expone métodos de instancia para crear, mover y enumerar archivos en directorios y subdirectorios. Esta clase no puede heredarse.
DirectoryNotFoundException	Excepción que se produce cuando no se encuentra parte de un archivo o de un directorio.
DriveInfo	Proporciona acceso a información sobre una unidad.
DriveNotFoundException	Excepción que se produce al intentar obtener acceso a una unidad o un recurso compartido que no está disponible.
EndOfStreamException	Excepción que se produce cuando se intenta leer después del final de una secuencia.
EnumerationOptions	Proporciona opciones de enumeración de archivos y directorios.
BufferedStream	Agrega una capa de almacenamiento en búfer para las operaciones de lectura y escritura en otra secuencia. Esta clase no puede heredarse.
Directory	Expone métodos estáticos para crear, mover y enumerar archivos en directorios y subdirectorios. Esta clase no puede heredarse.
DirectoryInfo	Expone métodos de instancia para crear, mover y enumerar archivos en directorios y subdirectorios. Esta clase no puede heredarse.
DirectoryNotFoundException	Excepción que se produce cuando no se encuentra parte de un archivo o de un directorio.
DriveInfo	Proporciona acceso a información sobre una unidad.
DriveNotFoundException	Excepción que se produce al intentar obtener acceso a una unidad o un recurso compartido que no está disponible.
EndOfStreamException	Excepción que se produce cuando se intenta leer después del final de una secuencia.
EnumerationOptions	Proporciona opciones de enumeración de archivos y directorios.
ErrorEventArgs	Proporciona datos para el evento <u>Error</u> .

File	Proporciona métodos estáticos para crear, copiar, eliminar, mover y abrir un solo archivo, y contribuye a la creación de objetos FileStream .
FileInfo	Proporciona propiedades y métodos de instancia para crear, copiar, eliminar, mover y abrir archivos y contribuye a la creación de objetos FileStream . Esta clase no puede heredarse.
FileLoadException	Excepción que se produce cuando se encuentra un ensamblado administrado pero no se puede cargar.
DirectoryInfo	Expone métodos de instancia para crear, mover y enumerar archivos en directorios y subdirectorios. Esta clase no puede heredarse.
DirectoryNotFoundException	Excepción que se produce cuando no se encuentra parte de un archivo o de un directorio.
DriveInfo	Proporciona acceso a información sobre una unidad.
DriveNotFoundException	Excepción que se produce al intentar obtener acceso a una unidad o un recurso compartido que no está disponible.
EndOfStreamException	Excepción que se produce cuando se intenta leer después del final de una secuencia.
EnumerationOptions	Proporciona opciones de enumeración de archivos y directorios.
ErrorEventArgs	Proporciona datos para el evento Error .
File	Proporciona métodos estáticos para crear, copiar, eliminar, mover y abrir un solo archivo, y contribuye a la creación de objetos FileStream .
FileInfo	Proporciona propiedades y métodos de instancia para crear, copiar, eliminar, mover y abrir archivos y contribuye a la creación de objetos FileStream . Esta clase no puede heredarse.
FileLoadException	Excepción que se produce cuando se encuentra un ensamblado administrado pero no se puede cargar.
FileStream	Proporciona un Stream para un archivo, lo que permite operaciones de lectura y escritura sincrónica y asincrónica.
FileStreamOptions	Define una variedad de opciones de configuración para FileStream .
FileSystemAclExtensions	Proporciona métodos de extensión estáticos específicos de Windows para manipular atributos de seguridad de la lista de control de acceso (ACL) para los archivos y directorios.
FileSystemEventArgs	Proporciona datos para los eventos de directorio Changed , Created y Deleted .
FileSystemInfo	Proporciona la clase base para los objetos FileInfo y DirectoryInfo .
FileSystemWatcher	Escucha las notificaciones de cambio del sistema de archivos y genera eventos cuando cambia un directorio o un archivo de un directorio.
InternalBufferOverflowException	Excepción que se produce cuando el búfer interno se desborda.
InvalidDataException	Excepción que se produce cuando una secuencia de datos tiene un formato no válido.
IOException	Excepción que se produce cuando hay un error de E/S.
MemoryStream	Crea una secuencia cuya memoria auxiliar es la memoria.
Path	Ejecuta operaciones en instancias de String que contienen información de rutas de acceso de archivos o directorios. Estas operaciones se ejecutan de forma adecuada para múltiples plataformas.
PathTooLongException	Excepción que se produce cuando la longitud de una ruta de acceso o un nombre de archivo completo supera la longitud máxima definida por el sistema.
RandomAccess	Proporciona API basadas en desplazamiento para leer y escribir archivos de forma segura para subprocesos.

RenamedEventArgs	Proporciona datos para el evento Renamed .
Stream	Proporciona una vista genérica de una secuencia de bytes. Esta es una clase abstracta.
StreamReader	Implementa un TextReader que lee los caracteres de una secuencia de bytes en una codificación determinada.
StreamWriter	Implementa TextWriter para escribir los caracteres de una secuencia en una codificación determinada.
StringReader	Implementa TextReader que lee en una cadena.
StringWriter	Implementa un TextWriter para escribir información en una cadena. La información se almacena en un StringBuilder subyacente.
TextReader	Representa un lector que puede leer una serie secuencial de caracteres.
TextWriter	Representa un sistema de escritura que puede escribir una serie secuencial de caracteres. Esta clase es abstracta.
UnmanagedMemoryAccessor	Proporciona acceso aleatorio a bloques de memoria no administrada desde código administrado.
UnmanagedMemoryStream	Proporciona acceso a los bloques de memoria no administrada desde el código administrado.
FileStream	Proporciona un Stream para un archivo, lo que permite operaciones de lectura y escritura sincrónica y asincrónica.
FileStreamOptions	Define una variedad de opciones de configuración para FileStream .
FileSystemAclExtensions	Proporciona métodos de extensión estáticos específicos de Windows para manipular atributos de seguridad de la lista de control de acceso (ACL) para los archivos y directorios.
FileSystemEventArgs	Proporciona datos para los eventos de directorio Changed , Created y Deleted .
FileSystemInfo	Proporciona la clase base para los objetos FileInfo y DirectoryInfo .
FileSystemWatcher	Escucha las notificaciones de cambio del sistema de archivos y genera eventos cuando cambia un directorio o un archivo de un directorio.
InternalBufferOverflowException	Excepción que se produce cuando el búfer interno se desborda.
InvalidDataException	Excepción que se produce cuando una secuencia de datos tiene un formato no válido.
IOException	Excepción que se produce cuando hay un error de E/S.
MemoryStream	Crea una secuencia cuya memoria auxiliar es la memoria.
Path	Ejecuta operaciones en instancias de String que contienen información de rutas de acceso de archivos o directorios. Estas operaciones se ejecutan de forma adecuada para múltiples plataformas.
PathTooLongException	Excepción que se produce cuando la longitud de una ruta de acceso o un nombre de archivo completo supera la longitud máxima definida por el sistema.
RandomAccess	Proporciona API basadas en desplazamiento para leer y escribir archivos de forma segura para subprocessos.
RenamedEventArgs	Proporciona datos para el evento Renamed .
Stream	Proporciona una vista genérica de una secuencia de bytes. Esta es una clase abstracta.
StreamReader	Implementa un TextReader que lee los caracteres de una secuencia de bytes en una codificación determinada.

6. Creación, apertura y cierre

Creación de un archivo

El objetivo de esta operación es permitir a los usuarios la creación de nuevos archivos. Mediante esta operación se indican las propiedades y las características del archivo para que el sistema de archivos pueda reconocerlo y procesarlo. En el proceso de creación del archivo debe registrarse la información necesaria para que el sistema pueda localizar el archivo y manipular sus registros lógicos. Para ello, el método de acceso debe obtener información sobre el formato y el tamaño de los registros lógicos y físicos, la identificación del archivo, la fecha de creación, su posible tamaño, su organización, aspectos de seguridad, etc.

Apertura de un archivo

En esta operación el método de acceso localiza e identifica un archivo existente para que los usuarios o el propio sistema operativo pueda operar con él. En algunos sistemas la operación de creación no existe como tal, y es la operación de archivo de un fichero no existente, la que implícitamente, crea un nuevo archivo. Los errores que pueden producirse en la apertura de un archivo son los siguientes:

- El archivo no se encuentra en el lugar indicado (dispositivo, directorio, nombre).
- El archivo se ha localizado pero el usuario no tiene permiso para acceder al mismo.
- El archivo no se puede leer por errores en el hardware del dispositivo de almacenamiento.

Cierre de un archivo

Esta operación se utiliza para indicar que se va a dejar de utilizar un archivo determinado. Mediante el método de acceso se encarga de "romper" la conexión entre el programa de usuario y el archivo, garantizando la integridad de los registros. Al ejecutar esta operación, el sistema se encarga de escribir en el dispositivo de almacenamiento aquella información que contienen los búfer asociados al archivo y se llevan a cabo las operaciones de limpieza necesarias. Tras cerrar el archivo, sus atributos dejan de ser accesibles para el método de acceso. El único parámetro

necesario para realizar esta operación es el identificador del archivo devuelto por el método de acceso al crear o abrir el archivo. Los errores que se pueden producir al cerrar un archivo son los siguientes:

- El archivo no está abierto.
- No se ha podido escribir en el dispositivo toda la información del archivo, debido a fallos en el hardware.
- No se ha podido escribir en el dispositivo toda la información del archivo por falta de espacio en el dispositivo de almacenamiento.

En el caso de los archivos text en C#, para crear un archivo:

```
// Si no se especifica la dirección y solo se pone el nombre del archivo éste
se guardará en la carpeta del proyecto.
string filePath = "ruta/del/archivo.txt";

StreamWriter writer = null;
try
{
    // Crea un nuevo archivo y abre un StreamWriter para escribir en él
    writer = new StreamWriter(filePath);

    // Escribe el contenido en el archivo
    writer.WriteLine("Hola, mundo!");
    writer.WriteLine("Este es un ejemplo de archivo de texto en C#.");
}
catch (IOException ex)
{
    Console.WriteLine("Error al crear el archivo: " + ex.Message);
}
finally
{
    // Cierra el StreamWriter en el bloque finally para asegurarse de que se
    cierre incluso si ocurre una excepción
    if (writer != null)
    {
        writer.Close();
    }
}
```

Para abrir un archivo:

```
string filePath = "ruta/del/archivo.txt";

StreamReader reader = null;
try
{
```

```
// Abre un StreamReader para leer el archivo
reader = new StreamReader(filePath);

// Lee y muestra el contenido del archivo línea por línea
string line;
while ((line = reader.ReadLine()) != null)
{
    Console.WriteLine(line);
}
}
catch (IOException ex)
{
    Console.WriteLine("Error al abrir el archivo: " + ex.Message);
}
finally
{
    if (reader != null)
    {
        reader.Close();
    }
}
```

Para cerrar un archivo:

En ambos casos, el bloque `finally` se utiliza para asegurarse de que el `StreamWriter` o el `StreamReader` se cierren correctamente, incluso si ocurre una excepción durante la operación.

7. Agregar datos

Para agregar datos a un archivo de texto existente, se abre el archivo en modo "agregar" en lugar de sobrescribirlo por completo. Esto permite que los nuevos datos se anexas al final del archivo sin afectar el contenido existente.

Ejemplo en C# con archivos text:

```
string filePath = "ruta/del/archivo.txt";

// Abre el archivo en modo "agregar" utilizando StreamWriter
using (StreamWriter writer = File.AppendText(filePath))
{
    // Escribe los nuevos datos en el archivo
    writer.WriteLine("Nuevo dato 1");
    writer.WriteLine("Nuevo dato 2");
    // Puedes agregar más líneas de datos según tus necesidades
}
```

8. Borrar datos

Para borrar datos de un archivo de texto, se puede abrir el archivo en modo de escritura y sobrescribir el contenido con los datos actualizados o simplemente eliminar el archivo por completo.

Ejemplo en C# con archivos text:

```
string filePath = "ruta/del/archivo.txt";

// Lee todos los datos del archivo
string[] lines = File.ReadAllLines(filePath);

// Borra el archivo existente
File.Delete(filePath);

// Crea un nuevo archivo y escribe los datos sin las líneas que deseas borrar
using (StreamWriter writer = new StreamWriter(filePath))
{
    foreach (string line in lines)
    {
        if (!line.Contains("dato a borrar"))
        {
            writer.WriteLine(line);
        }
    }
}
```

9. Cambios de datos

Los cambios en un archivo de texto se realizan mediante la apertura del archivo en modo de escritura y la modificación de los datos según sea necesario. Esto puede implicar la edición de líneas específicas, la eliminación o inserción de contenido, o cualquier otro tipo de manipulación requerida.

Ejemplo en C# con archivos text:

```
string filePath = "ruta/del/archivo.txt";

// Lee todos los datos del archivo
string[] lines = File.ReadAllLines(filePath);

// Modifica los datos necesarios
for (int i = 0; i < lines.Length; i++)
```

```

{
    if (lines[i].Contains("dato a cambiar"))
    {
        lines[i] = "Nuevo dato";
    }
}

// Sobrescribe el archivo con los datos modificados
File.WriteAllLines(filePath, lines);

```

10. Consulta

La consulta de datos en un archivo de texto implica la lectura de los caracteres o líneas relevantes del archivo para obtener la información deseada. Esto puede implicar la búsqueda de patrones específicos, la extracción de datos en función de ciertos criterios o cualquier otra operación de búsqueda y recuperación necesaria.

Ejemplo en C# con archivos text:

```

string filePath = "ruta/del/archivo.txt";

// Lee todos los datos del archivo
string[] lines = File.ReadAllLines(filePath);

// Realiza consultas en los datos
foreach (string line in lines)
{
    if (line.Contains("dato buscado"))
    {
        // Realiza alguna acción con el dato encontrado
        Console.WriteLine("Dato encontrado: " + line);
    }
}

```

Instrucciones del programa ejemplo ABCC

Utilizando una aplicación de consola con C#, se pide realizar un sistema de registro de vehículos dónde se ingrese la marca, modelo y placa. Se pide realizar un menú con las opciones de:

- Agregar vehículo.
- Borrar vehículo.
- Cambiar datos del vehículo.
- Consultar registros.

Estos datos se guardarán en un archivo de texto con las clases y métodos anteriormente vistos.

Listado del programa ejemplo ABCC

```
using System;
using System.IO;

class Program
{
    static string filePath = "vehiculos.txt"; // Ruta del archivo de texto donde se guardarán los datos

    static void Main()
    {
        Console.Title = "Sistema de Registro de Vehículos";

        // Colores de título
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine("=====");
        Console.WriteLine("        SISTEMA DE REGISTRO DE VEHÍCULOS        ");
        Console.WriteLine("=====");
        Console.ResetColor();

        while (true)
        {
            MostrarMenu(); // Mostrar el menú principal

            Console.WriteLine("\nSeleccione una opción:");
            string? opcion = Console.ReadLine();

            switch (opcion)
            {
                case "1":
                    LimpiarPantalla();
                    AgregarDatos(); // Opción para agregar datos
                    break;
                case "2":
                    LimpiarPantalla();
                    BorrarDatos(); // Opción para borrar datos
                    break;
                case "3":
                    LimpiarPantalla();
                    CambiarDatos(); // Opción para cambiar datos
                    break;
                case "4":
                    LimpiarPantalla();
                    ConsultarDatos(); // Opción para consultar datos
                    break;
                case "5":
                    return;
                default:
                    LimpiarPantalla();
                    Console.WriteLine("Opción no válida");
                    break;
            }

            Console.WriteLine();
            Console.WriteLine("Presione cualquier tecla para volver al menú principal...");
            Console.ReadKey();
            LimpiarPantalla();
        }
    }

    // Método para mostrar el menú principal
    static void MostrarMenu()
    {
        // Colores del menú principal
        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine("=====");
    }
}
```



```

        Console.WriteLine("                MENÚ PRINCIPAL                ");
        Console.WriteLine("=====");
        Console.ResetColor();

        // Opciones del menú principal
        Console.WriteLine("    1. Agregar datos");
        Console.WriteLine("    2. Borrar datos");
        Console.WriteLine("    3. Cambiar datos");
        Console.WriteLine("    4. Consultar datos");
        Console.WriteLine("    5. Salir");
    }

    // Método para limpiar la pantalla y mostrar el título
    static void LimpiarPantalla()
    {
        Console.Clear();
        Console.Title = "Sistema de Registro de Vehículos";

        // Colores del título
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine("=====");
        Console.WriteLine("                SISTEMA DE REGISTRO DE VEHÍCULOS                ");
        Console.WriteLine("=====");
        Console.ResetColor();
    }

    // Método para agregar datos de un vehículo
    static void AgregarDatos()
    {
        Console.WriteLine("Ingrese los datos del vehículo:");
        Console.WriteLine("Marca:");
        string? marca = Console.ReadLine();

        Console.WriteLine("Modelo:");
        string? modelo = Console.ReadLine();

        Console.WriteLine("Placa:");
        string? placa = Console.ReadLine();

        // Validación de datos de entrada
        if (string.IsNullOrEmpty(marca) || string.IsNullOrEmpty(modelo) ||
            string.IsNullOrEmpty(placa))
        {
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("Error: Debe ingresar todos los datos del vehículo.");
            Console.ResetColor();
            return;
        }

        try
        {
            using (StreamWriter writer = new StreamWriter(filePath, true))
            {
                writer.WriteLine($"Marca: {marca}, Modelo: {modelo}, Placa: {placa}");
            }

            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Datos agregados correctamente.");
            Console.ResetColor();
        }
        catch (IOException ex)
        {
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("Error al agregar datos: " + ex.Message);
            Console.ResetColor();
        }
    }

```

```

}

// Método para borrar datos de un vehículo
static void BorrarDatos()
{
    Console.WriteLine("Ingrese la placa del vehículo a borrar:");
    string? placa = Console.ReadLine();

    // Validación de la placa
    if (string.IsNullOrEmpty(placa))
    {
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error: Debe ingresar la placa del vehículo a borrar.");
        Console.ResetColor();
        return;
    }

    string[]? lines = null;
    try
    {
        lines = File.ReadAllLines(filePath);

        bool datosBorrados = false;
        using (StreamWriter writer = new StreamWriter(filePath))
        {
            foreach (string line in lines)
            {
                if (!line.Contains($"Placa: {placa}"))
                {
                    writer.WriteLine(line);
                }
                else
                {
                    datosBorrados = true;
                }
            }
        }

        if (datosBorrados)
        {
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Datos borrados correctamente.");
            Console.ResetColor();
        }
        else
        {
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine("No se encontraron datos con la placa especificada.");
            Console.ResetColor();
        }
    }
    catch (IOException ex)
    {
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error al borrar datos: " + ex.Message);
        Console.ResetColor();
    }
    finally
    {
        if (lines != null)
        {
            Array.Clear(lines, 0, lines.Length);
        }
    }
}

```

```

// Método para cambiar datos de un vehículo
static void CambiarDatos()
{
    Console.WriteLine("Ingrese la placa del vehículo a cambiar:");
    string? placa = Console.ReadLine();

    Console.WriteLine("Ingrese los nuevos datos del vehículo:");
    Console.WriteLine("Marca:");
    string? marca = Console.ReadLine();

    Console.WriteLine("Modelo:");
    string? modelo = Console.ReadLine();

    Console.WriteLine("Placa:");
    string? nuevaPlaca = Console.ReadLine();

    // Validación de los datos de entrada
    if (string.IsNullOrEmpty(placa) || (string.IsNullOrEmpty(marca) &&
string.IsNullOrEmpty(modelo) && string.IsNullOrEmpty(nuevaPlaca)))
    {
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error: Debe ingresar la placa del vehículo y al menos un dato
nuevo.");
        Console.ResetColor();
        return;
    }

    string[]? lines = null;
    try
    {
        lines = File.ReadAllLines(filePath);

        bool datosCambiados = false;
        using (StreamWriter writer = new StreamWriter(filePath))
        {
            foreach (string line in lines)
            {
                if (line.Contains($"Placa: {placa}"))
                {
                    string newLine = line;
                    if (!string.IsNullOrEmpty(marca))
                    {
                        newLine = newLine.Replace("Marca:", $"Marca: {marca}");
                    }
                    if (!string.IsNullOrEmpty(modelo))
                    {
                        newLine = newLine.Replace("Modelo:", $"Modelo: {modelo}");
                    }
                    if (!string.IsNullOrEmpty(nuevaPlaca))
                    {
                        newLine = newLine.Replace($"Placa: {placa}", $"Placa:
{nuevaPlaca}");
                    }

                    writer.WriteLine(newLine);
                    datosCambiados = true;
                }
                else
                {
                    writer.WriteLine(line);
                }
            }
        }

        if (datosCambiados)
        {
            Console.WriteLine();

```

```

        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("Datos cambiados correctamente.");
        Console.ResetColor();
    }
    else
    {
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine("No se encontraron datos con la placa especificada.");
        Console.ResetColor();
    }
}
catch (IOException ex)
{
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine("Error al cambiar datos: " + ex.Message);
    Console.ResetColor();
}
finally
{
    if (lines != null)
    {
        Array.Clear(lines, 0, lines.Length);
    }
}
}

// Método para consultar todos los datos de los vehículos
static void ConsultarDatos()
{
    try
    {
        string[] lines = File.ReadAllLines(filePath);

        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine("=====");
        Console.WriteLine("          DATOS DE LOS VEHÍCULOS          ");
        Console.WriteLine("=====");
        Console.ResetColor();

        Console.WriteLine();
        foreach (string line in lines)
        {
            Console.WriteLine(line);
        }
    }
    catch (IOException ex)
    {
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error al consultar datos: " + ex.Message);
        Console.ResetColor();
    }
}
}

```

Manual de usuario del programa ejemplo ABCC

Introducción

El Sistema de Registro de Vehículos es una aplicación de consola que te permite registrar información básica de vehículos, como marca, modelo y placa. A continuación, se describen las diferentes funciones disponibles y cómo utilizarlas.

Menú Principal

Al iniciar la aplicación, se mostrará un menú con las siguientes opciones:

1. *Agregar Vehículo:*
Permite ingresar los datos de un nuevo vehículo y almacenarlos en el archivo de texto.
2. *Borrar Vehículo:*
Permite eliminar un vehículo existente del archivo de texto.
3. *Cambiar Datos de Vehículo:*
Permite modificar los datos de un vehículo existente en el archivo de texto.
4. *Consultar Vehículo:*
Permite buscar y mostrar información de un vehículo específico.
5. *Salir.*



```
Sistema de Registro de Vehículos x + -
=====
SISTEMA DE REGISTRO DE VEHÍCULOS
=====
MENÚ PRINCIPAL
=====
1. Agregar datos
2. Borrar datos
3. Cambiar datos
4. Consultar datos
5. Salir

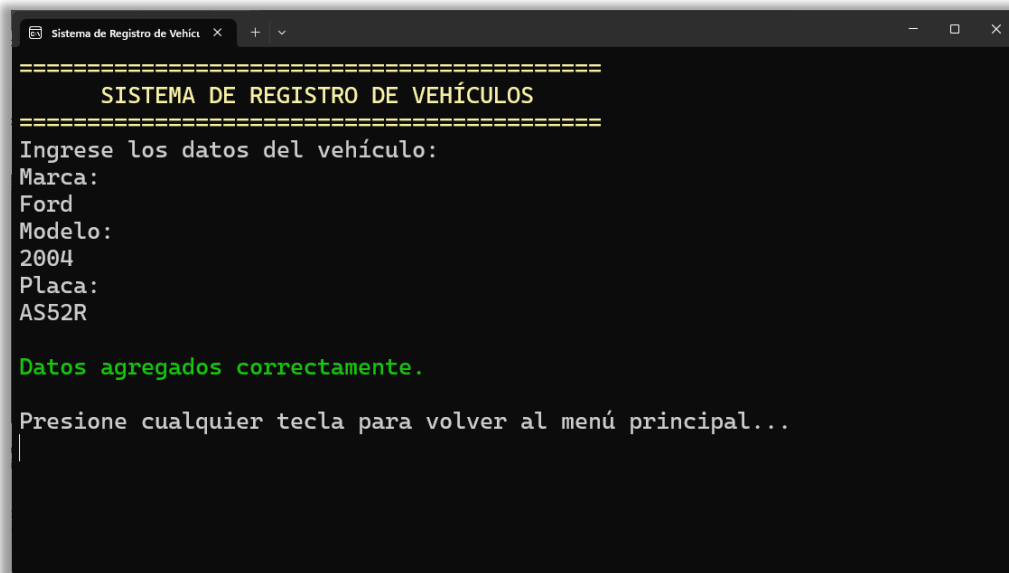
Seleccione una opción:
|
```

1. Agregar Vehículo:

Para agregar un nuevo vehículo, selecciona la opción "Agregar Vehículo" en el menú principal. A continuación, se te pedirá que ingreses la siguiente información:

- Marca: *Ingresa la marca del vehículo (por ejemplo, Toyota, Ford, etc.).*
- Modelo: *Ingresa el modelo del vehículo (por ejemplo, Corolla, Focus, etc.).*
- Placa: *Ingresa la placa del vehículo (por ejemplo, ABC123, XYZ789, etc.).*

Asegúrate de ingresar la información correctamente y sigue las instrucciones en pantalla. Una vez ingresados los datos, se guardarán en el archivo de texto.



```
Sistema de Registro de Vehículos x + -
=====
SISTEMA DE REGISTRO DE VEHÍCULOS
=====
Ingrese los datos del vehículo:
Marca:
Ford
Modelo:
2004
Placa:
AS52R

Datos agregados correctamente.

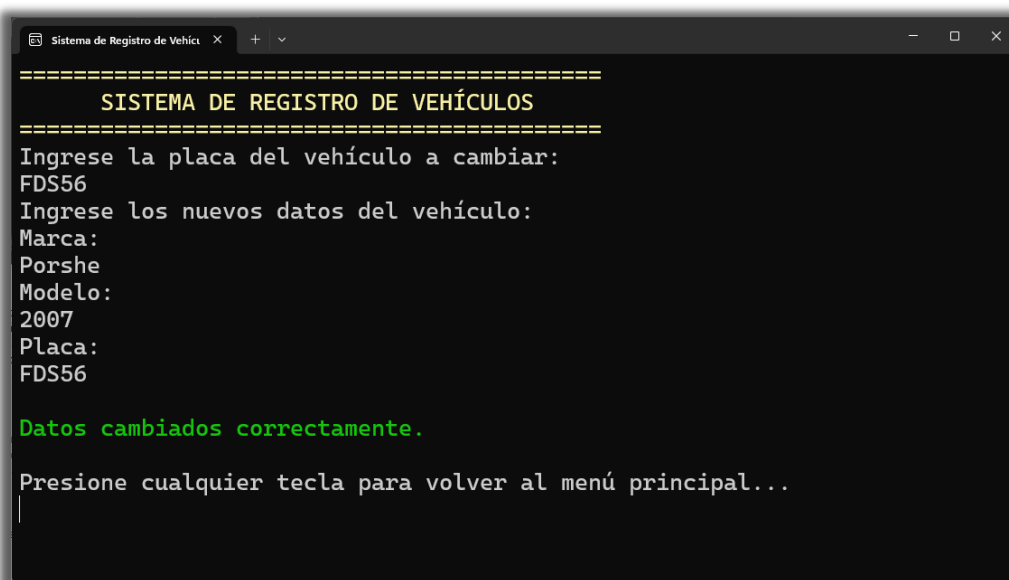
Presione cualquier tecla para volver al menú principal...
|
```

2. Borrar Vehículo:

Para eliminar un vehículo existente, selecciona la opción "Borrar Vehículo" en el menú principal. A continuación, se te pedirá que ingreses la placa del vehículo que deseas eliminar. Ingresa la placa correctamente y sigue las instrucciones en pantalla. El vehículo correspondiente será eliminado del archivo de texto.

3. Cambiar Datos de Vehículo:

Si deseas modificar los datos de un vehículo existente, selecciona la opción "Cambiar Datos de Vehículo" en el menú principal. A continuación, se te pedirá que



```
Sistema de Registro de Vehículos x + -
=====
SISTEMA DE REGISTRO DE VEHÍCULOS
=====
Ingrese la placa del vehículo a cambiar:
FDS56
Ingrese los nuevos datos del vehículo:
Marca:
Porsche
Modelo:
2007
Placa:
FDS56

Datos cambiados correctamente.

Presione cualquier tecla para volver al menú principal...
|
```

ingreses la placa del vehículo que deseas modificar. Si la placa es válida, se mostrará la información actual del vehículo y se te pedirá que ingreses los nuevos datos de marca y modelo. Sigue las instrucciones en pantalla para completar la modificación.

4. Consultar Vehículo:

Si deseas buscar y mostrar información de un vehículo específico, selecciona la opción "Consultar Vehículo" en el menú principal. A continuación, se te pedirá que ingreses la placa del vehículo que deseas consultar. Si la placa es válida, se mostrará la información del vehículo en pantalla.



```
Sistema de Registro de Vehículos x + v
=====
SISTEMA DE REGISTRO DE VEHÍCULOS
=====

=====
DATOS DE LOS VEHÍCULOS
=====

Marca: Porsche Porsche, Modelo: 2007 2006, Placa: FDS56
Marca: Ferrari, Modelo: 2018, Placa: GDSG2
Marca: Nissan, Modelo: 2023, Placa: FSDA5
Marca: Ford, Modelo: 1999, Placa: DS56D

Presione cualquier tecla para volver al menú principal...
|
```

5. Finalizar la aplicación:

Para salir de la aplicación, selecciona la opción "Salir" en el menú principal. La aplicación se cerrará y volverás al entorno de consola.

Instrucciones del programa ejercicio ABCC

Utilizando una aplicación de consola con C#, realizar un sistema de registro de pacientes, donde se pida el nombre, ciudad, edad, género y descripción de enfermedad o padecimiento. Se pide realizar un menú con las opciones de:

- Agregar paciente.
- Borrar paciente.
- Cambiar datos del paciente.
- Consultar registros.

Estos datos se guardarán en un archivo de texto con las clases y métodos anteriormente vistos.

Listado del programa ejercicio ABCC

```
using System.IO;

class Program
{
    static string filePath = "pacientes.txt"; // Ruta del archivo de texto

    static void Main(string[] args)
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("=== SISTEMA DE REGISTRO DE PACIENTES ===");
        Console.ResetColor();
        Console.WriteLine();

        bool salir = false;

        while (!salir)
        {
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.WriteLine("MENU PRINCIPAL");
            Console.WriteLine("=====\\n");
            Console.ResetColor();
            Console.WriteLine("    1. Agregar paciente");
            Console.WriteLine("    2. Borrar paciente");
            Console.WriteLine("    3. Cambiar datos de paciente");
            Console.WriteLine("    4. Consultar pacientes");
            Console.WriteLine("    5. Salir");
            Console.WriteLine();

            Console.Write("Ingrese la opción deseada: ");
            string opcion = Console.ReadLine();

            Console.Clear(); // Limpiar la pantalla

            switch (opcion)
            {
                case "1":
                    AgregarPaciente();
                    break;
                case "2":
                    BorrarPaciente();
                    break;
                case "3":
                    CambiarDatosPaciente();
                    break;
                case "4":
                    ConsultarPacientes();
                    break;
                case "5":
                    salir = true;
                    break;
                default:
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Opción inválida. Por favor, ingrese una opción válida del menú.");
                    Console.ResetColor();
                    break;
            }

            Console.WriteLine();
            Console.WriteLine("Presione cualquier tecla para continuar...");
            Console.ReadKey();
            Console.Clear(); // Limpiar la pantalla
        }
    }
}
```

```

// Método para agregar un paciente
static void AgregarPaciente()
{
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("AGREGAR PACIENTE");
    Console.WriteLine("=====");
    Console.ResetColor();
    Console.WriteLine();

    // Solicitar los datos del paciente
    Console.Write("Nombre: ");
    string nombre = Console.ReadLine();

    Console.Write("Ciudad: ");
    string ciudad = Console.ReadLine();

    Console.Write("Edad: ");
    string edadStr = Console.ReadLine();

    int edad;
    bool edadValida = int.TryParse(edadStr, out edad);

    Console.Write("Género (M/F): ");
    string genero = Console.ReadLine();

    Console.Write("Descripción de enfermedad o padecimiento: ");
    string descripcion = Console.ReadLine();

    // Validar los datos ingresados
    if (string.IsNullOrEmpty(nombre) || string.IsNullOrEmpty(ciudad) ||
        string.IsNullOrEmpty(edadStr) || !edadValida || string.IsNullOrEmpty(genero) ||
        string.IsNullOrEmpty(descripcion))
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error: Los datos ingresados son inválidos. Por favor, asegúrese
de completar todos los campos correctamente.");
        Console.ResetColor();
        return;
    }

    try
    {
        using (StreamWriter writer = new StreamWriter(filePath, true))
        {
            // Escribir los datos del paciente en el archivo de texto
            writer.WriteLine("Nombre: " + nombre);
            writer.WriteLine("Ciudad: " + ciudad);
            writer.WriteLine("Edad: " + edad);
            writer.WriteLine("Género: " + genero);
            writer.WriteLine("Descripción: " + descripcion);
            writer.WriteLine("-----");
        }

        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("Paciente agregado correctamente.");
        Console.ResetColor();
    }
    catch (IOException ex)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error al agregar paciente: " + ex.Message);
        Console.ResetColor();
    }
}

// Método para borrar un paciente
static void BorrarPaciente()
{

```

```

Console.ForegroundColor = ConsoleColor.Cyan;
Console.WriteLine("BORRAR PACIENTE");
Console.WriteLine("=====");
Console.ResetColor();
Console.WriteLine();

Console.Write("Ingrese el nombre del paciente a borrar: ");
string nombreABorrar = Console.ReadLine();

if (string.IsNullOrEmpty(nombreABorrar))
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine("Error: El nombre ingresado es inválido. Por favor, ingrese un
nombre válido.");
    Console.ResetColor();
    return;
}

bool pacienteEncontrado = false;
string[] lines = null;

try
{
    // Leer todas las líneas del archivo
    lines = File.ReadAllLines(filePath);

    using (StreamWriter writer = new StreamWriter(filePath))
    {
        // Recorrer todas las líneas del archivo
        for (int i = 0; i < lines.Length; i += 6)
        {
            // Obtener el nombre del paciente actual
            string nombre = lines[i].Replace("Nombre: ", "");

            if (nombre == nombreABorrar)
            {
                // Si se encuentra el paciente, marcar como encontrado y continuar con
la siguiente línea
                pacienteEncontrado = true;
                continue;
            }

            // Escribir las líneas del paciente en el archivo de texto (excepto las
del paciente a borrar)
            writer.WriteLine(lines[i]);
            writer.WriteLine(lines[i + 1]);
            writer.WriteLine(lines[i + 2]);
            writer.WriteLine(lines[i + 3]);
            writer.WriteLine(lines[i + 4]);
            writer.WriteLine(lines[i + 5]);
        }
    }

    if (pacienteEncontrado)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("Paciente borrado correctamente.");
        Console.ResetColor();
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine("No se encontró ningún paciente con el nombre
especificado.");
        Console.ResetColor();
    }
}
catch (IOException ex)
{

```

```

        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error al borrar paciente: " + ex.Message);
        Console.ResetColor();
    }
    finally
    {
        if (lines != null)
        {
            Array.Clear(lines, 0, lines.Length);
        }
    }
}

// Método para cambiar los datos de un paciente
static void CambiarDatosPaciente()
{
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("CAMBIAR DATOS DE PACIENTE");
    Console.WriteLine("=====");
    Console.ResetColor();
    Console.WriteLine();

    Console.Write("Ingrese el nombre del paciente: ");
    string nombreBusqueda = Console.ReadLine();

    if (string.IsNullOrEmpty(nombreBusqueda))
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error: El nombre ingresado es inválido. Por favor, ingrese un nombre válido.");
        Console.ResetColor();
        return;
    }

    Console.WriteLine();
    Console.WriteLine("NUEVOS DATOS DEL PACIENTE");

    Console.Write("Nuevo nombre (?): ");
    string nuevoNombre = Console.ReadLine();

    Console.Write("Nueva ciudad (?): ");
    string nuevaCiudad = Console.ReadLine();

    Console.Write("Nueva edad (?): ");
    string nuevaEdadStr = Console.ReadLine();

    int nuevaEdad;
    bool nuevaEdadValida = int.TryParse(nuevaEdadStr, out nuevaEdad);

    Console.Write("Nuevo género (M/F) (?): ");
    string nuevoGenero = Console.ReadLine();

    Console.Write("Nueva descripción de enfermedad o padecimiento (?): ");
    string nuevaDescripcion = Console.ReadLine();

    if (string.IsNullOrEmpty(nuevoNombre) && string.IsNullOrEmpty(nuevaCiudad) && string.IsNullOrEmpty(nuevaEdadStr) && string.IsNullOrEmpty(nuevoGenero) && string.IsNullOrEmpty(nuevaDescripcion))
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error: No se especificaron nuevos datos. Por favor, ingrese al menos un nuevo dato para actualizar.");
        Console.ResetColor();
        return;
    }

    bool pacienteEncontrado = false;

```

```

string[] lines = null;

try
{
    // Leer todas las líneas del archivo
    lines = File.ReadAllLines(filePath);

    using (StreamWriter writer = new StreamWriter(filePath))
    {
        // Recorrer todas las líneas del archivo
        for (int i = 0; i < lines.Length; i += 6)
        {
            // Obtener el nombre del paciente actual
            string nombre = lines[i].Replace("Nombre: ", "");

            if (nombre == nombreBusqueda)
            {
                // Si se encuentra el paciente, marcar como encontrado y actualizar
                pacienteEncontrado = true;

                if (!string.IsNullOrEmpty(nuevoNombre))
                {
                    lines[i] = "Nombre: " + nuevoNombre;
                }
                if (!string.IsNullOrEmpty(nuevaCiudad))
                {
                    lines[i + 1] = "Ciudad: " + nuevaCiudad;
                }
                if (!string.IsNullOrEmpty(nuevaEdadStr) && nuevaEdadValida)
                {
                    lines[i + 2] = "Edad: " + nuevaEdad;
                }
                if (!string.IsNullOrEmpty(nuevoGenero))
                {
                    lines[i + 3] = "Género: " + nuevoGenero;
                }
                if (!string.IsNullOrEmpty(nuevaDescripcion))
                {
                    lines[i + 4] = "Descripción: " + nuevaDescripcion;
                }
            }

            // Escribir las líneas del paciente en el archivo de texto
            writer.WriteLine(lines[i]);
            writer.WriteLine(lines[i + 1]);
            writer.WriteLine(lines[i + 2]);
            writer.WriteLine(lines[i + 3]);
            writer.WriteLine(lines[i + 4]);
            writer.WriteLine(lines[i + 5]);
        }
    }

    if (pacienteEncontrado)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("Datos del paciente actualizados correctamente.");
        Console.ResetColor();
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine("No se encontró ningún paciente con el nombre especificado.");
        Console.ResetColor();
    }
}
catch (IOException ex)
{

```

```

        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error al cambiar datos de paciente: " + ex.Message);
        Console.ResetColor();
    }
    finally
    {
        if (lines != null)
        {
            Array.Clear(lines, 0, lines.Length);
        }
    }
}

// Método para consultar los pacientes registrados
static void ConsultarPacientes()
{
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("CONSULTAR PACIENTES");
    Console.WriteLine("=====");
    Console.ResetColor();
    Console.WriteLine();

    try
    {
        // Leer todas las líneas del archivo
        string[] lines = File.ReadAllLines(filePath);

        if (lines.Length == 0)
        {
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine("No hay pacientes registrados.");
            Console.ResetColor();
        }
        else
        {
            Console.WriteLine("LISTA DE PACIENTES\n");

            for (int i = 0; i < lines.Length; i += 6)
            {
                Console.WriteLine(lines[i]);
                Console.WriteLine(lines[i + 1]);
                Console.WriteLine(lines[i + 2]);
                Console.WriteLine(lines[i + 3]);
                Console.WriteLine(lines[i + 4]);
                Console.WriteLine(lines[i + 5]);
                Console.WriteLine();
            }
        }
    }
    catch (IOException ex)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Error al consultar pacientes: " + ex.Message);
        Console.ResetColor();
    }
}
}

```

Manual de usuario del programa ejercicio ABCC

Introducción

El Sistema de Registro de Pacientes es una aplicación de consola que te permite registrar información de pacientes, como nombre, ciudad, edad, género y descripción de enfermedad o padecimiento. A continuación, se describen las diferentes funciones disponibles y cómo utilizarlas.

Menú Principal

Al iniciar la aplicación, se mostrará un menú con las siguientes opciones:

1. *Agregar Paciente:*

Permite ingresar los datos de un nuevo paciente y almacenarlos en el archivo de texto.

2. *Borrar Paciente:*

Permite eliminar un paciente existente del archivo de texto.

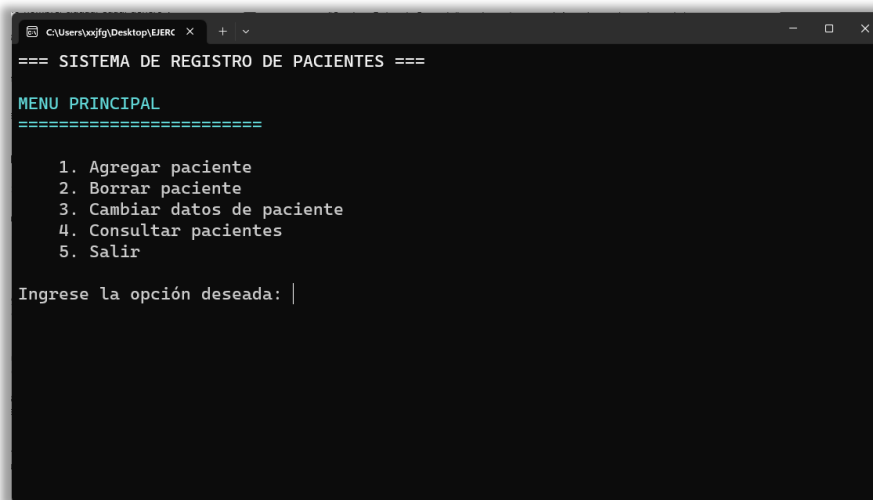
3. *Cambiar Datos de Paciente:*

Permite modificar los datos de un paciente existente en el archivo de texto.

4. *Consultar Paciente:*

Permite buscar y mostrar información de un paciente específico.

5. *Salir.*



```
=== SISTEMA DE REGISTRO DE PACIENTES ===  
  
MENU PRINCIPAL  
=====
```

1. Agregar paciente
2. Borrar paciente
3. Cambiar datos de paciente
4. Consultar pacientes
5. Salir

Ingrese la opción deseada: |

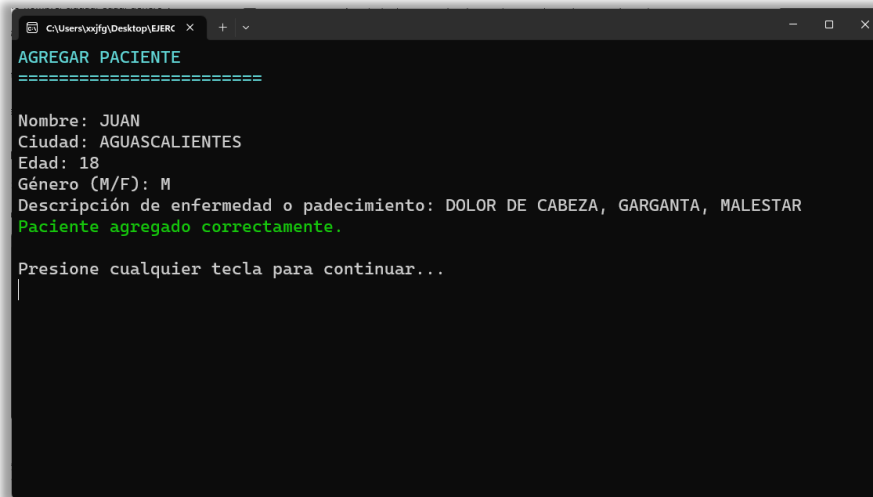
1. Agregar Paciente:

Para agregar un nuevo paciente, selecciona la opción "Agregar Paciente" en el menú principal. A continuación, se te pedirá que ingreses la siguiente información:

- Nombre: *Ingresa el nombre del paciente.*
- Ciudad: *Ingresa la ciudad de residencia del paciente.*

- Edad: *Ingresa la edad del paciente.*
- Género: *Ingresa el género del paciente (Masculino/Femenino).*
- Descripción de Enfermedad o Padecimiento: *Ingresa una breve descripción de la enfermedad o padecimiento del paciente.*

Asegúrate de ingresar la información correctamente y sigue las instrucciones en pantalla. Una vez ingresados los datos, se guardarán en el archivo de texto.



```

C:\Users\vojfy\Desktop\EJERC >
AGREGAR PACIENTE
=====

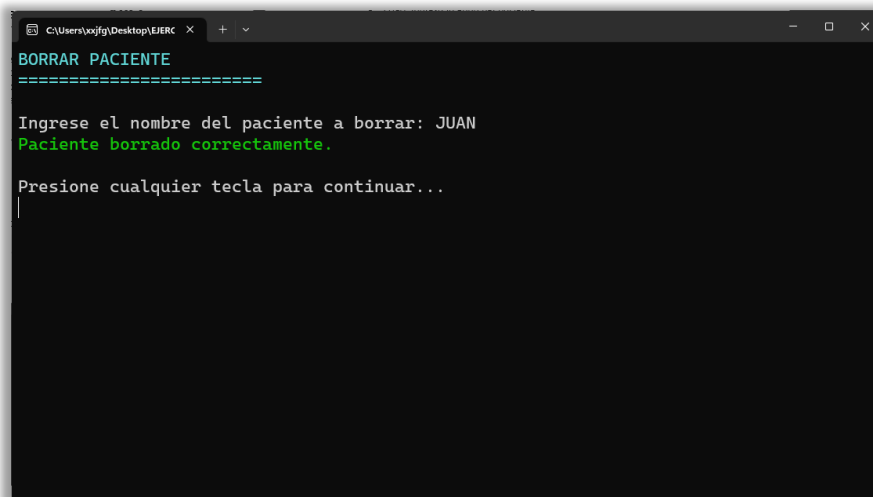
Nombre: JUAN
Ciudad: AGUASCALIENTES
Edad: 18
Género (M/F): M
Descripción de enfermedad o padecimiento: DOLOR DE CABEZA, GARGANTA, MALESTAR
Paciente agregado correctamente.

Presione cualquier tecla para continuar...

```

2. Borrar Paciente:

Para eliminar un paciente existente, selecciona la opción "Borrar Paciente" en el menú principal. A continuación, se te pedirá que ingreses el nombre del paciente que deseas eliminar. Ingresa el nombre correctamente y sigue las instrucciones en pantalla. El paciente correspondiente será eliminado del archivo de texto.



```

C:\Users\vojfy\Desktop\EJERC >
BORRAR PACIENTE
=====

Ingrese el nombre del paciente a borrar: JUAN
Paciente borrado correctamente.

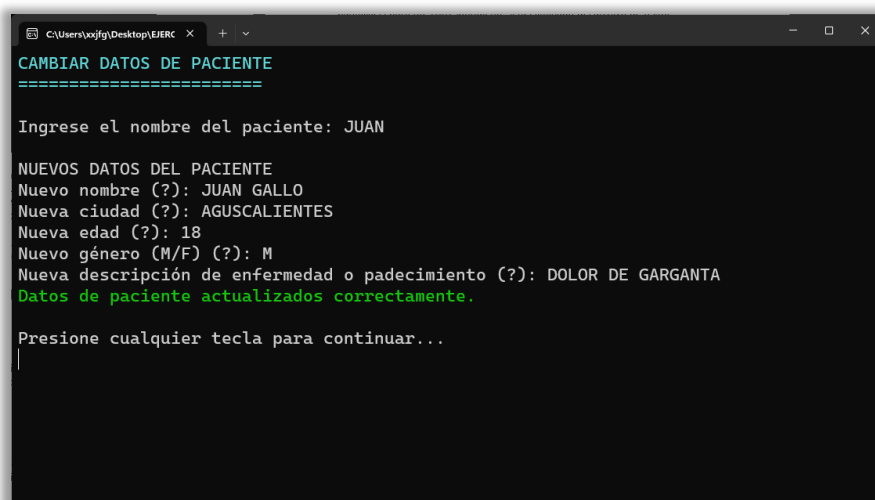
Presione cualquier tecla para continuar...

```

3. Cambiar Datos del Paciente:

Si deseas modificar los datos de un paciente existente, selecciona la opción "Cambiar Datos de Paciente" en el menú principal. A continuación, se te pedirá que

ingreses el nombre del paciente que deseas modificar. Si el nombre es válido, se mostrará la información actual del paciente y se te pedirá que ingreses los nuevos datos. Sigue las instrucciones en pantalla para completar la modificación.



```
C:\Users\vojfy\Desktop\EJERC X + -
CAMBIAR DATOS DE PACIENTE
=====

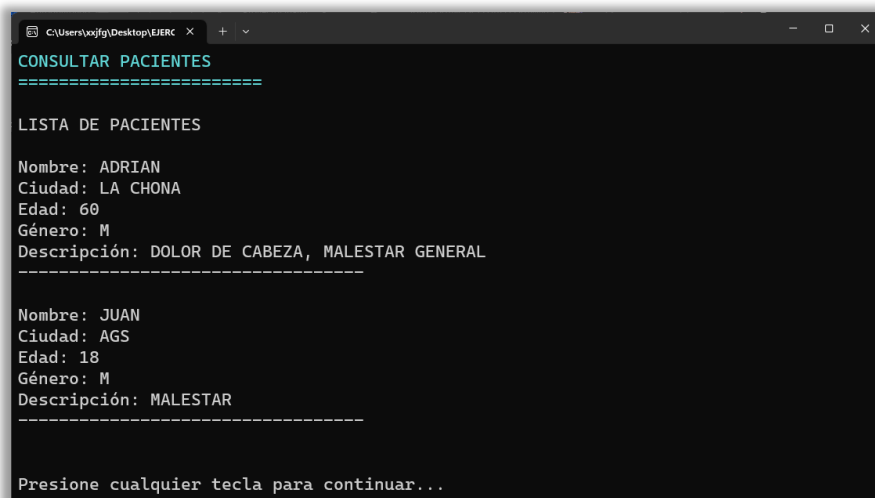
Ingrese el nombre del paciente: JUAN

NUEVOS DATOS DEL PACIENTE
Nuevo nombre (?): JUAN GALLO
Nueva ciudad (?): AGUSCALIENTES
Nueva edad (?): 18
Nuevo género (M/F) (?): M
Nueva descripción de enfermedad o padecimiento (?): DOLOR DE GARGANTA
Datos de paciente actualizados correctamente.

Presione cualquier tecla para continuar...
|
```

4. Consultar Paciente:

Si deseas buscar y mostrar información de un paciente específico, selecciona la opción "Consultar Paciente" en el menú principal. A continuación, se te pedirá que ingreses el nombre del paciente que deseas consultar. Si el nombre es válido, se mostrará la información del paciente en pantalla.



```
C:\Users\vojfy\Desktop\EJERC X + -
CONSULTAR PACIENTES
=====

LISTA DE PACIENTES

Nombre: ADRIAN
Ciudad: LA CHONA
Edad: 60
Género: M
Descripción: DOLOR DE CABEZA, MALESTAR GENERAL
-----
Nombre: JUAN
Ciudad: AGS
Edad: 18
Género: M
Descripción: MALESTAR
-----

Presione cualquier tecla para continuar...
```

5. Finalizar la aplicación:

Para salir de la aplicación, selecciona la opción "Salir" en el menú principal. La aplicación se cerrará y volverás al entorno de consola.

Ejemplo de un programa con Windows Forms

- ▶ [EJERCICIO ArchivosTxt.zip](#)

Preguntas del tema

1. ¿Cuál de las siguientes opciones describe mejor un archivo de texto?

- a) Un archivo binario que almacena información de texto.
- b) Un archivo que contiene solo caracteres legibles por humanos.
- c) Un archivo que almacena imágenes y texto.
- d) Un archivo que se puede abrir y editar solo con programas específicos.

2. ¿Cuál es una ventaja de utilizar archivos de texto en comparación con otro tipo de archivos?

- a) Los archivos de texto ocupan menos espacio en el disco.
- b) Los archivos de texto son más rápidos de leer y escribir.
- c) Los archivos de texto son compatibles con una amplia gama de programas y sistemas operativos.
- d) Los archivos de texto proporcionan una mayor seguridad para los datos almacenados.

3. ¿Cuál de las siguientes operaciones es adecuada para agregar datos a un archivo de texto existente en C#?

- a) AppendText(string)
- b) Exists(string)
- c) Write(string)
- d) OpenWrite(string)

4. ¿Qué es un stream?

- a) Clase para realizar operaciones de lectura.
- b) Abstracción de bytes.
- c) Un video en vivo.
- d) Un espacio de nombres.

5. ¿Cuál de las siguientes opciones describe mejor la función del método Close() en C# al escribir en un archivo de texto?

- a) Cierra el archivo después de escribir en él.
- b) Borra todo el contenido del archivo.
- c) Guarda los cambios realizados en el archivo.
- d) Limpia el búfer y escribe los datos almacenados en el archivo.

6. Un archivo de texto solo puede contener caracteres alfabéticos. Falso

7. El método `File.AppendText()` en C# se utiliza para agregar texto codificado UTF-8 a un archivo de texto existente o a un nuevo archivo si no existe. Verdadero
8. La clase `StreamReader` es una clase abstracta. Falso
9. El formato de archivo TXT es comúnmente utilizado para almacenar datos de texto sin formato. Verdadero
10. La clase `StreamWriter` utiliza a la clase `TextWriter` como clase base. Verdadero

Conclusiones

► Luis Pablo Esparza Terrones

Durante el curso, se exploraron diversos temas como archivos, clases, métodos, herencia, C#, Windows Forms y su aplicación práctica. Reconozco el valor de estos conceptos en el desarrollo de aplicaciones funcionales y amigables. Algunos destacaron la utilidad de la herencia para reutilizar y extender el código, mientras que otros elogiaron la simplicidad y versatilidad de Windows Forms. En general, se apreció la oportunidad de aprender herramientas y técnicas que permiten construir programas interactivos y resolver problemas del mundo real de manera creativa.

► Luis Manuel Flores García

La experiencia del curso ha sido enriquecedora para muchos participantes, ya que ha proporcionado una introducción sólida a la Programación Orientada a Objetos (POO). Algunos estudiantes valoran la estructura y claridad que brinda la POO al organizar y modularizar el código. Sin embargo, otros han expresado preocupaciones sobre posibles limitaciones inherentes a este enfoque. Existen opiniones divergentes sobre si la POO es la única forma efectiva de abordar los problemas de programación o si existen otras metodologías igualmente válidas. Algunos argumentan que la POO puede simplificar en exceso la realidad, mientras que otros consideran que es una forma eficiente de abordar la complejidad. En última instancia, la apreciación de la POO varía según las preferencias y necesidades individuales de cada estudiante.

► Adrián de Jesús Frausto Pérez

En conclusión, los archivos de texto son herramientas comunes utilizadas para almacenar y compartir información en forma de texto plano. Se caracterizan por no contener imágenes ni elementos no textuales. Los archivos con extensión .TXT son ejemplos típicos de archivos de texto y pueden ser abiertos fácilmente con cualquier editor de texto. La programación orientada a objetos, por otro lado, es un paradigma de programación ampliamente utilizado en el desarrollo de software. A diferencia de la programación estructurada, que se centra en algoritmos, la programación orientada a objetos se centra en los datos y busca adaptar el lenguaje de

programación al problema en lugar de forzar el problema al enfoque del lenguaje. En la POO, los datos y las funciones relacionadas se agrupan en unidades llamadas objetos. Estas dos áreas, los archivos de texto y la programación orientada a objetos, son conceptos distintos pero relevantes en el mundo de la tecnología. Mientras que los archivos de texto son herramientas simples para el almacenamiento de información, la programación orientada a objetos proporciona un enfoque más avanzado para resolver problemas complejos en el desarrollo de software. Ambas son importantes y utilizadas en diferentes contextos dentro de la informática y la programación.

► **Juan Francisco Gallo Ramírez**

Los conocimientos adquiridos en este semestre son de vital importancia en el campo de la programación, destaco principalmente los conocimientos adquiridos acerca de la POO, ya que en esta se basan múltiples lenguajes de programación, como C++, C#, Java, Python, etc. También fueron importantes todos aquellos conocimientos acerca de C#, y el uso de C# con Windows Forms, ya que con estos últimos ahora soy capaz de programar aplicaciones más complejas y con mayor utilidad. Acerca de los tipos de archivos creo que son importantes a la hora de tener un registro que no se borre al finalizar el programa de ejecución, por lo que son de mucha importancia. En cuanto a la materia y al semestre me ha gustado la forma en la que se ha impartido, algunas posibles mejoras pueden ser que se nos sea otorgadas posibles fuentes de información. Me ha gustado la materia y el curso.

Referencias

Anónimo. (21 de Febrero de 2013). *organizacion de datos*. Obtenido de Blogspot:
<http://organizaciondatosgiseyaneth.blogspot.com/2013/02/conceptos-basicos-de-archivos.html>

Colaboradores de Wikipedia. (19 de Abril de 2023). *Archivo (informática)*. Obtenido de Wikipedia, La enciclopedia libre.:
[https://es.wikipedia.org/w/index.php?title=Archivo_\(inform%C3%A1tica\)&oldid=150640945](https://es.wikipedia.org/w/index.php?title=Archivo_(inform%C3%A1tica)&oldid=150640945)

López Takeyas, B. (2007). *CONCEPTOS BÁSICOS DE ADMINISTRACIÓN DE ARCHIVOS*. Obtenido de
<http://www.itnuevolaredo.edu.mx/takeyas/Articulos/Archivos/ARTICULO%20Conceptos%20basicos%20de%20Administracion%20de%20Archivos.pdf>

Microsoft. (s.f.). *System.IO Espacio de nombres*. Obtenido de Microsoft:
<https://learn.microsoft.com/es-es/dotnet/api/system.io?view=net-7.0#classes>

naive. (21 de Junio de 2003). *Los archivos: tipos, extensiones y programas para su uso*. Obtenido de GEEKNETIC: <https://www.geeknetic.es/Guia/91/Los-archivos-tipos-extensiones-y-programas-para-su-uso.html>

Oracle. (2010). *Next: Técnicas básicas para la gestión de archivos*. Obtenido de
<https://docs.oracle.com/cd/E19683-01/816-3938/6ma6eh79q/index.html#:~:text=Un%20archivo%20es%20un%20contenedor,conoce%20como%20tipo%20de%20datos>.

Prado Peña, P. (2017). *Guia#12: Tema: Archivos en C#*. Obtenido de
<https://docplayer.es/27198373-Guia-12-tema-archivos-en-c.html>

The Linux Information Project. (15 de Febrero de 2005). *What is plain text?*
Obtenido de linfo.org: http://www.linfo.org/plain_text.html