Índice

- Ejercicio 1. Jugando con Rutas (Path)
- Ejercicio 2. Gestión de Directorios y Archivos
- Ejercicio 3. Copia con FileStream (Byte a Byte)
- Ejercicio 4. Datos Binarios y Exportación CSV
- ▼ Ejercicio 5. Diario con StreamWriter y StreamReader
  - Ejercicio 6. Procesamiento de Archivos y Conteo de Palabras

Ejercicios Unidad 21 - Ficheros

Descargar estos ejercicios

> **Antes de empezar**
>
> Para realizar estos ejercicios, deberás descargar los recursos del enlace de proyecto_poo. Como puedes ver, la solución está compuesta de varios proyectos. Cada uno de ellos corresponde con un ejercicio, deberás implementar todo el código, tanto de la Main como de los métodos que se piden en cada ejercicio. Cada proyecto contiene el test correspondiente, que deberás pasar para comprobar que has hecho el ejercicio correctamente.

Ejercicio 1. Jugando con Rutas (Path)

Vamos a practicar el uso de la clase estática `Path` para manipular cadenas de texto que representan rutas, sin necesidad de tocar el disco físico.

```
Ejercicio 1. Jugando con Rutas

Introduce nombre de archivo: archivo1.txt
Introduce nombre de carpeta: carpeta1
Ruta combinada: carpeta1\Documentos\archivo1.txt
Extensión: .txt
Nombre sin extensión: archivo1
Ruta con extensión cambiada: carpeta1\Documentos\archivo1.bak
Nombre aleatorio generado: h2ydt234.gsr
Pulsa una tecla para finalizar...
```

Requisitos

Crea un programa que solicite al usuario el nombre de un archivo (con extensión) y el nombre de una carpeta. Utiliza la clase `Path` para realizar las siguientes operaciones y mostrar los resultados por consola:

1. **Combinar**: Crea una ruta completa combinando la carpeta, el nombre del archivo y una subcarpeta llamada "Documentos". (Ej: `carpeta\Documentos\archivo.txt`).
2. **Extensión**: Muestra solo la extensión del archivo introducido.
3. **Nombre sin extensión**: Muestra el nombre del archivo sin la extensión.
4. **Cambio de extensión**: Simula que cambias la extensión del archivo a `.bak` y muestra la nueva ruta resultante.
5. **Nombre aleatorio**: Genera y muestra un nombre de archivo aleatorio usando el método correspondiente de la clase `Path`.

Ejercicio 2. Gestión de Directorios y Archivos

Vamos a utilizar las clases estáticas `Directory` y `File` para manipular el sistema de archivos real.

```
Ejercicio 2. Gestión de Archivos y Directorios

Carpeta Laboratorio creada.
Subcarpetas Originales y Copias creadas.
Fichero mensaje.txt creado en Originales.
Fichero copiado a Copias como mensaje_copia.txt.
Fichero original movido a raiz Laboratorio como mensaje_movido.txt.
Archivos en Copias:
mensaje_copia.txt
Archivos en Laboratorio:
mensaje_movido.txt
¿Quieres borrar todo? (S/N)
S
Carpeta Laboratorio y contenido eliminados.
Pulsa una tecla para finalizar...
```

Requisitos

Crea un programa que realice la siguiente secuencia de acciones controlada. Muestra mensajes por consola tras cada paso.

1. Crea una carpeta llamada "Laboratorio" en el directorio actual.
2. Dentro de "Laboratorio", crea dos subcarpetas: "Originales" y "Copias".
3. Usando `File.WriteAllText`, crea un fichero `mensaje.txt` dentro de "Originales" con el contenido: "Este es el contenido original.".
4. Copia el fichero `mensaje.txt` a la carpeta "Copias" con el nombre `mensaje_copia.txt`.
5. Mueve el fichero original `mensaje.txt` a la carpeta raíz "Laboratorio" cambiándole el nombre a `mensaje_movido.txt`.
6. Lista y muestra por pantalla todos los archivos que hay dentro de la carpeta "Copias" y de la carpeta "Laboratorio".
7. Pregunta al usuario si quiere borrar todo. Si dice "S", elimina la carpeta "Laboratorio" y todo su contenido (**recursivamente**).

Ejercicio 3. Copia con FileStream (Byte a Byte)

Para entender cómo funcionan los flujos de bajo nivel, vamos a copiar un fichero usándo directamente **`FileStream`**.

```
Ejercicio 3. Copia de Ficheros Byte a Byte

Fichero datos_origen.dat creado con 100 bytes aleatorios.
Copia finalizada.
Verificación exitosa: datos_destino.dat existe y tiene el mismo tamaño (100 bytes).
Pulsa una tecla para finalizar...
```

Requisitos

1. En el método `GeneraFicheroOrigen` al que le llega el nombre del fichero, genera programáticamente un fichero y escribe en él 100 bytes con valores aleatorios (puedes usar un array de bytes y `File.WriteAllBytes` para prepararlo).
2. Implementa un método `CopiaFichero(string origen, string destino)` que:
   - Abra un `FileStream` de lectura sobre el origen.
   - Abra un `FileStream` de escritura sobre el destino.
   - Lea el fichero origen **byte a byte** (usando `ReadByte()`) y lo escriba en el destino (usando `WriteByte()`) hasta llegar al final del stream.
   - *Opcional: Si te animas, hazlo con un buffer de 1024 bytes en lugar de byte a byte para ser más eficiente.*
3. Crea el método `VerificaCopia` al que le llegan los nombres de ficheros origen y destino y se encarga de Verificar que el fichero destino existe y tiene el mismo tamaño que el origen.

> 💡 **Nota:** Recuerda usar bloques `using` para asegurar que los streams se cierran y liberan los recursos correctamente.

Ejercicio 4. Datos Binarios y Exportación CSV

Vamos a diferenciar entre guardar datos en formato binario (propio de la aplicación) y formato texto (intercambiable). Fíjate en la llamada a los métodos, para crear estos y definir el código que corresponde a cada uno.

```
Ejercicio 4. Gestión de Inventario

Datos guardados en inventario.bin (Binario).
Leyendo datos desde binario:
Producto: Manzana, Precio: 0,5, Stock: 100
Producto: Pan, Precio: 1,2, Stock: 50
Producto: Leche, Precio: 0,9, Stock: 30
Producto: Huevos, Precio: 2,5, Stock: 20
Datos exportados a inventario.csv (CSV).

Pulsa una tecla para finalizar...
```

Requisitos

Crea una clase simple `Producto` con las propiedades: `string Nombre`, `double Precio`, `int Stock`.
Crea una lista con 3 o 4 productos de ejemplo.

Parte A: Binario

1. Usa `BinaryWriter` para guardar la lista de productos en un fichero `inventario.bin`. Deberás iterar la lista y escribir cada propiedad (string, double, int) en orden.
2. Usa `BinaryReader` para leer `inventario.bin`, reconstruir la lista de productos y mostrarlos por consola.

Parte B: Texto (CSV)

1. Genera un fichero `inventario.csv`. Recorre la misma lista y escribe cada producto en una línea siguiendo el formato estándar CSV: `Nombre;Precio;Stock`.
2. Usa `SteamWriter` para esto.
3. Abre el fichero CSV generado con el Bloc de notas o Excel para comprobar que es legible por humanos, a diferencia del `.bin`.

Ejercicio 5. Diario con StreamWriter y StreamReader

Vamos a crear un pequeño sistema de logs o diario personal.

```
Ejercicio 5. Diario con StreamWriter y StreamReader

Escribe una entrada (FIN para salir): Hoy he aprendido C#
Entrada guardada.
Escribe una entrada (FIN para salir): FIN

--- LEYENDO DIARIO ---
1. 08/01/2026 10:00:00 - Ayer empecé con ficheros
2. 08/01/2026 10:05:23 - Hoy he aprendido C#
```

Requisitos

1. El programa pedirá al usuario que escriba una frase para añadir a su diario.
2. Usa `StreamWriter` con la opción `append: true` para añadir esa frase al final de un fichero `diario.txt`.
   - Añade también la fecha y hora actual antes de la frase.
   - Repite el proceso hasta que el usuario escriba "FIN".
3. Cuando el usuario termine, usa `StreamReader` para abrir `diario.txt`.
4. Lee el fichero **línea a línea** y muéstralo por consola enumerando las líneas para que el usuario pueda ver todo su historial de entradas.

Ejercicio 6. Procesamiento de Archivos y Conteo de Palabras

Sistema de procesamiento de archivos y control de excepciones.

```
Ejercicio 6. Procesamiento de Archivos y Conteo de Palabras

Introduce la ruta del archivo: datos.txt
El archivo tiene 120 palabras.

Introduce la ruta del archivo: archivo_inexistente.txt
Error al leer el archivo: No se pudo leer el archivo.
Causa original: El archivo especificado no se encontró.

Introduce la ruta del archivo: datos.pdf
Error: Solo se permiten archivos de texto (.txt).

Introduce la ruta del archivo:
Error: La ruta no puede estar vacía.
```

Requisitos

Crea un sistema con las siguientes clases:

1. **Excepción personalizada**
   - `ArchivoLecturaException` (hereda de Exception). Su constructor debe aceptar un mensaje y una excepción interna.
2. **Clase GestorArchivos**
   - Método `LeerArchivo(string ruta)`:
     - Si la ruta es nula o vacía, lanza un `ArgumentException`.
     - Si la extensión del archivo no es `.txt`, lanza un `NotSupportedException`. Usa expresiones regulares para validar la extensión.
     - Intenta leer el archivo con `File.ReadAllText`.
       - Si el archivo no existe, captura el `FileNotFoundException` y lanza un `ArchivoLecturaException` con la excepción original como inner exception.
       - Si ocurre cualquier otro error de E/S (`IOException`), lanza también un `ArchivoLecturaException` encadenando la excepción original.
3. **Clase ProcesadorTexto**
   - Método `ContarPalabras(string texto)`:
     - Si el texto es nulo o vacío, lanza un `ArgumentException`.
     - Devuelve el número de palabras del texto.
4. **En el método Main:**
   - Pide al usuario la ruta de un archivo.
   - Usa `GestorArchivos.LeerArchivo` para leer el contenido.
   - Usa `ProcesadorTexto.ContarPalabras` para contar las palabras.
   - Gestiona todas las excepciones posibles:
     - Si ocurre un `ArchivoLecturaException`, muestra el mensaje y el mensaje de la excepción interna.
     - Si ocurre cualquier otra excepción, muestra un mensaje genérico.

- Ampliación!! Implementa el encadenamiento de excepciones usando inner exception.