Índice

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.

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 1. Jugando con Rutas

Introduce nombre de archivo: informe.docx
Introduce nombre de carpeta: Proyectos

Ruta combinada: Proyectos\Documentos\informe.docx
Extensión: .docx
Nombre sin extensión: informe
Ruta con extensión cambiada: Proyectos\Documentos\informe.bak
Nombre aleatorio generado: 4k2j1l3h.tmp
```

Ejercicio 2. Gestión de Directorios y Archivos

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

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`.

Requisitos

1. Crea programáticamente un fichero llamado `datos_origen.dat` y escribe en él 100 bytes con valores aleatorios (puedes usar un array de bytes y `File.WriteAllBytes` para prepararlo).
2. Implementa un método `CopiarFichero(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. Verifica que el fichero `datos_destino.dat` 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).

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.

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.

```
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#
```