Ejercicios Persistencia de Objetos

Descargar estos ejercicios

Índice

Flujos de entrada y salida

Ejercicio 5

> Escribe un programa que cree un fichero de texto de nombre `datos.txt` que se encuentre en la carpeta datos del directorio raíz de la unidad **C:**.
> El programa deberá comprobar si la carpeta existe y si no es así la creará.
> En ese fichero iremos guardando **carácter a carácter** los que se introduzca por teclado usando un adaptador `BinaryWriter` .
> Finalizaremos la introducción de caracteres al pulsar la tecla `ESC` .

Ejercicio 6

> Realiza un programa que cuente el número de caracteres de texto **unicode** de un fichero.
> **Nota**: El número de caracteres no tiene porque coincidir con el número de bytes.
>
> Para hacer la lectura del fichero debes usar un adaptador `StreamReader` .
> El nombre del fichero será pasado como argumento en la **línea de comandos**.

Ejercicio 7

> Realiza una programa programa llamado `mycp` que funcione parecido a el comando `cp` del **Linux**.
> El programa hará la copia de un archivo origen en otro destino. Introduciendo ambos como argumentos de la línea de comando.
> **Nota:** La operación de copiar se deberá realizar **byte a byte** usando **únicamente** `FileStreams` .

Ejercicio 8 (Ampliación)

> Vamos a ampliar el `mycp` del ejercicio anterior. Pero, diferencia de este, en lugar de copiar byte a byte, debes crear un `BufferedStream` de con un buffer de **100000** bytes que utilizaremos para ir copiando de **100 KB en 100 KB**.

Ejercicio 9

> Dado un fichero de **texto** con codificación **UTF8**, escribe un programa que convierta los caracteres alfabéticos que aparecen en mayúscula por caracteres alfabéticos en minúscula y

viceversa.
Nota: El cambio deberá realizarse **sobre el mismo stream** usando el método `Seek` de la clase `FileStream` .

Ejercicio 10 (Ampliación)

Programa que permita **buscar una palabra** en uno o más ficheros de texto (introducidos en la línea de comandos).

Se necesitará un método `BuscaEnFichero(string ruta, string palabra)` , que extraerá las **líneas** del fichero y llamará a la función `BuscaEnCadena(string cadena, string palabra)` que comprobará si las cadenas son iguales.

En la línea de comandos introducirás la palabra y los nombres de fichero a buscar y te mostrará un mensaje para cada fichero, en el que te indicará si ha sido encontrada en ese fichero.

Control de errores con excepciones

Ejercicio 3

Crea un programa principal donde se pida la ruta de un fichero, y posteriormente pase esta ruta a un método que lea carácter a carácter el fichero. Posteriormente, se la pasaremos a uno que modifique el fichero.
En el programa principal (`Main`), deberás controlar las excepciones que pueden saltar si el directorio en el que se encuentra el archivo no existe o este no se ha creado con antelación (`FileNotFoundException`).
Además de las anteriores excepciones, también deberás controlar la excepción que se lanza cuando se intenta modificar y el acceso no está autorizado.

Nota: Para probar el ejercicio pon el atributo del archivo a **solo lectura** al fichero.

Ejercicio 4

Modifica el **ejercicio anterior** para capturar la excepción `FileNotFoundException` antes del `finally` en los métodos de lectura y escritura.
En esta captura no mostraremos ningún mensaje y lo que haremos es relanzarla pasando en el parámetro `innerException` la referencia a la excepción que acabamos de capturar y un mensaje personalizado indicando en que ámbito se está produciendo.

```
Leyendo … <ruta>
Modificando … <ruta>
```

En el programa principal, donde capturábamos **`FileNotFoundException`** , ahora nos deberá llegar la nueva excepción donde se especifica el ámbito. Mostrando el mensaje personalizado y a continuación el mensaje original de la excepción contenido en **`e.InnerException.Message`** si **`e.InnerException != null`** .

Persistencia de objetos

✅ Ejercicio 11

> Aquí va enunciado de Xusa

Ejercicio 12

Vamos a copiar el ejercicio anterior pues partiremos del mismo. Pero en este caso vamos a ampliar la clase Alumno y vamos a serializarla a binario de forma manual.
1. Añadiremos la clase Nota:
a) Dentro definirá el tipo enumerado Modulo con los valores ED, PROG, SI, FOL, LM y GBD
b) La clase definirá los campos modulo de tipo Modulo, nota de tipo ushort y trimestre de tipo ushort y valores posibles 1, 2 y 3.
c) Definiremos accesores o getters para los 3 campos.
d) Un constructor que reciba todos los campos y un constructor copia.
e) El método void Serializa(Stream flujo) que usará un adaptador BinaryWriter para el flujo y guardará en el mismo campo a campo del objeto nota que estemos serializando. Si se produce un excepción, la capturaremos y la relanzaremos añadiendo un mensaje.
f) El método static Nota Deserializa(Stream flujo) que usará un adaptador BinaryReader para el flujo y leerá los campos de la nota en el mismo orden en que los serializamos. Gestionaremos cualquier excepción de forma análoga al método serializa.
2. Cambios en la clase Alumno:
a) Borraremos los métodos GuardaCSV y LeeCSV.
b) Añaremos una nuevo campo que será una array de notas (clase Nota) que inicializaremos a null en el constructor.
c) Añadiremos un método para añadir notas al array (dimensionándolo si procede) haciendo una copia de la nota al guardarla en el array.
d) Modificaremos el método ToString para que muestre los datos del alumno de forma similar a esta.
1 Pérez Pepa
1º 2º 3º
ED 4 2 4
PROG 9 8 3
…
e) Por último añade la serialización y deserialización de Alumno de forma análoga a cómo la hemos relizado con su nota.
Nota: Posiblemente necesites un nuevo constructor que deberá ser privado.
3. Modifica el programa principal para encajar las nuevas definiciones propuestas. Para introducir

las notas por módulo y trimestre, puedes hacerlo de forma aleatoria através del siguiente método void AñadeNotasAleatorias(Alumno a).

Ejercicio 13

> Vamos a copiar el ejercicio anterior pues partiremos del mismo. Pero en este caso vamos a realizar la serialización de forma 'automática' a través de la clase `BinaryFormatter` y el atributo `[Serializable]` como describe en los apuntes.
> Indica con un comentario en el programa principal los métodos que has tenido que quitar y por qué.

Ejercicio 14 (Ampliación larga)

> Se propone añadir también la clase notas al ejercicio de guardar y recuperar alumnos de un CSV. Sin embargo, en este caso las notas se guardarán en otro fichero denominado `notas.csv` de tal manera que se guardará junto a cada nota el nia del alumno al que pertenece (a modo de clave ajena en DB).
> Plantéate qué habría que hacer para añadir una opción de modificación de datos de un alumno.

Ficheros y expresiones regulares

Ejercicio 15

> Programa que muestre **los números línea** en un fichero que contengan una subcadena que nosotros indiquemos.
> Además del número de línea, nos indicará el **número de apariciones** de la subcadena en dicha línea.
> Si no encuentra la subcadena en todo el fichero, nos mostrará un mensaje de "*CADENA NO ENCONTRADA*".
> Para hacer este programa crearemos una **expresión regular** a partir de la cadena que nosotros indiquemos, que será la que se busque con las líneas del fichero.

Ejercicio 16 (Ampliación)

> Un diptongo está formado por dos vocales, una fuerte y una débil, o dos débiles. Las vocales fuertes son `a, e, o` ; las vocales débiles son `i, u` . La acentuación de `u` o `i` destruye el diptongo.

> Crea un programa que, a partir de un archivo que nosotros indiquemos desde teclado y usando expresiones regulares, nos permita:
>
> 1. Mostrar todas las palabras con diptongo formado por dos vocales débiles, **ordenadas** y **sin repetir**.
> 2. Mostrar del mismo modo todos los diptongos con `a`.
> 3. Buscar una **expresión regular mínima** que muestre **todos los diptongos**.