Pruebas pokemon

Índice

1º Ejercicio

o **Public static String strongestPokemon(string rutaFichero, int generation)**, que devolverá el nombre del pokémon con la generación igual a la pasada como parámetro que más ataque tenga. No tiene que ser legendario.

Está función se encarga de leer el fichero .csv y de retornar el Pokémon que más ataque tenga sin ser legendario y de la generación pasado por el usuario.

```
6 referencias | 5/5 pasando
public static string StrongestPokemon(string rutaFichero, int generacion)
{
    StreamReader SR = null;
    string StrongestPokemon = "";
    string[] Linea;
    int AtaqueMax = 0, Ataque;

    try
    {
        SR = new StreamReader(rutaFichero);
        // Descartar cabecera
        SR.ReadLine();

        // Ataque -> 6 Generacion -> 11
        while (!SR.EndOfStream)
        {
            Linea = SR.ReadLine().Split(',');
            if (Convert.ToInt32(Linea[6]) > AtaqueMax && Convert.ToInt32(Linea[11]) == generacion)
            {
                AtaqueMax = Convert.ToInt32(Linea[6]);
                StrongestPokemon = Linea[1];
            }
        }
        SR.Close();
    } catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return StrongestPokemon;
}
```

2º Ejercicio

o **Public void filterPokemon(string rutaFichero)**, que creará un nuevo fichero con toda la información de los Pokémon que tengan exactamente dos tipos. El programa no devolverá nada, pero creará un fichero de nombre “pokemonDosTipos.csv” en la ruta por defecto establecida en el IDE

```
2 referencias | 1/1 pasando
public static void FilterPokemon(string rutaFichero)
{
    string CSVnuevo = "pokemonDosTipos.csv";
    StreamReader SR = null;
    StreamWriter SW = null;
    string[] Linea;

    try
    {
        SR = new StreamReader(rutaFichero);
        SW = new StreamWriter(CSVnuevo);

        // Añadir la cabecera
        SW.WriteLine(SR.ReadLine());
        while (!SR.EndOfStream)
        {
            Linea = SR.ReadLine().Split(',');
            if (Linea[3] != "")
            {
                SW.WriteLine(string.Join(',', Linea));
            }
        }
    } catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

}
```

Pruebas de caja Negra

Nº caso	Tipo	Condición	Resultado
1	Generación válida	Números enteros entre 1 y 6	El Pokémon más fuerte de la generación dada
2	Generación No válida	Números enteros fuera del rango [1-6]	Excepción. Generación no válida
3	CSV vacío	Generación válida o no. CSV sin datos.	Resultado vacío
4	Error de lectura del CSV	El Pokémon buscado no coincide con el devuelto.	Excepción. Pokémon diferente.
5	CSV filtrado	Existen Pokémons con 2 tipos	CSV nuevo con los Pokémons que cumplen la condición
6	CSV filtrado vacío	No existen Pokémons con 2 tipos	CSV nuevo y vacío

PRUEBAS DE CAJA BLANCA

Ahora vamos a pasar con las pruebas de caja blanca.

Todas las pruebas que usan la variable “ruta” tienen la ruta del archivo CSV:

```
[TestClass]
0 referencias
public class UnitTest1
{
    string ruta = @"pokemon.csv";
```

Esta prueba se encarga de comprobar el pokémon de la generación 2 más fuerte.

```
[TestMethod]
| 0 referencias
public void GeneracionValida()
{
    int generacionValida = 2;

    string strongestPokemon = Program.StrongestPokemon(ruta, generacionValida);

    Assert.AreEqual("HeracrossMega Heracross", strongestPokemon);
}
```

Esta prueba comprueba que con un número fuera del rango [1-6] no da ningún dato por lo que no sería válido:

```
[TestMethod]
| 0 referencias
public void GeneracionInvalida()
{
    int generacionInvalida = 0;

    string strongestPokemon = Program.StrongestPokemon(ruta, generacionInvalida);

    Assert.AreEqual("", strongestPokemon);
}
```

Esta prueba se realiza con el archivo CSV vacío (Nombre cambiado para que sea más visual) por lo que cualquier dato devuelto no es válido:

```
[TestMethod]
0 referencias
public void ArchivoCSVVacio()
{
    string rutaCSVVacio = @"pokemon_vacio.csv";
    File.Create(rutaCSVVacio);

    string strongestPokemon = Program.StrongestPokemon(rutaCSVVacio, 2);

    Assert.AreEqual("", strongestPokemon);
}
```

En esta prueba se busca que en la generación dada salga, por ejemplo, “Pikachu”. Como no es el pokémon más fuerte de la generación dada, no sería un resultado válido:

```
[TestMethod]
0 referencias
public void ErrorLecturaArchivoCSV()
{
    int generacionValida = 2;

    string strongestPokemon = Program.StrongestPokemon(ruta, generacionValida);

    Assert.AreNotEqual("Pikachu", strongestPokemon);
}
```

Esta prueba comprueba que la función “FiltroPokemon()” crea el archivo CSV correctamente:

```
[TestMethod]
0 referencias
public void FiltroPokemon()
{
    string rutaCSVFiltrado = @"pokemonDosTipos.csv";

    Program.FilterPokemon(rutaCSVFiltrado);

    Assert.IsTrue(File.Exists(rutaCSVFiltrado));
}
```

Esta prueba da como resultado un archivo CSV vacío (Nombre cambiado para que sea más visual) por lo que cualquier dato dentro de él no es válido:

```
[TestMethod]
0 referencias
public void ArchivoFiltroPokemonCSVVacio()
{
    string rutaCSVVacio = @"pokemonDosTipos_vacio.csv";
    File.Create(rutaCSVVacio);

    string strongestPokemon = Program.StrongestPokemon(rutaCSVVacio, 2);

    Assert.AreEqual("", strongestPokemon);
}
```