9. Algunas bibliotecas adicionales de uso frecuente

9.1. Fecha y hora. Temporización

Desde C#, tenemos la posibilidad de manejar **fechas y horas** con facilidad. Para ello, tenemos el tipo de datos DateTime. Por ejemplo, podemos hallar la fecha (y hora) actual con:

```
DateTime fecha = DateTime.Now;
```

O crear un objeto DateTime a partir de una cierta fecha usando el constructor:

```
DateTime fecha = new DateTime(1990, 9, 18);
```

Dentro de ese tipo de datos DateTime, tenemos herramientas para saber el día (Day), el mes (Month) o el año (Year) de una fecha, entre otros, que veremos con detalle un poco más adelante. También podemos calcular otras fechas sumando a la actual una cierta cantidad de segundos (AddSeconds), días (AddDays), etc. Un ejemplo básico de su uso sería:

```
// Ejemplo_09_01a.cs
// Ejemplo básico de manejo de fechas
// Introducción a C#, por Nacho Cabanes

using System;

class Ejemplo_09_01a
{
    static void Main()
    {
        DateTime fecha = DateTime.Now;
        Console.WriteLine("Hoy es {0} del mes {1} de {2}",
            fecha.Day, fecha.Month, fecha.Year);
        DateTime manyana = fecha.AddDays(1);
        Console.WriteLine("Mañana será {0}",
            manyana.Day);
    }
}
```

Aunque también se puede escribir

```
Console.WriteLine("Fecha actual: "+ DateTime.Now);
```

Y en ese caso, aparecerán día, mes, año, hora, minutos y segundos en el formato regional que se haya escogido en el sistema operativo, que en España es de esperar que sea algo como 16/04/2022 11:01:02

Algunas de las propiedades más útiles del tipo de datos DateTime son:

- Now (fecha y hora actual de este equipo)
- Today (fecha actual, sin hora)
- Day (día del mes)
- Month (número de mes)
- Year (año)
- Hour (hora)
- Minute (minutos)
- Second (segundos)
- Millisecond (milisegundos)
- DayOfYear (día del año)
- DayOfWeek (día de la semana: su nombre en inglés, que se puede convertir en un número del 0 -domingo- al 6 -sábado- si se fuerza su tipo a entero, anteponiéndole "(int)").

Si sólo se desea mostrar fechas en formatos concretos, no hace falta acceder a cada una de esas propiedades, sino que se puede emplear directamente ToString, con varios formatos predefinidos, como

```
using System;

class FormatosDeFecha
{
    static void Main()
    {
        DateTime unaFecha = new DateTime(1990, 12, 31);
        Console.WriteLine(unaFecha);

        Console.WriteLine("d= " + unaFecha.ToString("d"));
        Console.WriteLine("D= " + unaFecha.ToString("D"));
        Console.WriteLine("g= " + unaFecha.ToString("g"));
        Console.WriteLine("t= " + unaFecha.ToString("t"));
    }
}
```

Que mostraría como resultado:

```
31/12/1990 0:00:00
d= 31/12/1990
D= lunes, 31 de diciembre de 1990
g= 31/12/1990 0:00
t= 0:00
```

Puedes encontrar más detalles sobre los formatos existentes en la referencia oficial de Microsoft, MSDN:

https://docs.microsoft.com/es-es/dotnet/api/system.datetime.tostring?view=netframework-4.8

Para calcular nuevas fechas a partir de una dada, podemos usar métodos que generan un nuevo DateTime, como:

- AddDays (que aparece en el ejemplo anterior), AddMonths, AddHours, AddMinutes, AddSeconds, AddMilliseconds.
- También un Add más genérico (para sumar una fecha a otra) y un Subtract también genérico (para restar una fecha de otra).

Cuando restamos dos fechas, obtenemos un dato de tipo "intervalo de tiempo" (TimeSpan), del que podemos saber detalles como la cantidad de días (sin decimales, Days, o con decimales, TotalDays), como se ve en este ejemplo:

```
// Ejemplo_09_01b.cs
// Diferencia entre dos fechas
// Introducción a C#, por Nacho Cabanes

using System;

class Ejemplo_09_01b
{
    static void Main()
    {
        DateTime fechaActual = DateTime.Now;
        DateTime fechaNacimiento = new DateTime(1990, 9, 18);

        TimeSpan diferencia =
            fechaActual.Subtract(fechaNacimiento) ;

        Console.WriteLine("Han pasado {0} días",
            diferencia.Days);

        Console.WriteLine("Si lo quieres con decimales: {0} días",
            diferencia.TotalDays);

        Console.WriteLine("Y si quieres más detalles: {0}",
            diferencia);
    }
}
```

El resultado de este programa sería algo como

```
Han pasado 8886 días
Si lo quieres con decimales: 8886,41919806277 días
Y si quieres más detalles: 8886.10:03:38.7126236
```

También podemos hacer una **pausa** en la ejecución: si necesitamos que nuestro programa se detenga una cierta cantidad de tiempo, no hace falta que usemos un "while" que compruebe la hora continuamente, sino que podemos "bloquear" (Sleep) el subproceso (o hilo, "Thread") que representa nuestro programa durante una cierta cantidad de milésimas de segundo con: Thread.Sleep(5000);

Este método pertenece a System.Threading, que deberíamos incluir en nuestro apartado de"using", o bien emplear la llamada completa:

```
// Ejemplo_09_01c.cs
// Pausas
// Introducción a C#, por Nacho Cabanes

using System;
using System.Threading;

class Ejemplo_09_01c
{
    static void Main()
    {
        DateTime fecha = DateTime.Now;
        Console.WriteLine("Son las {0}:{1}:{2}",
            fecha.Hour, fecha.Minute, fecha.Second);
        Console.WriteLine("Vamos a esperar 3 segundos...");
        Thread.Sleep(3000);
        fecha = DateTime.Now;
        Console.WriteLine("Ahora son las {0}:{1}:{2}",
            fecha.Hour, fecha.Minute, fecha.Second);
    }
}
```

Ejercicios propuestos:

(9.1.1) Crea una versión mejorada del ejemplo 09_01a, que muestre el nombre del mes (usa un array para almacenar los nombres y accede a la posición correspondiente), la hora, los minutos, los segundos y las décimas de segundo (no las milésimas). Si los minutos o los segundos son inferiores a 10, deberán mostrarse con un cero inicial, de modo que siempre aparezcan con dos cifras.

(9.1.2) Crea un reloj que se muestre en pantalla, y que se actualice cada segundo (usando "Sleep"). En esta primera aproximación, el reloj se escribirá con "WriteLine", de modo que aparecerá en la primera línea de pantalla, luego en la segunda, luego en la tercera y así sucesivamente (en el próximo apartado veremos cómo hacer que se mantenga fijo en unas ciertas coordenadas de la pantalla).

(9.1.3) Crea un programa que pida al usuario su fecha de nacimiento, y diga de qué día de la semana se trataba, usando ".DayOfWeek".
(9.1.4) Crea un programa que muestre el calendario del mes actual (pista: primero deberás calcular qué día de la semana es el día 1 de este mes). Deberá ser algo como:

```
    Abril 2022

lu ma mi ju vi sá do
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

(9.1.5) Crea un programa que muestre el calendario correspondiente al mes y el año que se indiquen como parámetros en línea de comandos.
(9.1.6) Crea un programa que vuelque a un fichero de texto el calendario del año que se indique como parámetro en línea de comandos. Deben mostrarse tres meses en anchura: el primer bloque contendrá enero febrero y marzo, el segundo tendrá abril, mayo y junio y así sucesivamente.
(9.1.7) Crea un programa que pregunte al usuario el día y mes en que nació, y le diga cuántos días faltan hasta su próximo cumpleaños.

9.2. Más posibilidades de la "consola"

En "Console" hay mucho más que ReadLine y WriteLine, aunque no todas las posibilidades están incluidas en la primera versión de la "plataforma punto net", sino a partir de la versión 2 (de enero de 2006). Vamos a ver algunas de las funcionalidades adicionales que nos pueden resultar más útiles:

- Clear(): borra la pantalla.
- ForegroundColor: cambia el color de primer plano (para indicar los colores, hay definidas constantes como "ConsoleColor.Yellow", que se detallan al final de este apartado).
- BackgroundColor: cambia el color de fondo (para el texto que se escriba a partir de entonces; si se quiere borrar la pantalla con un cierto color, se deberá primero cambiar el color de fondo y después usar "Clear").
- SetCursorPosition(x, y): cambia la posición del cursor ("x" se empieza a contar desde el margen izquierdo, empezando en 0, e "y" desde la parte superior de la pantalla, también comenzando desde 0).

- Readkey(interceptar): lee una tecla desde teclado. El parámetro "interceptar" es un "bool", y es opcional. Indica si se debe capturar la tecla sin permitir que se vea en pantalla ("true" para que no se vea, "false" para que sí se muestre). Si no se indica este parámetro, la tecla pulsada se muestra en pantalla.
- KeyAvailable: indica si hay alguna tecla disponible para ser leída (es un "bool"). Permite no bloquear un programa hasta que se pulse una tecla, que es lo que ocurriría si se llama directamente a Readkey. Veremos un ejemplo dentro de poco.
- Title: el título que se va a mostrar en la consola (es un "string")
- Como el tamaño de la ventana de consola no necesariamente será algo prefijado en el sistema, se puede saber o fijar con WindowHeight y WindowWidth. Además, existe un "buffer" que puede ser mayor que la zona visible de la pantalla (y en ese caso, se podría hacer scroll para ver el resto). Su tamaño se podría conocer o cambiar con BufferHeight y BufferWidth. Tanto el ancho como el alto de ventana se pueden fijar a la vez con SetWindowSize(w,h) y los del buffer con SetBufferSize(w,h).

```
// Ejemplo_09_02a.cs
// Más posibilidades de "System.Console"
// Introducción a C#, por Nacho Cabanes

using System;

class Ejemplo_09_02a
{
    static void Main()
    {
        int posX, posY;

        Console.Title = "Ejemplo de consola";
        Console.SetWindowSize (80, 25);
        Console.BackgroundColor = ConsoleColor.Green;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.Clear();

        posY = 10;  // En la fila 10
        Random r = new Random(DateTime.Now.Millisecond);
        posX = r.Next(20, 40); // Columna al azar entre 20 y 40
        Console.SetCursorPosition(posX, posY);
        Console.WriteLine("Bienvenido");

        Console.ForegroundColor = ConsoleColor.Blue;
        Console.SetCursorPosition(10, 15);
        Console.Write("Pulsa 1 o 2: ");
        ConsoleKeyInfo tecla;
        do
        {
            tecla = Console.ReadKey(false);
        }
        while ((tecla.KeyChar != '1') && (tecla.KeyChar != '2'));
```

```
            int maxY = Console.WindowHeight;
            int maxX = Console.WindowWidth;
            Console.SetCursorPosition(maxX-50, maxY-1);
            Console.ForegroundColor = ConsoleColor.Red;
            Console.Write("Pulsa una tecla para terminar... ");
            Console.ReadKey(true);
        }
}
```

Para comprobar el valor de una **tecla**, como se ve en el ejemplo anterior, tenemos que usar una variable de tipo "ConsoleKeyInfo" (información de tecla de consola). Un ConsoleKeyInfo tiene propiedades como:

- KeyChar, que representa el carácter que se escribiría al pulsar esa tecla. Por ejemplo, podríamos hacer `if (tecla.KeyChar == '1') ...`
- Key, que se refiere al código de la tecla (porque hay teclas que no tienen un carácter visualizable, como F1 o las teclas de movimiento del cursor). Por ejemplo, para comprobar la tecla ESC podríamos hacer `if (tecla.Key == ConsoleKey.Escape) ...` . Algunos de los códigos de tecla disponibles son:
  - Teclas de edición y control como, como: Backspace (Tecla Retroceso), Tab (Tecla del tabulador), Enter (Tecla Intro), Pause (Tecla de Pausa), Escape (Tecla Esc), Spacebar (Tecla de barra espaciadora), PrintScreen (Tecla ImprPant), Insert (Tecla Ins o Insert), Delete (Tecla Supr o Suprimir).
  - Teclas de movimiento del cursor, como: PageUp (Tecla Re.Pág.), PageDown (Tecla Av.Pág.), End (Tecla Fin), Home (Tecla Inicio), LeftArrow (Tecla de flecha izquierda), UpArrow (flecha arriba), RightArrow (flecha derecha), DownArrow (flecha abajo).
  - Teclas alfabéticas, como: A (Tecla A), B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
  - Teclas numéricas, como: D0 (Tecla 0), D1, D2, D3, D4, D5, D6, D7, D8, D9
  - Teclado numérico adicional: NumPad0 (Tecla 0 del teclado numérico), NumPad1, NumPad2, NumPad3, NumPad4, NumPad5, NumPad6, NumPad7, NumPad8, NumPad9, Multiply (Tecla Multiplicación), Add (Tecla Suma), Separator (Tecla Separador de miles, si la hubiera), Subtract (Tecla Resta), Decimal (Tecla Decimal), Divide (Tecla División)
  - Teclas de función: F1, F2 y sucesivas (hasta F24)
  - Teclas especiales de Windows: LeftWindows (Tecla izquierda con el logotipo de Windows), RightWindows (Tecla derecha con el logotipo de Windows, si existe)

- Incluso teclas multimedia, si el teclado las incorpora, como: VolumeMute (Tecla Silenciar el volumen), VolumeDown (Bajar el volumen), VolumeUp (Subir el volumen), MediaNext (Siguiente pista de multimedia), etc.
- Modifiers, que permite comprobar si se han pulsado simultáneamente teclas modificadoras: Alt, Shift o Control. Se usaría con el operador binario "and", como en este ejemplo:

```
if ((tecla.Modifiers & ConsoleModifiers.Alt) != 0)
    Console.Write("Has pulsado Alt");
```

Como ya hemos anticipado, Console.Readkey hace que el programa se quede **parado** hasta que se pulse una tecla. Si queremos hacer algo mientras que el usuario no pulse ninguna tecla, podemos emplear Console.KeyAvailable para comprobar si ya se ha pulsado alguna tecla que haya que analizar, como en este ejemplo, que permite mover un símbolo a izquierda y derecha, pero muestra una animación simple si no pulsamos ninguna tecla:

```
// Ejemplo_09_02b.cs
// No bloquear el programa con Console.Readkey
// Introducción a C#, por Nacho Cabanes

using System;
using System.Threading;

class Ejemplo_09_02b
{
    static void Main()
    {
        int posX=40, posY=10;
        string simbolos = "^>v<";
        byte simboloActual = 0;
        bool terminado = false;

        do
        {
            Console.Clear();
            Console.SetCursorPosition(posX, posY);
            Console.Write( simbolos[ simboloActual ]);
            Thread.Sleep(500);
            if (Console.KeyAvailable)
            {
                ConsoleKeyInfo tecla = Console.ReadKey(true);
                if (tecla.Key == ConsoleKey.RightArrow) posX++;
                if (tecla.Key == ConsoleKey.LeftArrow) posX--;
                if (tecla.Key == ConsoleKey.Escape) terminado = true;
            }
            simboloActual++;
            if (simboloActual > 3) simboloActual = 0;
        }
        while ( ! terminado );
    }
```

```
}
```

Al igual que en este ejemplo, será recomendable hacer una pequeña **pausa** entre una comprobación de teclas y la siguiente, con Thread.Sleep, tanto para que la animación no sea demasiado rápida como para no hacer un consumo muy alto de procesador para tareas poco importantes.

Los **colores** que tenemos disponibles (y que se deben escribir precedidos con "ConsoleColor") son: Black (negro), DarkBlue (azul marino), DarkGreen (verde oscuro), DarkCyan (verde azulado oscuro), DarkRed (rojo oscuro), DarkMagenta (fucsia oscuro o púrpura), DarkYellow (amarillo oscuro u ocre), Gray (gris), DarkGray (gris oscuro), Blue (azul), Green (verde), Cyan (aguamarina o verde azulado claro), Red (rojo), Magenta (fucsia), Yellow (amarillo), White (blanco).

Ejercicios propuestos:

(9.2.1) Crea un programa que muestre una "pelota" (la letra "O") rebotando en los bordes de la pantalla. Para que no se mueva demasiado rápido, haz una pausa de 50 milisegundos entre un "fotograma" de la animación y otro.

(9.2.2) Crea una versión de la "base de datos de ficheros" (ejemplo 04_06a) que use colores para ayudar a distinguir los mensajes del programa de las respuestas del usuario, y que no necesite pulsar Intro tras escoger cada opción.

(9.2.3) Crea un programa que permita "dibujar" en consola, moviendo el cursor con las flechas del teclado y pulsando "espacio" para dibujar un punto o borrarlo.

(9.2.4) Crea una versión del programa de "dibujar" en consola (9.2.3), que permita escribir más caracteres (por ejemplo, las letras), así como mostrar ayuda (pulsando F1), guardar el contenido de la pantalla en un fichero de texto (con F2) o recuperarlo (con F3).

(9.2.5) Crea una versión mejorada del programa 9.1.2 (mostrar el reloj actualizado en pantalla, que lo dibuje siempre en la esquina superior derecha de la pantalla).

(9.2.6) Crea un programa que "dibuje" un círculo en consola, usando las ecuaciones x = xCentro + radio * coseno(ángulo), y = yCentro + radio * seno(ángulo). Si tu array es de 24x79, las coordenadas del centro serían (12,40).

9.3. Números aleatorios

En un programa de gestión o una utilidad que nos ayude a administrar un sistema, no es habitual que podamos permitir que las cosas ocurran al azar. Pero los juegos se encuentran muchas veces entre los ejercicios de programación más completos, y para un juego sí puede ser conveniente que haya algo de azar, para que una partida no sea exactamente igual a la anterior.

Generar números al azar ("números aleatorios") usando C# no es difícil: debemos crear un objeto de tipo "Random" (una única vez), y luego llamaremos a "Next" cada vez que queramos obtener valores entre dos extremos:

```
// Creamos un objeto Random
Random generador = new Random();

// Generamos un número entre dos valores dados
// (entre 1 y 100: el segundo límite no está incluido)
int aleatorio = generador.Next(1, 101);
```

Como alternativa, una forma simple de obtener un único número "casi al azar" entre 0 y 999 es tomar las milésimas de segundo de la hora actual:

```
int falsoAleatorio = DateTime.Now.Millisecond;
```

Pero esta forma simplificada no sirve si necesitamos obtener dos números aleatorios a la vez, porque los dos datos se obtendrían en el mismo milisegundo y tendrían el mismo valor (o valores muy próximos); en ese caso, no habría más remedio que utilizar (un único) "Random" y llamar dos veces a "Next".

Vamos a ver un ejemplo, que muestre en pantalla dos números al azar:

```
// Ejemplo_09_03_01a.cs
// Números al azar
// Introducción a C#, por Nacho Cabanes

using System;

class Ejemplo_09_03_01a
{
    static void Main()
    {
        Random r = new Random();
        int aleatorio = r.Next(1, 11);
        Console.WriteLine("Un número entre 1 y 10: {0}", aleatorio);
        int aleatorio2 = r.Next(10, 21);
        Console.WriteLine("Otro entre 10 y 20: {0}", aleatorio2);
    }
```

```
}
```

Si tenemos que generar números con decimales, existe la alternativa de usar ".NextDouble()", que genera un número real de doble precisión entre 0 (incluido) y 1 (no incluido).

Ejercicios propuestos:
(9.3.1.1) Crea un programa que imite el lanzamiento de un dado, generando un número al azar entre 1 y 6.
(9.3.1.2) Crea un programa que genere un número al azar entre 1 y 100. El usuario tendrá 6 oportunidades para acertarlo.
(9.3.1.3) Mejora el programa del ahorcado (4.4.9.3), para que la palabra a adivinar no sea tecleada por un segundo usuario, sino que se escoja al azar de un "array" de palabras prefijadas (por ejemplo, nombres de ciudades).
(9.3.1.4) Crea un programa que genere un array relleno con 100 números enteros al azar entre -10000 y 10000. Luego deberá calcular y mostrar su media.
(9.3.1.5) Crea un programa que "dibuje" asteriscos en 100 posiciones al azar de la pantalla. Para ayudarte para escribir en cualquier coordenada, puedes usar un array de dos dimensiones (usando como tamaño: 24 para el alto y 79 para el ancho), que primero rellenes y luego dibujes en pantalla. Como alternativa, puedes emplear Console.SetCursorPosition.
(9.3.1.6) Crea un programa que genere un array, lo rellene con 100 números reales al azar entre -1000 y 1000 con dos cifras decimales (números como -123,45). Luego deberá ordenar los datos y mostrarlos.

9.4. Lectura de directorios

Si queremos analizar el contenido de un directorio, podemos emplear las clases Directory y DirectoryInfo.

La clase Directory contiene métodos para crear un directorio (CreateDirectory), borrarlo (Delete), moverlo (Move), comprobar si existe (Exists), etc. Por ejemplo, podríamos crear un directorio con:

```
// Ejemplo_09_04a.cs
// Crear un directorio
// Introducción a C#, por Nacho Cabanes

using System.IO;

class Ejemplo_09_04a
{
```

```
    static void Main()
    {
        string miDirectorio = @"d:\datos";
        if (!Directory.Exists(miDirectorio))
            Directory.CreateDirectory(miDirectorio);
    }
}
```

(la clase Directory está declarada en el espacio de nombres System.IO, por lo que deberemos añadirlo entre los "using" de nuestro programa).

También tenemos un método "GetFiles" que nos permite obtener la lista de ficheros que contiene un directorio. Así, podríamos listar todo el contenido de una carpeta con:

```
// Ejemplo_09_04b.cs
// Lista de ficheros en un directorio
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_09_04b
{
    static void Main()
    {
        string miDirectorio = @"c:\";
        string[] listaFicheros;

        listaFicheros = Directory.GetFiles(miDirectorio);
        foreach(string fichero in listaFicheros)
            Console.WriteLine(fichero);
    }
}
```

Se puede añadir un segundo parámetro a "GetFiles", que sería el patrón que deben seguir los nombres de los ficheros que buscamos, como "*.txt". Por ejemplo, podríamos saber todos los fuentes de C# ("*.cs") de la carpeta actual (".") con:

```
// Ejemplo_09_04c.cs
// Lista de ficheros en un directorio
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_09_04c
{
    static void Main()
    {
        string[] listaFicheros = Directory.GetFiles(".", "*.cs");
```

```
        foreach(string fichero in listaFicheros)
            Console.WriteLine(fichero);
    }
}
```

Si necesitamos más detalles, la clase DirectoryInfo permite obtener información sobre fechas de creación, modificación y acceso, y, de forma análoga, FileInfo nos permite conseguir información similar sobre un fichero. Podríamos usar estas dos clases para ampliar el ejemplo anterior, y que no sólo muestre el nombre de cada fichero, sino otros detalles adicionales como el tamaño y la fecha de creación:

```
// Ejemplo_09_04d.cs
// Lista detallada de ficheros en un directorio
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_09_04d
{
    static void Main()
    {
        string miDirectorio = @"c:\";
        DirectoryInfo dir = new DirectoryInfo(miDirectorio);
        FileInfo[] infoFicheros = dir.GetFiles();
        foreach (FileInfo infoUnFich in infoFicheros)
        {
            Console.WriteLine("{0}, de tamaño {1}, creado {2}",
                infoUnFich.Name,
                infoUnFich.Length,
                infoUnFich.CreationTime);
        }

    }
}
```

que escribiría cosas como

```
hiberfil.sys, de tamaño 6775930880, creado 15/07/2013 17:48:07
```

También se puede comprobar los valores de los "**atributos**" de un fichero, que permiten saber detalles como si está oculto, si es de sólo lectura, si está pendiente de copia de seguridad... Como es algo más complejo y menos frecuente, vamos a ver sólo un ejemplo de su forma de uso, que nos permita saber si un fichero está oculto. Para ello, se debe comparar el valor de la propiedad "Attributes" con los posibles valores de la enumeración "FileAttributes", usando una multiplicación bit a bit (un producto lógico, con el operador "&"):

```
bool oculto =
    (infoUnFich.Attributes & FileAttributes.Hidden) == FileAttributes.Hidden;
```

Ejercicios propuestos:

(9.4.1) Crea un programa que muestre en pantalla el contenido de un fichero de texto, cuyo nombre escoja el usuario. Si el usuario no sabe el nombre, podrá pulsar "Intro" y se le mostrará la lista de ficheros existentes en el directorio actual, para luego volver a preguntarle el nombre del fichero.

(9.4.2) Crea un programa que cree un fichero de texto a partir del contenido de todos los ficheros de texto existentes en la carpeta actual.

(9.4.3) Crea un programa que permita "pasear" por la carpeta actual, al estilo del antiguo "Comandante Norton": mostrará la lista de ficheros y subdirectorios de la carpeta actual, y permitirá al usuario moverse hacia arriba o abajo dentro de la lista usando las flechas del cursor. El elemento seleccionado se mostrará en color distinto del resto.

(9.4.4) Mejora el ejercicio 9.4.3 para que muestre directorios (en primer lugar) y ficheros (a continuación), y permita entrar a un directorio si se pulsa Intro sobre él (en ese momento, se actualizará la lista de ficheros y directorios, para mostrar el contenido del directorio al que se ha accedido).

(9.4.5) Mejora el ejercicio 9.4.4 para que contenga dos paneles, uno al lado del otro, cada uno de los cuales podrá estar mostrando el contenido de un directorio distinto. Si se pulsa el "tabulador", cambiará el panel activo.

(9.4.6) Mejora el ejercicio 9.4.5, para que se pueda "seleccionar un fichero" pulsando "Espacio" o "Insert". Los ficheros seleccionados se mostrarán en un color distinto. Se podrán deseleccionar volviendo a pulsar "Espacio" o "Insert". Si se pulsa F5, los ficheros seleccionados en la carpeta actual del panel actual se copiarán a la carpeta del otro panel.

(9.4.7) Mejora el ejercicio 9.4.6, para que mientras se están copiando ficheros, el programa muestre una "barra de progreso" de color amarillo, que indicará el porcentaje de ficheros que ya se han copiado.

9.5. Llamadas al sistema

Si hay algo que no sepamos o podamos hacer, pero que alguna utilidad del sistema operativo sí es capaz de hacer por nosotros, podemos hacer que ella realice ese trabajo. La forma de llamar a otras órdenes del sistema operativo (e incluso programas externos de casi cualquier tipo) es creando un nuevo proceso con "Process.Start". Por ejemplo, podríamos lanzar el bloc de notas de Windows con:

```
Process proc = Process.Start("notepad.exe");
```

(que necesitará System.Diagnostics en la lista de "using").

En los actuales sistemas operativos multitarea se da por sentado que no es necesario esperar a que termine la otra tarea, sino que nuestro programa puede proseguir. Si, aun así, queremos esperar a que se complete la otra tarea (por ejemplo, porque deseemos comprobar si ha existido algún error durante la ejecución), lo conseguiríamos con "WaitForExit", añadiendo esta segunda línea:

```
proc.WaitForExit();
```

Un programa completo que lanzara el bloc de notas y que esperase a que se cerrase podría ser:

```
// Ejemplo_09_05a.cs
// Lanzar otro proceso y esperar
// Introducción a C#, por Nacho Cabanes

using System;
using System.Diagnostics;

class Ejemplo_09_05a
{
    static void Main()
    {
        Console.WriteLine("Lanzando el bloc de notas...");
        Process proc = Process.Start("notepad.exe");
        Console.WriteLine("Esperando a que se cierre...");
        proc.WaitForExit();
        Console.WriteLine("Terminado!");
    }
}
```

Si se desea añadir algún parámetro al programa que se lanza, se puede hacer usando una versión sobrecargada de Process.Start:

```
Process proc = Process.Start("notepad.exe","fichero.txt");
```

De igual modo, si no deseamos esperar indefinidamente, sino una cierta cantidad máxima de segundos, se puede indicar como parámetro opcional de WaitForExit:

```
proc.WaitForExit(5000);
```

Así, una variante del programa anterior podría ser:

```
// Ejemplo_09_05b.cs
// Lanzar otro proceso y esperar (2)
// Introducción a C#, por Nacho Cabanes
```

```
using System;
using System.Diagnostics;

class Ejemplo_09_05b
{
    static void Main()
    {
        Console.WriteLine("Lanzando el bloc de notas...");
        Process proc = Process.Start("notepad.exe", "fichero.txt");
        Console.WriteLine("Esperando 5 segundos...");
        proc.WaitForExit(5000);
        Console.WriteLine("Terminado!");
    }
}
```

Se puede saber si el proceso ha salido con algún código de error mirando el valor de "proc.ExitCode". También se puede afinar un poco más el comportamiento, por ejemplo, forzando a que la ventana del nuevo proceso esté inicialmente minimizada (Minimized) o incluso oculta (Hidden) con la ayuda de la clase "ProcessStartInfo", así:

```
ProcessStartInfo proc = new ProcessStartInfo("miPrograma.exe");
proc.WindowStyle = ProcessWindowStyle.Minimized;
Process.Start(proc);
```

Ejercicios propuestos:

(9.5.1) Crea un programa que muestre un menú, que te permita lanzar ciertas aplicaciones que uses con frecuencia pulsando sólo una tecla (p.ej., del 0 al 9).

(9.5.2) Haz un programa que mida el tiempo que tarda en ejecutarse un cierto proceso. Este proceso se le indicará como parámetro en la línea de comandos.

(9.5.3) Crea un programa que lance un compresor de línea de comandos (como 7-zip o RAR, por ejemplo) para comprimir un fichero cuyo nombre escoja el usuario, usando la contraseña "1234".

(9.5.4) Crea un programa que trate de descomprimir el fichero comprimido que has creado en el ejercicio anterior, probando todas las contraseñas numéricas de 4 cifras (desde "0000" hasta "9999"). Podrás saber si has encontrado la contraseña correcta porque en ese intento proc.ExitCode valdrá 0 (para indicar que el proceso anterior se ha podido lanzar sin problemas), mientras que en el resto de intentos será un número distinto de cero (puedes mirar la documentación del compresor que hayas utilizado, si quieres saber qué códigos de error concretos devuelve al sistema operativo). Quizá te interese lanzar el (des)compresor como ventana oculta o al menos como ventana minimizada, para que las 10.000 ventanas que pueden llegar a aparecer no te impidan la visión del escritorio y, por tanto, el manejo normal del sistema.

(9.5.5) Mejora el ejercicio 9.4.3 (la versión básica del programa "tipo Comandante Norton") para que, si se pulsa Intro sobre un cierto fichero, lance el correspondiente proceso.
(9.5.6) Aplica esta misma mejora al ejercicio 9.4.5 (la versión con dos paneles del programa "tipo Comandante Norton").

9.6. Datos sobre "el entorno"

La clase "Environment" nos sirve para acceder a información sobre el sistema: unidades de disco disponibles, directorio actual, versión del sistema operativo y de la plataforma .Net, nombre de usuario y máquina, carácter o caracteres que se usan para avanzar de línea, etc:

```
// Ejemplo_09_06a.cs
// Información sobre el sistema
// Introducción a C#, por Nacho Cabanes

using System;
using System.Diagnostics;

class Ejemplo_09_06a
{
    static void Main()
    {
        string avanceLinea = Environment.NewLine;
        Console.WriteLine("Directorio actual: {0}",
            Environment.CurrentDirectory);
        Console.WriteLine("Nombre de la máquina: {0}",
            Environment.MachineName);
        Console.WriteLine("Nombre de usuario: {0}", Environment.UserName);
        Console.WriteLine("Dominio: {0}", Environment.UserDomainName);
        Console.WriteLine("Código de salida del programa anterior: {0}",
            Environment.ExitCode);
        Console.WriteLine("Linea de comandos: {0}", Environment.CommandLine);
        Console.WriteLine("Versión del S.O.: {0}",
            Convert.ToString(Environment.OSVersion));
        Console.WriteLine("Version de .Net: {0}",
            Environment.Version.ToString());
        String[] discos = Environment.GetLogicalDrives();
            Console.WriteLine("Unidades lógicas: {0}",
                String.Join(", ", discos));
            Console.WriteLine("Carpeta de sistema: {0}",
                Environment.GetFolderPath(Environment.SpecialFolder.System));
    }
}
```

Ejercicios propuestos:

(9.6.1) Haz un programa que muestre el espacio disponible en todas las unidades de disco que tenga tu equipo, medido en GB. Además de "GetLogicalDrives", puedes necesitar usar la clase "DriveInfo", parecida a "DirectoryInfo", que tiene un

método estático "GetDrives" para obtener la lista de dispositivos, y propiedades como Name (nombre del dispositivo), AvailableFreeSpace (espacio disponible), TotalSize (espacio total) o IsReady (para saber si está disponible, que puede ser necesario en el caso de unidades extraíbles, como un lector de DVD).

(9.6.2) Mejora la versión más completa que hayas realizado del programa "tipo Comandante Norton", para que se pueda pasar a "navegar" por otra unidad de disco. Si se pulsa Alt+F1, se mostrará la lista de unidades lógicas, el usuario podrá escoger una de ellas, y esa pasará a ser la unidad activa en el panel izquierdo. Si se pulsa Alt+F2, se realizará el mismo proceso, pero cambiará la unidad activa en el panel derecho. Además, añade otra combinación de teclas (a tu criterio) que te muestre cierta información del sistema, como la que has visto en el fuente de ejemplo 09_06.

9.7. Algunos servicios de red.

Muchos de los servicios que podemos obtener de Internet o de una red local son accesibles de forma sencilla, típicamente enmascarados como si leyéramos de un fichero o escribiéramos en él.

(Nota: en este apartado se asumirá que el lector entiende algunos conceptos básicos de redes, como qué es una dirección IP, qué significa "localhost" o qué diferencias hay entre el protocolo TCP y el UDP).

Como primer ejemplo, vamos a ver cómo podríamos recibir una **página web** (por ejemplo, la página principal de "www.nachocabanes.com"), línea a línea como si se tratara de un fichero de texto (StreamReader), que se apoya en un "WebClient" y mostrar sólo las líneas que contengan un cierto texto (por ejemplo, la palabra "Pascal"):

```
// Ejemplo_09_07a.cs
// Ejemplo de descarga y análisis de una web:
//   Muestra las líneas que contienen "Pascal"
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;  // Para Stream
using System.Net; // Para System.Net.WebClient

class DescargarWeb
{

    static void Main()
    {
        WebClient cliente = new WebClient();
```

```
        Stream conexion = cliente.OpenRead("http://www.nachocabanes.com");
        StreamReader lector = new StreamReader(conexion);
        string linea;
        int contador = 0;
        while ( (linea=lector.ReadLine()) != null )
        {
            contador ++;
            if (linea.Contains("Pascal"))
                Console.WriteLine("{0}: {1}", contador, linea);
        }
        conexion.Close();
    }
}
```

El resultado de este programa sería algo como:

```
28:       <li><a href="pascal/index.php">Pascal/Delphi</a></li>
54:    | <a href="pascal/index.php">Pascal / Delphi</a>
204:     <li><a href="pascal/index.php">Pascal/Delphi</a></li>
```

Otra posibilidad relacionada con las redes y que tampoco es complicada (aunque sí algo más que ésta última) es la de **comunicar** dos ordenadores, para enviar información desde uno y recibirla desde el otro. Se puede hacer de varias formas. Una de ellas es usando directamente el protocolo TCP: emplearemos un TcpClient para enviar y un TcpListener para recibir, y en ambos casos trataremos los datos como un tipo especial de fichero binario, un NetworkStream:

```
// Ejemplo_09_07b.cs
// Ejemplo de envio y recepción de frases a través de la red
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;    // Para Stream
using System.Text;  // Para Encoding
using System.Net;   // Para Dns, IPAddress
using System.Net.Sockets;    // Para NetworkStream

class ComunicacionRed
{
    static string direccionPrueba = "localhost";
    static int puertoPrueba = 2112;

    private static void enviar(string direccion,
        int puerto, string frase)
    {
        TcpClient cliente = new TcpClient(direccion, puerto);
        NetworkStream conexion = cliente.GetStream();
        byte[] secuenciaLetras = Encoding.ASCII.GetBytes( frase );

        conexion.Write(secuenciaLetras, 0, secuenciaLetras.Length);

        conexion.Close();
        cliente.Close();
    }
```

```
    private static string esperar(string direccion, int puerto)
    {
        // Tratamos de hallar la primera IP que corresponde
        // a una dirección como "localhost"
        IPAddress direccionIP = Dns.GetHostEntry(direccion).AddressList[0];
        // Comienza la espera de información
        TcpListener listener = new TcpListener(direccionIP,puerto);
        listener.Start();
        TcpClient cliente = listener.AcceptTcpClient();
        NetworkStream conexion = cliente.GetStream();
        StreamReader lector = new StreamReader(conexion);

        string frase = lector.ReadToEnd();

        cliente.Close();
        listener.Stop();

        return frase;
    }

    static void Main()
    {
        Console.WriteLine("Pulse 1 para recibir o 2 para enviar");
        string respuesta = Console.ReadLine();

        if (respuesta == "2")  // Enviar
        {
            Console.Write("Enviando... ");
            enviar( direccionPrueba, puertoPrueba, "Prueba de texto");
            Console.WriteLine("Enviado");
        }
        else   // Recibir
        {
            Console.WriteLine("Esperando... ");
            Console.WriteLine( esperar(direccionPrueba, puertoPrueba) );
            Console.WriteLine("Recibido");
        }
    }
}
```

Cuando lanzáramos el programa, se nos preguntaría si queremos enviar o recibir:

```
Pulse 1 para recibir o 2 para enviar
```

Lo razonable es lanzar primero el proceso que espera para recibir, pulsando 1:

```
1
Esperando...
```

Entonces lanzaríamos otra sesión del mismo programa en el mismo ordenador (porque estamos conectando a la dirección "localhost"), y en esta nueva sesión escogeríamos la opción de Enviar (2):

```
Pulse 1 para recibir o 2 para enviar
2
```

```
Enviando...
```

Instantáneamente, en la primera sesión recibiríamos el texto que hemos enviado desde la segunda, y se mostraría en pantalla:

```
Prueba de texto
Recibido
```

Y la segunda sesión confirmaría que el envío ha sido correcto:

```
Enviando... Enviado
```

Esta misma idea se podría usar como base para programas más elaborados, que comunicaran diferentes equipos (en este caso, la dirección no sería "localhost", sino la IP del otro equipo), como podría ser un chat o cualquier juego multijugador en el que hubiera que avisar a otros jugadores de los cambios realizados por cada uno de ellos.

Esto se puede conseguir a un nivel algo más alto, usando los llamados "Sockets" en vez de los TcpClient, o de un modo "no fiable", usando el protocolo UDP en vez de TCP, pero nosotros no veremos más detalles de ninguno de ambos métodos.

Ejercicios propuestos:

(9.7.1) Crea un juego de "3 en raya" en red, para dos jugadores, en el que cada jugador escoja un movimiento en turnos alternativos, pero ambos vean un mismo estado del tablero.

(9.7.2) Crea un programa básico de chat, en el que dos usuarios puedan comunicarse, sin necesidad de seguir turnos estrictos.

(9.7.3) Crea un programa que monitorice cambios en una página web, comparando el contenido actual con una copia guardada en fichero. Deberá mostrar en pantalla un mensaje que avise al usuario de si hay cambios o no.

(9.7.4) Crea un programa que descargue todo un sitio web, partiendo de la página que indique el usuario, analizando los enlaces que contiene y descargando de forma recursiva las páginas que corresponden a dichos enlaces. Sólo deberá procesar los enlaces internos (en el mismo sitio web), no las páginas externas (alojadas en otros sitios web).

(9.7.5) Crea un juego de "barquitos" en red, para dos jugadores, en el que cada jugador decida dónde quiere colocar sus barcos, y luego cada uno de ellos "dispare" por turnos, escogiendo una casilla (por ejemplo, "B5"), y siendo avisado de si ha acertado ("impacto") o no ("agua"). Los barcos se podrán colocar en

horizontal o en vertical sobre un tablero de 8x8, y serán: 4 de longitud 1, 3 de longitud 2, 2 de longitud 3 y uno de longitud 4. Cada jugador verá el estado de su propio tablero y los impactos que ha conseguido en el tablero del otro jugador. El juego terminará cuando uno de los jugadores hunda toda la flota del otro.
(9.7.6) Mejora el juego de "barquitos" (9.7.5) para que el aviso de impacto sea más detallado ("tocado" o "hundido", según el caso).
(9.7.7) Mejora el juego de "barquitos" completo (9.7.6) para que un jugador pueda decidir aplazar la partida, y entonces el resultado quede guardado en fichero, y en la siguiente sesión se reanude el juego en el punto en el que quedó.

9.8. Contacto con LINQ

LINQ (Language-integrated Query, consulta integrada en el lenguaje) es una posibilidad añadida en la plataforma .Net versión 3.5, y accesible a partir de Visual Studio 2008, que nos permite hacer consultas a cualquier colección de datos usando una sintaxis que recuerda a la de SQL (aunque se escribe "casi al revés" que en el caso de SQL: primero el conjunto de datos, luego la condición y luego la variable a devolver).

Por ejemplo, podemos obtener los números enteros de un array cuyo valor es mayor que 10 con:

```
// Ejemplo_09_08a.cs
// Ejemplo básico de LINQ
// Introducción a C#, por Nacho Cabanes

using System;
using System.Linq;

class EjemploLinq1
{
    static void Main()
    {
        int[] datos = { 20, 5, 7, 4, 25, 18, 5, 8, 21, 2 };

        var result =
            from n in datos
            where n > 10
            select n;

        foreach(int i in result)
            Console.Write("{0} ", i);
        Console.WriteLine();
    }
}
```

También podemos recorrer la información de una lista de cadenas de texto y obtenerla ordenada, así:

```
// Ejemplo_09_08b.cs
// Segundo ejemplo de LINQ
// Introducción a C#, por Nacho Cabanes

using System;
using System.Linq;
using System.Collections.Generic;

class EjemploLinq2
{
    static void Main()
    {
        List<string> nombres = new List<string>
            { "pan", "carne", "queso", "manzana", "natillas" };

        var resultado =
            from nombre in nombres
            orderby nombre
            select nombre;

        foreach (string n in resultado)
            Console.Write("{0} ", n);
        Console.WriteLine();
    }
}
```

Profundizar en LINQ es algo que queda fuera del propósito de este texto. Si quieres saber más, puedes acudir a la propia referencia oficial en línea, en MSDN:

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

https://learn.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries

Ejercicios propuestos

(9.8.1) Crea un programa que pida al usuario varios números enteros, los guarde en una lista y luego muestre todos los que sean positivos, ordenados, empleando LINQ.

(9.8.2) Crea un programa que prepare un array de palabras. Luego, el usuario debe introducir una palabra, y el programa responderá si es palabra está en el array o no, utilizando LINQ.

9.9. Introducción a las expresiones regulares.

Las "expresiones regulares" permiten hacer comparaciones mucho más abstractas que si se usa un simple "Contains". Por ejemplo, podemos comprobar con una orden breve si todos los caracteres de una cadena son numéricos, o si empieza por mayúscula y el resto son minúsculas, etc.

Vamos a comenzar por un caso habitual: comprobar si una cadena es numérica, alfabética o alfanumérica. Las ideas básicas son:

- Usaremos el tipo de datos "RegEx" (expresión regular).
- Tenemos un método IsMatch, que devuelve "true" si una cadena de texto coincide con un cierto patrón.
- Uno de los patrones más habituales es indicar un rango de datos: [a-z] quiere decir "un carácter entre la a y la z".
- Podemos añadir modificadores: * para indicar "0 o más veces", + para "1 o más veces", ? para "0 o una vez". Así, [a-z]+, que quiere decir "uno o más caracteres entre la a y la z".
- Aun así, esa expresión puede dar resultados inesperados: un secuencia como [0-9]+ aceptaría cualquier cadena que contuviera una secuencia de números... aunque tuviera otros símbolos al principio y al final. Por eso, si queremos que sólo tenga cifras numéricas, nuestra expresión regular debería ser "inicio de cadena, cualquier secuencia de cifras, final de cadena", que se representaría como "\A[0-9]+\z". Una alternativa es plantear la expresión regular al contrario: "no es válido si contiene algo que no sea del 0 al 9", que se podría conseguir devolviendo lo contrario de lo que indique la expresión "[^0-9]".

Un ejemplo completo podría ser:

```
// Ejemplo_09_09a.cs
// Ejemplo de expresiones regulares
// Introducción a C#, por Nacho Cabanes

using System;
using System.Text.RegularExpressions;

class PruebaExprRegulares
{
    public static bool EsNumeroEntero(String cadena)
    {
        Regex patronNumerico = new Regex("[^0-9]");
        return !patronNumerico.IsMatch(cadena);
    }

    public static bool EsNumeroEntero2(String cadena)
    {
```

```
    Regex patronNumerico = new Regex(@"\A[0-9]*\z");
    return patronNumerico.IsMatch(cadena);
}

public static bool EsNumeroConDecimales(String cadena)
{
    Regex patronNumericoConDecimales =
        new Regex(@"\A[0-9]*,?[0-9]+\z");
    return patronNumericoConDecimales.IsMatch(cadena);
}

public static bool EsAlfabetico(String cadena)
{
    Regex patronAlfabetico = new Regex("[^a-zA-Z]");
    return !patronAlfabetico.IsMatch(cadena);
}

public static bool EsAlfanumerico(String cadena)
{
        Regex patronAlfanumerico = new Regex("[^a-zA-Z0-9]");
    return !patronAlfanumerico.IsMatch(cadena);
}



static void Main(string[] args)
{
    if (EsNumeroEntero("hola"))
        Console.WriteLine("hola es número entero");
    else
        Console.WriteLine("hola NO es número entero");

    if (EsNumeroEntero("1942"))
        Console.WriteLine("1942 es un número entero");
    else
        Console.WriteLine("1942 NO es un número entero");

    if (EsNumeroEntero2("1942"))
        Console.WriteLine("1942 es entero (forma 2)");
    else
        Console.WriteLine("1942 NO es entero (forma 2)");

    if (EsNumeroEntero("23,45"))
        Console.WriteLine("23,45 es un número entero");
    else
        Console.WriteLine("23,45 NO es un número entero");

    if (EsNumeroEntero2("23,45"))
        Console.WriteLine("23,45 es entero (forma 2)");
    else
        Console.WriteLine("23,45 NO es entero (forma 2)");

    if (EsNumeroConDecimales("23,45"))
        Console.WriteLine("23,45 es un número con decimales");
    else
        Console.WriteLine("23,45 NO es un número con decimales");

    if (EsNumeroConDecimales("23,45,67"))
        Console.WriteLine("23,45,67 es un número con decimales");
    else
```

```
            Console.WriteLine("23,45,67 NO es un número con decimales");

        if (EsAlfabetico("hola"))
            Console.WriteLine("hola es alfabetico");
        else
            Console.WriteLine("hola NO es alfabetico");

        if (EsAlfanumerico("hola1"))
            Console.WriteLine("hola1 es alfanumerico");
        else
            Console.WriteLine("hola1 NO es alfanumerico");
    }
}
```

Su salida es:

```
hola NO es número entero
1942 es un número entero
1942 es entero (forma 2)
23,45 NO es un número entero
23,45 NO es entero (forma 2)
23,45 es un número con decimales
23,45,67 NO es un número con decimales
hola es alfabetico
hola1 es alfanumerico
```

Las expresiones regulares son algo complejo. Por una parte, su sintaxis puede llegar a ser difícil de seguir. Por otra parte, el manejo en C# no se limita a buscar, sino que también permite otras operaciones, como reemplazar unas expresiones por otras. Como profundizar podría hacer este texto demasiado extenso, puede ser recomendable ampliar información usando la página oficial de Microsoft:

```
https://learn.microsoft.com/es-
es/dotnet/api/system.text.regularexpressions.regex?view=net-7.0
```

Ejercicios propuestos

(9.9.1) Crea una función que valide códigos postales españoles (5 cifras numéricas, la primera de 0 a 5 y el resto de 0 a 9): devolverá true si la cadena recibida como parámetro es un código postal válido.

(9.9.2) Crea una función que diga si una cadena que se le pase como parámetro parece un correo electrónico válido: tiene una @ precedida por al menos una letra, seguida por letras o números, un punto y más letras o números.