Sistemas distribuidos y de tiempo real

1.2 – E/S e hilos en .NET

Entrada – salida en .NET

- Clase Console del espacio de nombres System
- Físicamente representa
  - entrada = teclado, salida = ventana pantalla
- Métodos para escribir
  - Write(): Escribe los datos indicados en la salida estándar
  - WriteLine(): Escribe los datos indicados con ‘\n’ al final
- Métodos para leer
  - Read(): Lee un carácter de la entrada estándar
  - ReadLine(): Lee una línea de caracteres de la entrada estándar
- Ejemplos
  - c = Console.Read();
  - Console.WriteLine(“Hola mundo”);

Sobrecarga del método Write

- **(bool value)**
- **(__wchar_t value)**
- **(__wchar_t buffer __gc[])**
- **(__wchar_t buffer __gc[], int index, int count);**
- **(Decimal value)**
- **(double value)**
- **(int value)**
- **(__int64 value)**
- **(Object value)**
- **(float value)**
- **(String value)**
- **(unsigned int value)**
- **(String format, Object arg __gc[])**
- **(String format, Object arg0)**
- **(String format, Object arg0, Object arg1)**
- **(String format, Object arg0, Object arg1, Object arg2)**
- **(String format, Object arg0, Object arg1, Object arg2, Object arg3, …)**

Uso de cadenas de formato

Uso de cadenas de formato

- La única forma de escribir más de un dato en una sola llamada a Write es usando cadenas de formato
- El método Write escribe la cadena de formato en la consola sustituyendo cada **elemento de formato** que incluye por los valores de cada objeto correspondiente, por ejemplo:

```
String  Cadena = "Texto";
Console.WriteLine("La Cadena de caracteres es: {0}",Cadena);
```

- **{ Indice [,alineacion][:CadenaDeFormato] }**

Especificadores de formato

Índice

Identifica un elemento de la lista de valores
El especificador de formato con índice 0 da formato al primer valor de la lista, etc. …
Un valor que no es referenciado por un elemento de formato se ignora

Alineación

Indica el ancho mínimo para el campo y

Si es + → Alineación a Derecha
Si es – → Alineación a Izquierda

CadenaDeFormato (del especificador)

Incluye especificadores del formato estándar o personalizados

C ó c	Moneda (Currency)
D ó d	Decimal
E ó e	Exponencial ó Científico
F ó f	Punto Flotante
G ó g	General
N ó n	Numero
P ó p	Porcentaje
X ó x	Hexadecimal

Ejemplo de salida por consola

```
// Escribir una sola variable
bool b = true; Console.WriteLine(b);
float f = 12.34F; Console.WriteLine(f);
double d = -1.0; Console.WriteLine(d);
int i = 20; Console.WriteLine(i);
String Cadena = “texto"; Console.WriteLine(Cadena);

// Escribir una o mas variables con formato
Console.WriteLine("La Cadena de caracteres es: {0}",Cadena);
Console.WriteLine("El entero es: {0} ({0:X}hex) y el doble: {1}",i,d);
Console.WriteLine("El numero flotante es: {0}  {0:N}  {0:F}  {0:E}",f);
```

```
True
12,34
-1
20
texto
La Cadena de caracteres es: texto
El numero entero es: 20 (14hex) y el doble: -1
El numero flotante es: 12,34  12,34  12,34  1,234000E+001
```

La clase String

Un String es una colección secuencial de objetos System.Char que representan una cadena de caracteres

El valor de un String es el contenido de la colección secuencial y NO puede modificarse una vez creado

Los métodos de la clase String que aparentemente modifican un String devuelven en realidad un nuevo String que contiene la modificación

La clase String dispone de métodos para la manipulación de cadenas

Ejemplo:

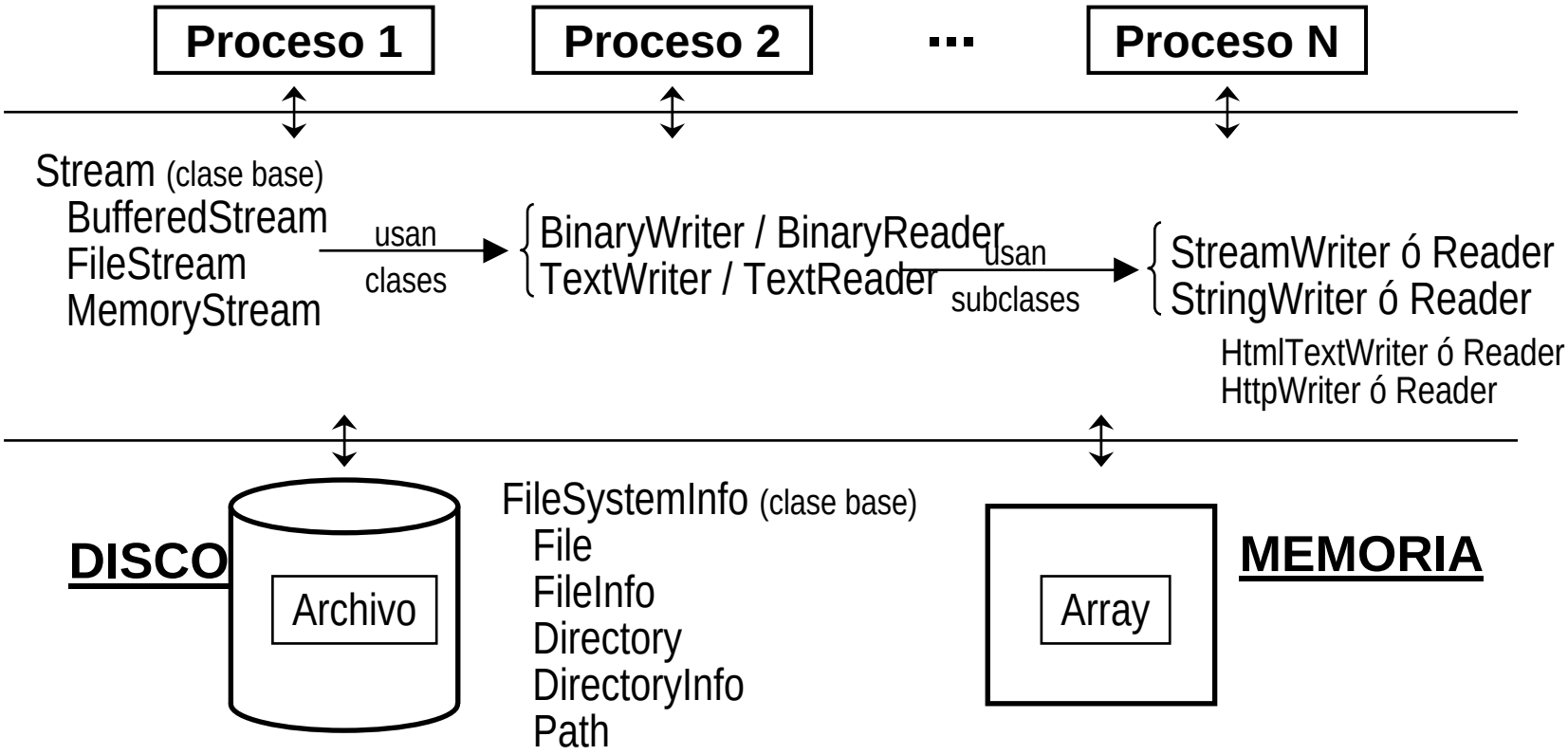
```
String Cad = “garcia, julio";

Cad = Cad.Substring(0,6);
```

Entrada / Salida en el entorno .NET

El espacio de nombres System.IO = clases que aporta .NET para realizar operaciones de E/S

Los procesos hacen la E/S sobre Secuencias (Streams) que a su vez la realizan sobre ficheros



Secuencias (Streams)

La clase base de todas las secuencias es Stream

Una secuencia, que es una instancia de la clase Stream, implementa una abstracción útil para manejar elementos que usan secuencias de bytes, como por ejemplo …

- Archivos
- Dispositivos de E/S
- Canales de comunicación entre procesos
- Sockets TCP/IP

La clase Stream y sus clases derivadas y asociadas proporcionan una vista genérica de diversos tipos de E/S aislando al programador de los detalles específicos del SO y sus dispositivos periféricos subyacentes

→ Binary(W/R) para escribir/leer tipos primitivos (Int32, Double) en/de un Stream directamente en binario

Los procesos usan las clases …

→Text(W/R) y sus derivadas Stream(W/R) y String(W/R) para escribir/leer caracteres en/de un Stream

Para los programadores de C → ¡ StreamWriter ← → fprintf()
StringWriter ← → sprintf() !

Archivos y Directorios

FileSystemInfo Contiene métodos y propiedades para manipular indistintamente archivos o directorios
Es la clase base de las clases FileInfo y DirectoryInfo

- **File** Contiene métodos ESTÁTICOS para manipular ficheros y para crear objetos FileStream
  Todos los métodos de File necesitan la ruta de acceso del archivo que están manipulando
- **FileInfo** Contiene métodos DE INSTANCIA para manipular ficheros y para crear objetos FileStream

- **Directory** Contiene métodos ESTÁTICOS para manipular directorios
  La mayoría de sus métodos necesitan la ruta de acceso del directorio que manipulan
- **DirectoryInfo** Contiene métodos DE INSTANCIA para manipular directorios

- **Path** Contiene métodos para manipular rutas de acceso a ficheros y directorios
  Todos sus miembros son ESTÁTICOS

Ejemplo de escritura en archivos (I)

```
using System;
using System.IO;

class archivo1
{

 static void Main(String []args)
{
    int NFIL = 10, NCOL = 5;
    int f, c;
    int [,] Tabla = new int[NFIL,NCOL];

   // Rellenar la matriz (tabla) de datos a almacenar
   for(f=0; f<NFIL; f++)
      for(c=0; c<NCOL; c++)
         Tabla[f,c]=100*f + c;

   // Ruta del fichero en el que se almacenan los datos
   String Ruta = "C:\\F1.TXT";
```

Ejemplo de escritura en archivos (II)

```
// (1) Crear directamente un Escritor de Secuencias para un fichero
 StreamWriter EscSec = new StreamWriter(Ruta);

// (2) Crear un objeto FileInfo para un fichero
// y luego un Escritor de Secuencias para el FileInfo
   FileInfo Fch = new FileInfo(Ruta);
   StreamWriter EscSec = Fch.CreateText();

// (3) Crear un objeto FileStream para un fichero
// y luego un Escritor de Secuencias para el FileStream
FileStream SecFch = new FileStream(Ruta, FileMode.Create, FileAccess.Write,
                                   FileShare.Write);
   StreamWriter EscSec = new StreamWriter(SecFch);
```

Utilizar solo uno de 1, 2, o 3

```
   for(f=0; f<NFIL; f++)
   {  for(c=0; c<NCOL; c++)
      {  EscSec.Write(" {0,3}\t",Tabla[f,c]);
         Console.Write(" {0,3}\t",Tabla[f,c]);
      }
      EscSec.WriteLine();
      Console.WriteLine();
   }
   EscSec.Close();
} }
```

Gestión del tiempo en .NET DateTime (I)

Las fechas y horas (el tiempo) se representan y manipulan usando 2 estructuras { DateTime, TimeSpan

DateTime

Representa un instante de tiempo, normalmente expresado en forma de fecha y hora del día

La estructura contiene el número de pulsos (ticks) desde …

las 12:00h de la noche del 1 de Enero del año 1 después de Cristo

Cada tick representa 100 nanosegundos 10^{-9} segundos

Declaración

```
DateTime Tini, Tfin;
```

Observar que las variables Tini y Tfin son tipos de valor

Inicialización

Se puede usar un constructor para inicializar una nueva instancia o una ya existente

```
Tini = DateTime(200);
DateTime Tx = DateTime(200);
```

} La fecha y hora se inicializa con 200 ticks

```
DateTime Ty = DateTime( año, mes, dia, hora, min, seg ,miliseg );
```

Enteros → 1-9999 1-12 1-28/31 0-23 0-59 0-59 0-999

Hay diversos constructores …

Gestión del tiempo en .NET DateTime (II)

Cargar la fecha y hora actual : Se utiliza la propiedad pública estática Now

Devuelve un DateTime con la fecha/hora actuales del computador en el que se ejecuta get_Now()

Recordatorio: Como la propiedad es estática se puede usar sin estar asociada a la instancia de un objeto concreto, basta con anteponerle el nombre de la clase

```
Tini = DateTime.Now;
for (int i=0; i<100000; i++);
Tfin = DateTime.Now;
```

← Computación cuya duración se debe medir

Visualización

La estructura DateTime dispone de multitud de propiedades que permiten obtener los componentes de una fecha y hora para visualizarlos o asignarlos a variables

```
Tini.Hour
Tini.Minute
Tini.Second
Tini.Millisecond
Tini.Ticks
```

→ Para obtener la máxima precisión

Gestión del tiempo en .NET TimeSpan (I)

TimeSpan

Representa un intervalo o período de tiempo

La estructura contiene el número de pulsos (ticks) que representan el período

La especificación de un número de ticks y el valor de un TimeSpan puede ser positivo o negativo

TimeSpan dispone de muchos métodos y propiedades que permiten manipular períodos e tiempo

```
TimeSpan PerTpo;
// Después de capturar 2 instantes de tiempo Tini y Tfin
PerTpo = Tfin –Tini;
Console.WriteLine(PerTpo.Ticks);
```

¡¡ Se puede restar directamente 2 estructuras DateTime y asignar el resultado a un TimeSpan !

PARECE QUE

NO es necesario realizar conversiones del período de tiempo a otros formatos, ya que TimeSpan dispone de propiedades que permiten acceder al valor del período en diferentes formatos

PERO ... →

Gestión del tiempo en .NET TimeSpan (II)

```
PerTpo.Ticks;
PerTpo.Milliseconds; → int
PerTpo.TotalMilliseconds; → double
PerTpo.Seconds; → int
PerTpo.TotalSeconds; → double
```

Componente de miliseg de una instancia (0-999)

Componente de segundos de una instancia (0-59)

NO son útiles para obtener el período total de tiempo transcurrido entre Tini y Tfin

TotalMilliseconds
TotalSeconds

SÍ son útiles, pero devuelven un tipo double

Si se desea almacenar el período de tiempo en un entero en unidades diferentes de ticks, hacer

```
PerTpo.Ticks / 10;        → Microsegundos
PerTpo.Ticks / 10000;     → Milisegundos
PerTpo.Ticks / 10000000;  → Segundos
```

Ejemplo: `5 1 3 7 3 8 7 2 ticks`

Componente de Segundos = 5
Componente de MiliSeg = 137

Pero …
Total de MiliSeg = 5137,3872

Generación de tiempos aleatorios (I)

En el espacio de nombres System se dispone de la clase Random que representa un dispositivo generador de números seudo-aleatorios con una distribución uniforme

Constructores

```
Random();
Random(int Semilla);
```

Método Next()

```
virtual int Next();
```
Devuelve un número aleatorio entero, NA tal que: 0 <= NA < MaxValue

```
virtual int Next(int Max);
```
Devuelve un número aleatorio entero, NA tal que: 0 <= NA < Max

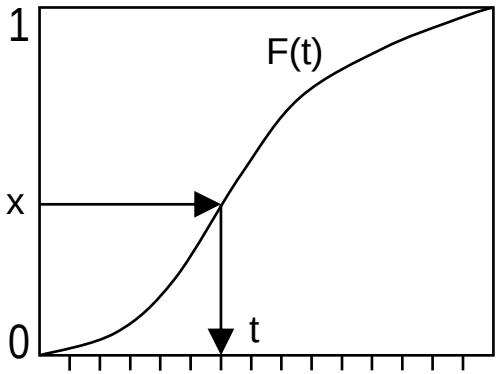
```
virtual int Next(int Min, int Max);
```
Devuelve un número aleatorio entero, NA tal que: Min <= NA < Max

Método NextDouble()

```
virtual double NextDouble();
```
Devuelve un número aleatorio doble, DA tal que: 0.0 <= DA < 1.0

Generación de tiempos aleatorios (II)

Para generar valores aleatorios siguiendo una determinada distribución de probabilidad usar la función de distribución acumulada de probabilidad F(t)



Seleccionar un valor aleatorio x de probabilidad acumulada entre 0 y 1

y obtener el valor de t correspondiente

Para algunas distribuciones el camino x → t puede hacerse analíticamente invirtiendo F(t)

En el caso de la distribución exponencial, F(t) = 1-Exp(-t / T) donde T = Media de la distribución

Despejando t … t = -T Ln(1-x)

La clase Math del ámbito System proporciona constantes y métodos estáticos para realizar operaciones trigonométricas, logarítmicas, etc. …

Ejemplo de Generación de tiempos aleatorios

```
using System;

class Aleatorios
{
static void Main (String [] args)
{   double NumAle, Tiempo, MediaExpo = 7.0;
    int i, Total = 1000;
//  int EntAle;
    Random Aleatorio = new Random(12345);

    for(i=0; i<Total; i++)
    {  NumAle = Aleatorio.NextDouble();
       // EntAle = Aleatorio.Next();
       Tiempo = -MediaExpo * Math.Log(1.0-NumAle);
       Console.WriteLine(" {0,6:0.0000}  {1,8:F}",NumAle,Tiempo);
    }
    return 0;
}
}
```

Delegados en hilos

Un delegado es una clase que permite llamar a uno o más métodos con el mismo prototipo

Un delegado equivale en .NET a un puntero a función en C++

Los delegados cada vez que se usan permiten:
- Llamar a una función, se derivan de la clase System.Delegate
- Llamar a una secuencia de funciones, derivan de la clase System.MulticastDelegate

Se utilizarán para especificar el método que constituye el cuerpo de un hilo

Subprocesos o Hilos (I)

El espacio de nombres **System.Threading** proporciona clases para la programación multihilo

Además dispone de clases para la sincronización de los hilos y el acceso a recursos compartidos

La clase Thread

Permite crear y controlar subprocesos administrados, establecer su prioridad, obtener su estado, …

Para crear un hilo administrado se usa un constructor

```
Thread(ThreadStart start);
```

delegado

```
ThreadStart(System.Object instancia, System.IntPtr metodo);
```

Nombre del objeto que contiene el método que actuará como cuerpo del hilo
Si el método es estático no es necesario especificar un objeto (poner 0=null)

Dirección del método que actúa como cuerpo del hilo

Uso típico del delegado y el constructor

```
ThreadStart HiloDelegado;
HiloDelegado = new ThreadStart(ClaseHilo.MetodoHilo);
Thread Hilo = new Thread(HiloDelegado);
```

Subprocesos o Hilos (II)

Hay dos métodos básicos en la clase Thread

El método Start()

Permite a un hilo (pj el principal) arrancar a otro hilo (pj uno auxiliar)

```
Hilo.Start();
```

El objeto Thread indica el hilo concreto que es arrancado

Método Join()

Permite a un hilo (pj el principal) esperar a que otro hilo (pj uno auxiliar) termine su ejecución

```
Hilo.Join();
```

El objeto Thread indica el hilo concreto por el que se espera

Sobrecargas del método

```
void Join()
bool Join(int MiliSegTimeout);
bool Join(TimeSpan Timeout);
```

Tiempo que se espera a que termine el subproceso

true = El subproceso ha terminado
false = El subproceso NO termino antes de transcurrir el tiempo especificado

Ejemplo de Hilos (I)

```
using System;
using System.Threading;

class ClaseHilo
{
     static void MetodoHilo()
     {  for(int i = 0; i < 10; i++)
        { Console.Write("MetodoHilo: ");
          Console.WriteLine(i);
          // Cede el resto del cuanto de tiempo de cpu
          Thread.Sleep(0);
        }
     }

static void Main(String [] args)
{ String Prioridad;
  // Crea el hilo pasando un delegado de ThreadStart
  ThreadStart DelegadoHilo = new ThreadStart(ClaseHilo.MetodoHilo);
  Thread HiloAux = new Thread(DelegadoHilo);
  Console.WriteLine("Hilo principal: Creado un hilo auxiliar");
```

Forma compacta de crear un hilo

Ejemplo de Hilos (II)

```
// Prioridad del hilo auxiliar creado
switch(HiloAux.Priority)
{ case ThreadPriority.Highest:      Prio = "Highest"; break;
  case ThreadPriority.AboveNormal:  Prio = "AboveNormal"; break;
  case ThreadPriority.Normal:       Prio = "Normal"; break;
  case ThreadPriority.BelowNormal:  Prio = "BelowNormal"; break;
  case ThreadPriority.Lowest:       Prio = "Lowest"; break;
}
Console.WriteLine("\nHilo principal: Prioridad del hilo auxiliar = {0}",Prioridad);

// Arrancar el hilo
HiloAux.Start();

// En un monoprocesador el nuevo hilo no obtiene la cpu hasta que la cede el hilo main

for(int i = 0; i < 4; i++)
{ Console.WriteLine("Hilo principal: Hace un trabajo");
  Thread.Sleep(0);
}

Console.WriteLine("Hilo principal: Espera que termine MetodoHilo");
HiloAux.Join();
Console.WriteLine("Hilo principal: Retorna de Join()");
}}
```