10. Persistencia de objetos

10.1. ¿Por qué la persistencia?

Para que la información utilizada por un programa esté disponible para una ejecución posterior, no sólo podemos emplear ficheros. Existen otras alternativas. Una de ellas es pedir al sistema que se conserve el estado actual de los objetos que forman el programa.

Podríamos imitar esta idea "de forma artesanal" si creamos un método que guarde cada uno de los atributos en un fichero y otro método que recupere cada uno de esos atributos desde el mismo fichero, como en este ejemplo:

```
// Ejemplo_10_01a.cs
// Primer acercamiento a la persistencia
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

public class Persist01
{

    int numero;

    public void SetNumero(int n)
    {
        numero = n;
    }

    public int GetNumero()
    {
        return numero;
    }

    public void Guardar(string nombre)
    {
        BinaryWriter ficheroSalida = new BinaryWriter(
            File.Open(nombre, FileMode.Create));
        ficheroSalida.Write(numero);
        ficheroSalida.Close();
    }

    public void Cargar(string nombre)
    {
        BinaryReader ficheroEntrada = new BinaryReader(
            File.Open(nombre, FileMode.Open));
        numero = ficheroEntrada.ReadInt32();
        ficheroEntrada.Close();
    }


    public static void Main()
```

```
    {
        // Preparamos el objeto y lo guardamos
        Persist01 ejemplo = new Persist01();
        ejemplo.SetNumero(5);
        Console.WriteLine("Valor: {0}", ejemplo.GetNumero());
        ejemplo.Guardar( "ejemplo.dat" );

        // Creamos un nuevo objeto
        Persist01 ejemplo2 = new Persist01();
        Console.WriteLine("Valor 2: {0}", ejemplo2.GetNumero());

        // Y cambiamos su valor, cargando los datos del primer objeto
        ejemplo2.Cargar( "ejemplo.dat" );
        Console.WriteLine("Y ahora: {0}", ejemplo2.GetNumero());
    }
}
```

Que daría como resultado:

```
Valor: 5
Valor 2: 0
Y ahora: 5
```

Pero esta forma de trabajar se complica mucho cuando nuestro objeto tiene muchos atributos, y más aún si se trata de bloques de objetos (por ejemplo, un "array" o una "lista") que a su vez contienen otros objetos. Por eso, existen maneras más automatizadas y que permiten escribir menos código. Las veremos a continuación.

Ejercicios propuestos:

(10.1.1) Amplía la clase Persona (ejercicio 6.2.1), para que permita guardar su estado y recuperarlo posteriormente.

10.2. Creando un objeto "serializable"

Vamos ver una forma más automática de guardar todo un objeto, incluyendo los valores actuales de sus atributos, en un fichero. En primer lugar, tendremos que añadir la etiqueta "[Serializable]" antes de la clase:

```
[Serializable]
public class Persist02
```

En segundo lugar, como vamos a sobrescribir todo el objeto, en vez de sólo los atributos, ahora los métodos "Cargar" y "Guardar" deberán estar en una clase auxiliar o al menos estar declarados como "static" para que puedan devolver todo un objeto (en el caso de Cargar) o recibir un objeto (en el caso de Guardar):

```
public static void Guardar(string nombre, Persist02 objeto)
```

El programa completo usará un FileStream como fichero subyacente y un "formateador" que se encargue de serializar los objetos en ese fichero. En los primeros ejemplos usaremos un formateador "binario", el más habitual. Un programa completo podría ser algo como

```
// Ejemplo_10_02a.cs
// Ejemplo básico de persistencia
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class Persist02
{

    int numero;

    public void SetNumero(int n)
    {
        numero = n;
    }

    public int GetNumero()
    {
        return numero;
    }

    // Métodos para guardar en fichero y leer desde él

    public static void Guardar(string nombre, Persist02 objeto)
    {
        IFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(nombre,
            FileMode.Create, FileAccess.Write, FileShare.None);
        formatter.Serialize(stream, objeto);
        stream.Close();
    }

    public static Persist02 Cargar(string nombre)
    {
        Persist02 objeto;
        IFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(nombre,
            FileMode.Open, FileAccess.Read, FileShare.Read);
        objeto = (Persist02)formatter.Deserialize(stream);
        stream.Close();
        return objeto;
    }


    public static void Main()
    {
```

```
        // Preparamos el objeto y lo guardamos
        Persist02 ejemplo = new Persist02();
        ejemplo.SetNumero(5);
        Console.WriteLine("Valor: {0}", ejemplo.GetNumero());
        Guardar("ejemplo.dat", ejemplo);

        // Creamos un nuevo objeto
        Persist02 ejemplo2 = new Persist02();
        Console.WriteLine("Valor 2: {0}", ejemplo2.GetNumero());

        // Y cambiamos su valor, cargando los datos del primer objeto
        ejemplo2 = Cargar("ejemplo.dat");
        Console.WriteLine("Y ahora: {0}", ejemplo2.GetNumero());
    }
}
```

Y su resultado sería el mismo que antes:

```
Valor: 5
Valor 2: 0
Y ahora: 5
```

Ejercicios propuestos:

(10.2.1) Crea una variante del ejercicio 10.1.1, que use serialización para guardar y recuperar los datos.

10.3. Precauciones con la persistencia

Puede parecer que la persistencia es una alternativa muy eficiente a guardar los datos de forma artesanal, pero no siempre es así: ciertos cambios en la clase pueden provocar que el fichero serializado no se pueda leer correctamente.

Como ejemplo, vamos a comenzar por ver un nuevo fuente en el que añadamos un atributo adicional y luego carguemos los datos desde fichero:

```
// Ejemplo_10_03a.cs
// Persistencia: intentar cargar desde un objeto con menos datos
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class Persist02
{

    int numero;
    double numero2;
```

```
    public void SetNumero(int n)
    {
        numero = n;
    }

    public int GetNumero()
    {
        return numero;
    }

    public void SetNumero2(double n)
    {
        numero2 = n;
    }

    public double GetNumero2()
    {
        return numero2;
    }

    // Métodos para guardar en fichero y leer desde él

    public static void Guardar(string nombre, Persist02 objeto)
    {
        IFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(nombre, Share.None);
        formatter.Serialize(stream, objeto);
        stream.Close();
    }

    public static Persist02 Cargar(string nombre)
    {
        Persist02 objeto;
        IFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(nombre,
            FileMode.Open, FileAccess.Read, FileShare.Read);
        objeto = (Persist02)formatter.Deserialize(stream);
        stream.Close();
        return objeto;
    }


    public static void Main()
    {
        Persist02 ejemplo = new Persist02();
        ejemplo.SetNumero(5);
        ejemplo.SetNumero2(6.3);
        Console.WriteLine("Numero: {0}", ejemplo.GetNumero());
        Console.WriteLine("Numero2: {0}", ejemplo.GetNumero2());

        ejemplo = Cargar("ejemplo2.dat");
        Console.WriteLine("Ahora numero: {0}", ejemplo.GetNumero());
        Console.WriteLine("Ahora numero2: {0}", ejemplo.GetNumero2());
    }
}
```

Al lanzar este programa es probable que obtengamos un mensaje de error como

```
Excepción no controlada: System.InvalidCastException: [A]Persist02 no puede
convertirse en [B]Persist02.
```

Eso también puede ocurrir si no cambiamos atributos, sino sólo métodos, como hace este programa:

```
// Ejemplo_10_03b.cs
// Persistencia: intentar cargar desde un objeto con menos métodos
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class Persist02
{

    int numero;

    public void SetNumero(int n)
    {
        numero = n;
    }

    public int GetNumero()
    {
        return numero;
    }

    public void MostrarNumero()
    {
        Console.Write(numero);
    }

    // Métodos para guardar en fichero y leer desde él

    public static void Guardar(string nombre, Persist02 objeto)
    {
        IFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(nombre,
            FileMode.Create, FileAccess.Write, FileShare.None);
        formatter.Serialize(stream, objeto);
        stream.Close();
    }

    public static Persist02 Cargar(string nombre)
    {
        Persist02 objeto;
        IFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(nombre,
            FileMode.Open, FileAccess.Read, FileShare.Read);
        objeto = (Persist02)formatter.Deserialize(stream);
        stream.Close();
        return objeto;
    }
```

```
    public static void Main()
    {
        Persist02 ejemplo = new Persist02();
        ejemplo.SetNumero(5);
        ejemplo.MostrarNumero();

        ejemplo = Cargar("ejemplo2.dat");
        ejemplo.MostrarNumero();
    }
}
```

Pero en este caso, si nos fijamos con detalle en el mensaje de error, veremos que es posible que se deba simplemente al **nombre** del fichero:

```
Excepción no controlada: System.InvalidCastException: [A]Persist02 no puede
convertirse en [B]Persist02. El tipo A se origina a partir de '10_02a,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null' en el contexto
'Default' en la ubicación [...]. El tipo B se origina a partir de '10_03b,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null' en el contexto
'Default' en la ubicación [...]'.
   en Persist02.Cargar(String nombre)
   en Persist02.Main()
```

Si tanto la clase como el fichero tienen el mismo nombre que aquellos que se emplearon para guardar los datos, es esperable que sí se comporte correctamente, y que cuando se carguen los datos desde fichero, los atributos nuevos (que no existían en la clase cuando se serializó por primera vez) tengan el valor por defecto (0 para los números, cadena vacía para los "string", etc). Éste es el resultado del ejemplo 10_03b tras renombrarlo a 10_03a:

```
Numero: 5
Numero2: 6,3
Ahora numero: 5
Ahora numero2: 0
```

En general, la clase a serializar será una de tantas de las que formen parte del proyecto, y es probable que "Main" esté en otra clase distinta. En ese caso, la llamada a "Cargar" y a "Guardar" deberá estar precedida por el nombre de la clase, por tratarse de métodos estáticos, como en el siguiente ejemplo, en el que se usan como "Ejemplo.Cargar" y "Ejemplo.Guardar":

```
// Ejemplo_10_03c.cs
// Clase serializable separada de la clase principal
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;
```

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class Ejemplo
{
    int numero;

    public void SetNumero(int n)
    {
        numero = n;
    }

    public int GetNumero()
    {
        return numero;
    }

    // Métodos para guardar en fichero y leer desde él

    public static void Guardar(string nombre, Ejemplo objeto)
    {
        IFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(nombre,
            FileMode.Create, FileAccess.Write, FileShare.None);
        formatter.Serialize(stream, objeto);
        stream.Close();
    }

    public static Ejemplo Cargar(string nombre)
    {
        Ejemplo objeto;
        IFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(nombre,
            FileMode.Open, FileAccess.Read, FileShare.Read);
        objeto = (Ejemplo)formatter.Deserialize(stream);
        stream.Close();
        return objeto;
    }
}

public class PruebaDeEjemplo
{
    public static void Main()
    {
        Ejemplo ejemplo = new Ejemplo();
        ejemplo.SetNumero(5);
        Console.WriteLine("Valor: {0}", ejemplo.GetNumero());
        Ejemplo.Guardar("ejemplo3.dat", ejemplo);

        Ejemplo ejemplo2 = new Ejemplo();
        Console.WriteLine("Valor 2: {0}", ejemplo2.GetNumero());
        ejemplo2 = Ejemplo.Cargar("ejemplo3.dat");
        Console.WriteLine("Y ahora: {0}", ejemplo2.GetNumero());
    }
}
```

Ejercicios propuestos:

(10.3.1) Amplía el ejercicio 10.2.1, añadiendo algún atributo y/o método adicional a la clase persona y comprobando que se puede cargar correctamente.

(10.3.2) En el ejercicio 8.3.4 habías ampliado la "base de datos de ficheros", de modo que los datos se leyeran desde fichero. Crea una versión alternativa que emplee serialización.

10.4. Volcando a un fichero de texto

Hasta ahora, hemos estamos serializando los datos a un fichero binario. Si queremos guardarlos en un fichero XML (por ejemplo para maximizar la portabilidad, garantizando que se va a leer correctamente desde algún otro sistema en el que quizá el orden de los bytes sea distinto), los cambios son mínimos: cambiar "BinaryFormatter" por "SoapFormatter", así como el correspondiente "using" y quizá añadir una DLL adicional al proyecto (un poco más adelante detallaremos cómo):

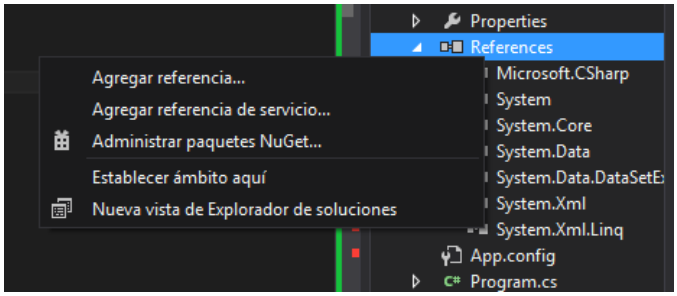
```
// Ejemplo_10_04a.cs
// Ejemplo de persistencia (XML)
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;
using System.Runtime.Serialization;
// Para la siguiente línea, puede ser necesario añadir a las referencias
// del proyecto "System.Runtime.Serialization.Formatters.Soap.dll"
using System.Runtime.Serialization.Formatters.Soap;

// ------------------------------------------
// Las clases "de prueba"

[Serializable]
public class MiniEjemplo
{
    int dato;

    public void SetDato(int n)
    {
        dato = n;
    }

    public int GetDato()
    {
        return dato;
    }
}

[Serializable]
public class Ejemplo
{

    int numero;
```

```
    float numero2;
    MiniEjemplo[] mini;

    public Ejemplo()
    {
        mini = new MiniEjemplo[100];
        for (int i=0; i<100; i++)
        {
            mini[i] = new MiniEjemplo();
            mini[i].SetDato( i*2 );
        }
    }

    public void SetNumero(int n)
    {
        numero = n;
    }

    public int GetNumero()
    {
        return numero;
    }

    public void SetMini(int n, int valor)
    {
        mini[n].SetDato(valor);
    }

    public int GetMini(int n)
    {
        return mini[n].GetDato();
    }

}


// ------------------------------------------
// Una clase adicional "encargada de guardar"

public class Serializador
{
    string nombre;

    public Serializador(string nombreFich)
    {
        nombre = nombreFich;
    }

    public void Guardar(Ejemplo objeto)
    {
        IFormatter formatter = new SoapFormatter();
        Stream stream = new FileStream(nombre,
            FileMode.Create, FileAccess.Write, FileShare.None);
        formatter.Serialize(stream, objeto);
        stream.Close();
    }

    public Ejemplo Cargar()
    {
        Ejemplo objeto;
```

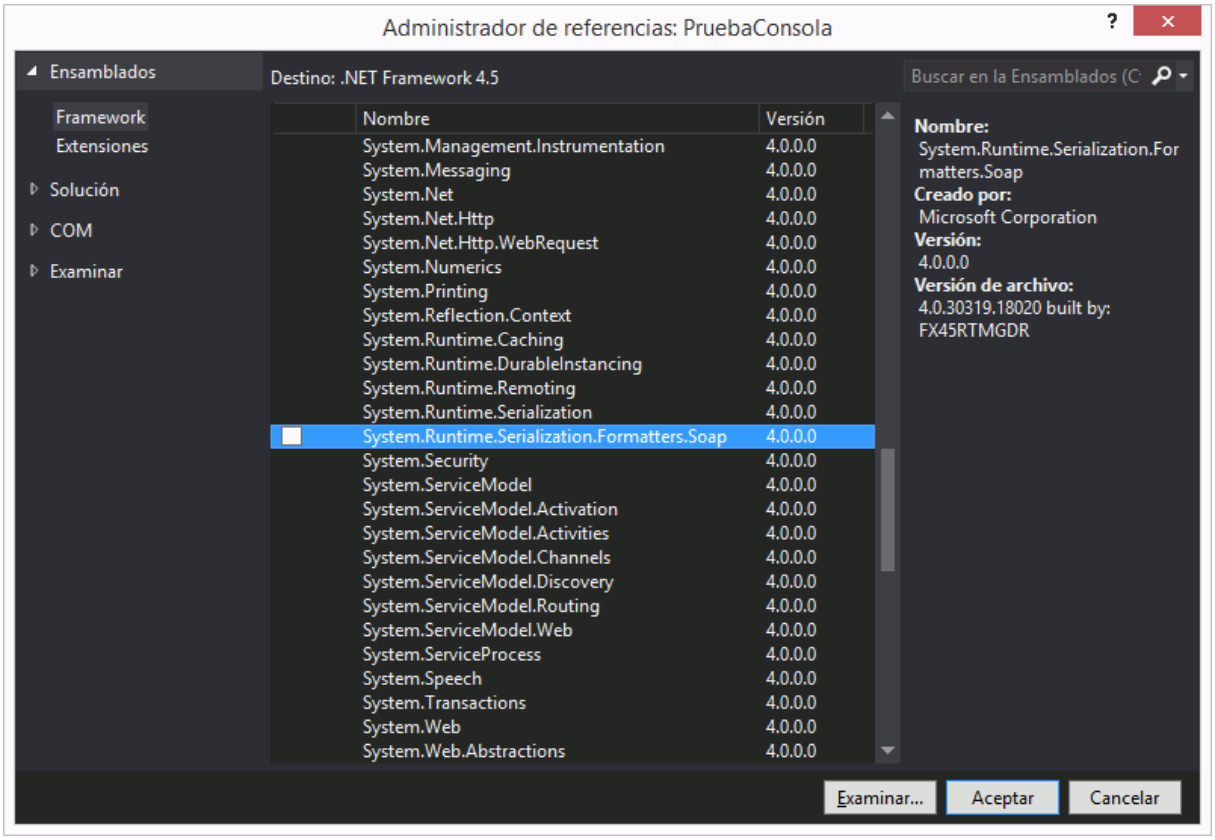
```
        IFormatter formatter = new SoapFormatter();
        Stream stream = new FileStream(nombre,
            FileMode.Open, FileAccess.Read, FileShare.Read);
        objeto = (Ejemplo)formatter.Deserialize(stream);
        stream.Close();
        return objeto;
    }
}


// -------------------------------------------
// Y el programa de prueba

public class Prueba
{
    public static void Main()
    {
        Ejemplo ejemplo = new Ejemplo();
        ejemplo.SetNumero(5);
        ejemplo.SetMini(50, 500);
        Console.WriteLine("Valor: {0}", ejemplo.GetNumero());

        Serializador s = new Serializador("ejemplo2.dat");
        s.Guardar(ejemplo);

        Ejemplo ejemplo2 = new Ejemplo();
        Console.WriteLine("Valor 2: {0}", ejemplo2.GetNumero());
        ejemplo2 = s.Cargar();
        Console.WriteLine("Y ahora: {0}", ejemplo2.GetNumero());
        Console.WriteLine("dato 50: {0}", ejemplo2.GetMini(50));
    }
}
```

Como hemos comentado, este programa, que compilará correctamente en versiones de .Net como la 2.0, puede no compilar tal cual en versiones más recientes, como la 4. Si es el caso, deberemos añadir una "referencia" adicional al proyecto.

Si usamos Visual Studio, deberemos ir al panel derecho (Explorador de soluciones), hacer clic con el botón derecho en "References" e indicar que deseamos "Agregar referencia":



Dentro de esta opción, deberemos buscar dentro de las referencias que ya están previstas para nuestra versión de .Net (por ejemplo, la 4.5) el nombre "System.Runtime.Serialization.Formatters.Soap" y después aceptar los cambios:



Si usamos XML (SoapFormatter), el fichero de datos resultante será de mayor tamaño, pero a cambio se podrá analizar con mayor cantidad de herramientas, al ser texto puro. Por ejemplo, este es un fragmento del fichero de datos generado por el programa anterior:

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<a1:ClaseAGuardar id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/assem/persist05%2C%20Version%3D0.0
.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<e href="#ref-3"/>
</a1:ClaseAGuardar>
<a1:Ejemplo id="ref-3"
xmlns:a1="http://schemas.microsoft.com/clr/assem/persist05%2C%20Version%3D0.0
.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<numero>5</numero>
```

```
<numero2>0</numero2>
<mini href="#ref-4"/>
</a1:Ejemplo>
<SOAP-ENC:Array id="ref-4" SOAP-ENC:arrayType="a1:MiniEjemplo[100]"
xmlns:a1="http://schemas.microsoft.com/clr/assem/persist05%2C%20Version%3D0.0
.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<item href="#ref-5"/>
<item href="#ref-6"/>
...
<item href="#ref-103"/>
<item href="#ref-104"/>
</SOAP-ENC:Array>
<a1:MiniEjemplo id="ref-5"
xmlns:a1="http://schemas.microsoft.com/clr/assem/persist05%2C%20Version%3D0.0
.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<dato>0</dato>
</a1:MiniEjemplo>
<a1:MiniEjemplo id="ref-6"
xmlns:a1="http://schemas.microsoft.com/clr/assem/persist05%2C%20Version%3D0.0
.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<dato>2</dato>
</a1:MiniEjemplo>
...

<a1:MiniEjemplo id="ref-104"
xmlns:a1="http://schemas.microsoft.com/clr/assem/persist05%2C%20Version%3D0.0
.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<dato>198</dato>
</a1:MiniEjemplo>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(Se ha omitido la mayor parte de los datos repetitivos)

Como curiosidad, SOAP (que da su nombre al SoapFormatter) es un protocolo diseñado para poder intercambiar de forma sencilla información estructurada (como objetos y sus componentes) en aplicaciones Web.

Ejercicios propuestos:

(10.4.1) Crea una variante del ejercicio 10.3.1, que guarde los datos en formato XML.

(10.4.2) Crea una versión ampliada de la "base de datos de ficheros" (ejemplo 04_06a) que use objetos en vez de struct, y que guarde los datos (en formato XML) usando persistencia.

(10.4.3) Añade al ejercicio de los trabajadores (6.8.1) la posibilidad de guardar sus datos en formato XML.