```
      Console.WriteLine("El int leido es un {0}",
        unDatoInt);
      unDatoFloat = ficheroEntrada.ReadSingle();
      Console.WriteLine("El float leido es un {0}",
        unDatoFloat);
      unDatoDouble = ficheroEntrada.ReadDouble();
      Console.WriteLine("El double leido es un {0}",
        unDatoDouble);
      unDatoString = ficheroEntrada.ReadString();
      Console.WriteLine("El string leido es \"{0}\"",
        unDatoString);
      Console.WriteLine("Volvamos a leer el int...");
      ficheroEntrada.BaseStream.Seek(1, SeekOrigin.Begin);
      unDatoInt = ficheroEntrada.ReadInt32();
      Console.WriteLine("El int leido es un {0}",
        unDatoInt);
      ficheroEntrada.Close();
    }
    catch (Exception exp)
    {
      Console.WriteLine(exp.Message);
      return;
    }
  }

}
```

Como se puede ver en este ejemplo, también podemos usar "Seek" para movernos a un punto u otro de un fichero si usamos un "BinaryReader", pero está un poco más escondido: no se lo pedimos directamente a nuestro fichero, sino al "Stream" (flujo de datos) que hay por debajo: `ficheroEntrada.BaseStream.Seek(1, SeekOrigin.Begin);`

El resultado de este programa es:

```
Introduzca el nombre del fichero a crear: 1234
Creando fichero...
Leyendo fichero...
El byte leido es un 1
El int leido es un 2
El float leido es un 3
El double leido es un 4
El string leido es "Hola"
Volvamos a leer el int...
El int leido es un 2
```

En este caso hemos usado "FileMode.Create" para indicar que queremos crear el fichero, en vez de abrir un fichero ya existente. Los modos de fichero que podemos emplear en un Binary-Reader o en un BinaryWriter son los siguientes:

- CreateNew: Crear un archivo nuevo. Si existe, se produce una excepción IOException.
- Create: Crear un archivo nuevo. Si ya existe, se sobrescribirá.
- Open: Abrir un archivo existente. Si el archivo no existe, se produce una excepción System.IO.FileNotFoundException.

```
    public static void Main()
    {
        Persist01 ejemplo = new Persist01();
        ejemplo.SetNumero(5);
        Console.WriteLine("Valor: {0}", ejemplo.GetNumero());
        ejemplo.Guardar( "ejemplo.dat" );

        Persist01 ejemplo2 = new Persist01();
        Console.WriteLine("Valor 2: {0}", ejemplo2.GetNumero());
        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") que a su vez contienen otros objetos. Por eso, existen maneras más automatizadas y que permiten escribir menos código.

Ejercicios propuestos:

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

9.2. Creando un objeto "serializable"

Vamos a guardar todo un objeto (incluyendo los valores de sus atributos) en un fichero. En primer lugar, tendremos que indicar "[Serializable]" antes de la clase:

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

En segundo lugar, como vamos a sobreescribir todo el objeto, en vez de sólo los atributos, ahora los métodos "Cargar" y "Guardar" ya no pueden pertenecer a esa misma clase: deberán estar en una clase auxiliar, que se encargue de salvar los datos y recuperarlos. En este primer ejemplo, nos limitaremos a declararlos "static" para que sea Main el que se encargue de esas tareas:

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

El programa completo podría ser algo como

```
/*--------------------------*/
/*  Ejemplo en C#           */
/*  persist02.cs            */
```

```
/*                          */
/*  Ejemplo básico de       */
/*  persistencia            */
/*                          */
/*  Introduccion a C#,      */
/*    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();
        Stream 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();
        Stream 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);
        Console.WriteLine("Valor: {0}", ejemplo.GetNumero());
        Guardar("ejemplo.dat", ejemplo);
```

```
        Persist02 ejemplo2 = new Persist02();
        Console.WriteLine("Valor 2: {0}", ejemplo2.GetNumero());
        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:

- **(9.2.1)** Crea una variante del ejercicio 9.1.1, que use serialización para guardar y recuperar los datos.

9.3. Empleando clases auxiliares

Una solución un poco más elegante podría ser encapsular la clase (o clases) que vamos a guardar, incluyéndola(s) dentro de otra clase auxiliar:

```
[Serializable]
public class ClaseAGuardar
{
    Ejemplo e;

    public void SetDatos(Ejemplo e1)
    {
        e = e1;
    }

    public Ejemplo GetDatos()
    {
        return e;
    }
}
```

Y crear una segunda clase, que sea la encargada de guardar y recuperar los datos:

```
public class Serializador
{
    string nombre;

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

    public void Guardar(ClaseAGuardar objeto)
    {
        IFormatter formatter = new BinaryFormatter();
```

```
        Stream stream = new FileStream(nombre,
            FileMode.Create, FileAccess.Write, FileShare.None);
        formatter.Serialize(stream, objeto);
        stream.Close();
    }

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

De modo que un único fuente que contuviera estas tres clases podría ser:

```
/*---------------------------*/
/*  Ejemplo en C#            */
/*  persist03.cs             */
/*                           */
/*  Ejemplo de               */
/*  persistencia             */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

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

// ------------------------------------------
// La clase "de prueba"

[Serializable]
public class Ejemplo
{

    int numero;

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

    public int GetNumero()
    {
        return numero;
    }
}

// ------------------------------------------
```

```
// La clase "que realmente se va a guardar"

[Serializable]
public class ClaseAGuardar
{
    Ejemplo e;

    public void SetDatos(Ejemplo e1)
    {
        e = e1;
    }

    public Ejemplo GetDatos()
    {
        return e;
    }
}


// -----------------------------------------
// La clase "encargada de guardar"

public class Serializador
{
    string nombre;

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

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

    public ClaseAGuardar Cargar()
    {
        ClaseAGuardar objeto;
        IFormatter formatter = new BinaryFormatter();
        Stream stream = new FileStream(nombre,
            FileMode.Open, FileAccess.Read, FileShare.Read);
        objeto = (ClaseAGuardar)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);
        Console.WriteLine("Valor: {0}", ejemplo.GetNumero());

        ClaseAGuardar guardador = new ClaseAGuardar();
        guardador.SetDatos(ejemplo);

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

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

Vamos a comprobar que se comporta correctamente en un caso más complicado, haciendo que nuestra clase contenga varios atributos, entre ellos un array de objetos de otra clase. La clase "Serializador" no cambiará, y en un caso real apenas cambiaría la clase "ClaseAGuardar" (que aquí tampoco va a modificarse):

```
/*--------------------------*/
/*  Ejemplo en C#           */
/*  persist04.cs            */
/*                          */
/*  Ejemplo de              */
/*  persistencia            */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

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

// -----------------------------------------
// Las dos clases "de prueba": una contiene a otra

[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();
    }

}

// ------------------------------------------
// La clase "que realmente se va a guardar"

[Serializable]
public class ClaseAGuardar
{
    Ejemplo e;

    public void SetDatos(Ejemplo e1)
    {
        e = e1;
    }

    public Ejemplo GetDatos()
```

```
    {
        return e;
    }
}


// -------------------------------------------
// La clase "encargada de guardar"

public class Serializador
{
    string nombre;

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

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

    public ClaseAGuardar Cargar()
    {
        ClaseAGuardar objeto;
        IFormatter formatter = new BinaryFormatter();
        Stream stream = new FileStream(nombre,
            FileMode.Open, FileAccess.Read, FileShare.Read);
        objeto = (ClaseAGuardar)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());

        ClaseAGuardar guardador = new ClaseAGuardar();
        guardador.SetDatos(ejemplo);

        Serializador s = new Serializador("ejemplo.dat");
```

```
        s.Guardar(guardador);

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

Esto daría como resultado:

```
Valor: 5
Valor 2: 0
Y ahora: 5
dato 50: 500
```

Ejercicios propuestos:

- **(9.3.1)** Crea una variante del ejercicio 9.2.1, que use una clase auxiliar para la serialización.

9.4. Volcando a un fichero de texto

La forma de trabajar que estamos empleando guarda los datos en 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":

```
/*--------------------------*/
/*  Ejemplo en C#           */
/*  persist05.cs            */
/*                          */
/*  Ejemplo de              */
/*  persistencia (XML)      */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;
using System.IO;
using System.Runtime.Serialization;
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();
    }

}

// -------------------------------------------
// La clase "que realmente se va a guardar"

[Serializable]
public class ClaseAGuardar
{
    Ejemplo e;
```

```
    public void SetDatos(Ejemplo e1)
    {
        e = e1;
    }

    public Ejemplo GetDatos()
    {
        return e;
    }
}


// ------------------------------------------
// La clase "encargada de guardar"

public class Serializador
{
    string nombre;

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

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

    public ClaseAGuardar Cargar()
    {
        ClaseAGuardar objeto;
        IFormatter formatter = new SoapFormatter();
        Stream stream = new FileStream(nombre,
            FileMode.Open, FileAccess.Read, FileShare.Read);
        objeto = (ClaseAGuardar)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());

        ClaseAGuardar guardador = new ClaseAGuardar();
        guardador.SetDatos(ejemplo);

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

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

El fichero 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-7"/>
...

<item href="#ref-102"/>
<item href="#ref-103"/>
<item href="#ref-104"/>
</SOAP-ENC:Array>
```