Autenticación de usuarios mediante contraseñas
Práctica 9

1. Objetivo

En esta práctica el alumno debe aprender a crear un fichero de usuarios y contraseñas (**en el que NUNCA deben guardarse las contraseñas directamente**) y luego a verificar el usuario y la contraseña proporcionados a un programa usando los usuarios y contraseñas almacenados en el fichero.

2. Creación del fichero de contraseñas

Crea una solución de VS para generar un fichero de contraseñas. Se puede basar en dos clases.

La clase pública User define los datos que constituyen cada uno de los registros del fichero de contraseñas. Cada registro contendrá tres datos de un mismo usuario: el nombre, el Salt y el resumen de la contraseña elegida por el usuario.

En la clase **User** definir el array de caracteres **Nombre**, y los arrays de bytes **Salt** y **ResuContra** (Resumen de la Contraseña). Conviene definir previamente tres constantes enteras privadas para definir la longitud de estos arrays, de modo que queden bien documentadas las longitudes usadas. Por ejemplo, se puede usar 16 caracteres para el Nombre, 16 bytes para el Salt y 32 bytes para el resumen. Estos valores permiten "ver" cómodamente el fichero binario con el programa HxD.

Después, definir el método User() que será el constructor de la clase y que tendrá dos parámetros de tipo string: **NuevoNombre** y **NuevaContra**. Se asume que la longitud (en caracteres) del string **NuevoNombre** es menor que la del array de caracteres **Nombre**. Este será el único método de la clase, y en él, se realizará la secuencia de operaciones siguiente:

1) Dar valor a **Nombre** copiando carácter a carácter, el string **NuevoNombre** al array **Nombre**. Si la longitud de **NuevoNombre** es menor que la de **Nombre** hay que decidir cómo se rellena la parte final de **Nombre**: (1) con ceros ó (2) con espacios en blanco.

El entorno de ejecución, al crear un array de caracteres, suele rellenarlo de ceros. En este caso no hay que hacer nada para usar la primera opción. Pero si queremos estar completamente seguros de cómo se almacenan los nombres de los usuarios, lo mejor es inicializar todo el array de caracteres **Nombre** con el carácter deseado para la parte final, cero o espacio, y luego copiar el string **NuevoNombre** en **Nombre**, carácter a carácter.

2) Para generar el **Salt**, crear un proveedor de números aleatorios usando la clase **RNGCryptoServiceProvider** y utilizar su método **GetBytes()**.

3) Generar el resumen de la contraseña, usando un objeto de la clase **Rfc2898DeriveBytes**, al que se le pasan como parámetros de entrada, **NuevaContra** y **Salt**. Asignar la propiedad **IterationCount** al valor 1000, que es el valor por defecto. Posteriormente, se podrá modificar este valor para hacer pruebas. Invoca al método **GetBytes()** del objeto, que devuelve **ResuContra**, pasándole al método la longitud de ResuContra.

La clase pública ListaU define un array unidimensional (vector) de objetos User y proporciona métodos para: inicializar los datos de los usuarios, ver la lista de usuarios y guardar la lista en un fichero (dos versiones: en binario y en texto).

En la clase **ListaU** definir la constante entera **MaxUsu** para establecer el tamaño de la lista y darle un valor bajo, por ejemplo 5. Declarar **Lista** como un array de User y crearla con el operador new con el tamaño definido por MaxUsu. Después definir los métodos de la clase.

Definir el método **IniUsu**(int **Indice**, User **Usuario**), que permite asociar un objeto **Usuario** a un **Indice** cualquiera del array Lista.

Definir el método **VerLista()** que muestra el nombre de todos los usuarios que contiene la Lista en la consola.

Definir el método **GuardaListaBin**(string **NombreFich**), que almacena los tres datos de cada usuario (Nombre, Salt, ResuContra) en el fichero indicado, por ejemplo zz_Usuarios.bin. Este método debe guardar los datos en formato binario. El método utilizará la clase **BinaryWriter** con un **Encoding** para los caracteres de tipo **Unicode**.

Definir el método **GuardaListaTxt**(string **NombreFich**), que también almacena los tres datos de cada usuario en el fichero indicado, por ejemplo zz_Usuarios.txt.

PERO Desarrollar este método GuardaListaTxt() después de la sección 3 y antes de la 4.

Este método debe guardar los datos en formato de texto. El método utilizará la clase **StreamWriter** con un **Encoding** para los caracteres de tipo **ASCII**. Los datos que son intrínsecamente binarios como Salt y ResuContra hay que traducirlos a una representación textual. Informalmente, se puede almacenar en formato texto los dos dígitos hexadecimales que representan a cada byte. Más formalmente, se debe codificar cada dato en formato Base64 usando el método **ToBase64String()** de la clase **Convert**.

El método Main() debe crear un objeto LU de la clase ListaU y luego inicializar 5 usuarios llamando al método IniUsu() de LU, por ejemplo, del modo siguiente:

```
LU.IniUsu(0, new User("Antonio", "conA"));

LU.IniUsu(1, new User("Benito", "conB"));
```

Continuar introduciendo Carlos - conC, David - conD y Eduardo - conE. Así es muy fácil acordarse de los usuarios y sus contraseñas.

Posteriormente, se puede llamar al método VerLista() para comprobar que todos los usuarios están en la Lista.

Finalmente, se llama a los métodos GuardaListaBin() y GuardaListaTxt() para almacenar la lista en ficheros.

3. Verificación de usuario y contraseña (Fichero Binario)

Crea una **nueva solución de Visual Studio** para verificar un usuario y su contraseña que se basará en una clase, en la que se encapsula todo el código necesario.

La clase UsuContra comenzará definiendo cuatro constantes enteras privadas con el máximo número de usuarios contenido en el fichero de usuarios y contraseñas, MaxUsu (5), la longitud del Nombre (16 caracteres), la longitud del Salt (16 bytes) y la longitud del resumen (32 bytes).

Después, definir tres arrays, para cargar un registro del fichero: Nombre, Salt y ResuContra, del tamaño definido por las constantes enteras declaradas previamente.

Definir el método **Verifica**(string **NombreIn**, string **ContraIn**) que devuelve un entero. **NombreIn** y **ContraIn** contienen el nombre del usuario y su contraseña introducidos por el usuario y que hay que verificar. El entero devuelto será 0 si **NombreIn** y **ContraIn** son válidos. Será 1 si **NombreIn** es desconocido y 2 si **ContraIn** es inválida.

La secuencia de operaciones a realizar en el método es la siguiente:

1) Copiar el string **NombreIn** en un array de caracteres del mismo tamaño que los nombres almacenados en el fichero de contraseñas. Si la longitud de **NombreIn** es menor que la del array de caracteres, completar el array con el mismo carácter usado para completar los nombres en el fichero de usuarios/contraseñas: ceros o espacios en blanco.

2) Buscar el NombreIn en el fichero de contraseñas. Para leer el fichero usar un **FileStream** y un **BinaryReader** con un **Encoding** para los caracteres de tipo **Unicode**. Implementar un bucle do{}while() en el que, en cada iteración, se lee un registro de un usuario (Nombre, Salt, ResuContra). Se comparan dos arrays de caracteres, el leído y el preparado en la fase 1). Desarrollar un método auxiliar para realizar la comparación de arrays de caracteres que devuelva 'true' si los arrays son iguales. Usar un contador que se incremente con cada registro leído. El bucle termina cuando se encuentra el registro del usuario buscado o se llega al final del fichero de contraseñas.

3) Cerrar el BinaryReader y después el FileStream.

4) Si NO se ha encontrado el nombre del usuario en el fichero, retornar del método con el valor 1.

5) Calcular el resumen de la contraseña, a partir de la contraseña introducida, ContraIn, y del Salt obtenido del fichero. Para ello declarar un objeto de la clase **Rfc2898DeriveBytes** y asignar a la propiedad **IterationCount** el mismo valor que se utilizó previamente para crear el fichero de contraseñas. LLamar al método **GetBytes()** para obtener el resumen de la contraseña introducida por el usuario.

6) Comparar el resumen calculado en la fase 5) con el resumen leído del fichero de contraseñas. Desarrollar un método auxiliar para realizar la comparación de arrays de bytes que devuelva 'true' si los arrays son iguales. Si NO son iguales, retornar del método con el valor 2.

7) La última sentencia retorna del método con el valor 0, indicando que se ha encontrado el nombre del usuario en el fichero y que la contraseña introducida coincide con la almacenada en el fichero.

El método Main() debe leer de la consola un nombre de usuario y su contraseña, por ejemplo utilizando **Console.ReadLine()**. Después, crear un objeto UC de la clase **UsuContra**, e invocar el método Verifica(). Finalmente, utilizar una sentencia **switch** para mostrar en la consola un mensaje para cada valor devuelto por el método Verifica().

4. Verificación de usuario y contraseña (Fichero de Texto)

Si aún no has desarrollado el método GuardaListaTxt(), prográmalo ahora y ejecútalo para obtener un fichero de contraseñas en formato de texto.

Crea una **nueva solución** y reutiliza en ella el programa (Program.cs) de verificación para un fichero binario. Se supone que el fichero de texto contiene caracteres ASCII y que está organizado en líneas, conteniendo cada línea un registro.

Cada línea del fichero contiene el Nombre (16 caracteres), el Salt (24 caracteres en Base64), el resumen de la contraseña (44 caracteres en Base64) y el terminador de línea, CRLF ó \r\n.

Se reaprovecha gran parte del código del programa ya realizado para el fichero binario. Tan solo es necesario cambiar la lectura del fichero binario por la lectura del fichero de texto en el método **Verifica**(string **NombreIn**, string **ContraIn**) desarrollado anteriormente.

La secuencia de operaciones a realizar en el método es la siguiente:

1) Copiar el string **NombreIn** en un array de caracteres del mismo tamaño que los nombres almacenados en el fichero de contraseñas. Si la longitud de **NombreIn** es menor que la del array de caracteres completar el array con el mismo carácter usado para completar los nombres en el fichero de usuarios/contraseñas: ceros o espacios en blanco.

2) Declarar el array de caracteres **SaltChar** para cargar los caracteres de **Salt** en Base64 leyéndolos del fichero. El número de caracteres de este array se puede calcular a partir de la longitud de **Salt** en Bytes:

```
int NumCarSalt = (int)(4 * Math.Ceiling(((double)LongiSalt / 3.0)));
```

3) Declarar el array de caracteres **ResuContraChar** para cargar los caracteres de **ResuContra** en Base64 leyéndolos del fichero. El número de caracteres de este array se puede calcular a partir de la longitud de **ResuContra** en Bytes:

```
int NumCarResuContra = (int)(4 * Math.Ceiling(((double)LongiResuContra / 3.0)));
```

4) Buscar el **NombreIn** en el fichero de contraseñas. Para leer el fichero usar un **FileStream** y un **StreamReader** con un **Encoding** para los caracteres de tipo **ASCII**. El bucle de lectura de registros, hasta localizar el nombre del usuario o llegar al final del fichero, es similar al ya desarrollado. Dentro del bucle, usa el método **Read()** de StreamReader para leer **Nombre** directamente. Después lee **SaltChar** y usa el método **FromBase64CharArray()** de la clase **Convert** para obtener el array de bytes **Salt**. Seguidamente, lee **ResuContraChar** y usa el método FromBase64CharArray() para obtener el array de bytes **ResuContra**. Finalmente usa el método **ReadLine()** de StreamReader para leer el terminador de la línea de caracteres, CRLF o \r\n.

El resto del método es idéntico al desarrollado para el fichero binario.

5. Medir el tiempo de cálculo del resumen con la clase Rfc2898DeriveBytes

Realiza esta parte de la práctica sobre la solución de VS desarrollada previamente para crear el archivo de usuarios y contraseñas.

Para medir el tiempo que emplea un objeto de la clase Rfc2898DeriveBytes en calcular el resumen de una contraseña, usar un cronómetro, creando la variable **Crono** a partir de la clase **Stopwatch**. Crear la variable **Frecuencia** para almacenar la frecuencia de trabajo del cronómetro, que es el número de tics que cuenta en un segundo, usando la propiedad **Frequency**.

Para utilizar la clase **Stopwatch** es necesario usar el espacio de nombres **System.Diagnostics**.

Justo antes de llamar al método GetBytes() de la clase Rfc2898DeriveBytes, arranca el cronómetro invocando el método **Start()**. Justo después de llamar al método GetBytes(), para el cronómetro invocando el método **Stop()**.

Mostrar en la consola los milisegundos empleados a partir de la propiedad **ElapsedMilliseconds**. Si se necesita una resolución mayor en la medida de tiempos, usar la propiedad **ElapsedTicks**, que nos devuelve los tics que ha contado el cronómetro. Para convertir los tics a unidades de tiempo hay que dividirlos por la frecuencia del cronómetro. Pero si el tiempo a medir es muy pequeño el número de tics es menor que la frecuencia y al hacer la división entera de los tics entre la frecuencia se obtienen cero segundos transcurridos. Para medir tiempos muy pequeños, por ejemplo microsegundos, se multiplican los tics por 10^6 antes de efectuar la división, por ejemplo del modo siguiente:

```
double Tresu = ((double)(1000000L * Crono.ElapsedTicks)) / Frecuencia
```

Determinar la dependencia del tiempo de cálculo del resumen del valor de la propiedad **IterationCount** de la clase **Rfc2898DeriveBytes**. ¿Es lineal?. Si es lineal, determina la constante por la que hay que multiplicar el número de iteraciones para obtener el tiempo de cálculo del resumen.

6. Uso de funciones de resumen estándar en vez de Rfc2898DeriveBytes

En vez de usar una función de resumen "lenta", como la implementada en la clase Rfc2898DeriveBytes, se puede usar cualquier función de resumen estándar.

NO crees una nueva solución de VS. En la solución ya desarrollada para crear el fichero de usuarios y contraseñas, comenta el código de generación del resumen con Rfc2898DeriveBytes para sustituirlo por otro. Como se ha usado una longitud de 32 bytes para el resumen generado por Rfc2898DeriveBytes, es conveniente usar una función de resumen que también genere un resumen de 32 bytes (256 bits), como SHA256. De esta forma la estructura (longitud) de los registros no se modifica.

Las funciones de resumen estándar solo reciben un parámetro, un array de bytes que contiene la información a resumir. Por ello hay que concatenar el Salt y la Contraseña en un solo array.

Declara el array de bytes **NuevaContraBytes** y asígnale los bytes que se extraen del string **NuevaContra** usando **Encoding.Unicode.GetBytes()**.

Declara el array de bytes **SaltYContra** y resérvale el número de bytes definido por la suma de las longitudes de **Salt** y **NuevaContraBytes**.

Utiliza el método **CopyTo()** de Salt para copiar Salt en SaltYContra, al principio.

Utiliza el método **CopyTo()** de NuevaContraBytes para copiar NuevaContraBytes en SaltYContra, justo a continuación de Salt.

Crea un objeto **ProvSHA** proveedor de servicios criptográficos SHA256, por ejemplo, usando la clase **SHA256Managed**.

Invoca al método ComputeHash() del proveedor pasándole el array de bytes SaltYContra. Utiliza el cronómetro antes y después de invocar al método.

¿Qué tiempo emplea la función de resumen estándar en calcular el resumen? ¿Es significativamente menor que el de la función Rfc2898DeriveBytes?

Ahora habría que modificar los programas de verificación de contraseñas para que también pudiesen verificar una contraseña resumida con SHA256.

7. Variaciones del formato usado para almacenar los datos de los usuarios

Esta práctica admite infinidad de variaciones, que el alumno debe ser capaz de manejar. La primera variación obvia es cambiar de posición los tres elementos de cada registro del fichero de contraseñas y utilizar otras longitudes para los elementos.

Cambio del tipo utilizado para implementar la lista

La lista de usuarios se ha implementado usando un array unidimensional de tamaño fijo. Un tipo de dato más conveniente para implementarla es la colección **List<T>**, donde el tipo T sería el objeto User utilizado previamente.

Esto permite utilizar un tamaño de lista variable. El programa de creación de la Lista de Usuarios, se puede convertir en un pequeño gestor de usuarios. Además de contener los tres métodos ya descritos —**VerLista()**, **GuardaListaBin()**, **GuardaListaTxt()**— puede incluir nuevos métodos como **CargaListaBin()** o **CargaListaTxt()**, que permitan cargar una lista de usuarios desde ficheros.

Si el número de registros de usuarios en el fichero puede ser cualquiera, es necesario comprobar en la lectura de cada registro que se leen correctamente el número de bytes esperados para cada campo del registro. Cuando se lean cero bytes en el primer elemento de un registro se habrá alcanzado el final de fichero.

Tras leer los tres campos de un registro es necesario crear un objeto de la clase User y añadirlo a la colección del tipo **List<User>**. Para ello es necesario añadir a la clase User un segundo constructor que reciba como argumentos los tres elementos de un registro y los copie en los tres campos correspondientes del objeto.

También se pueden realizar otras operaciones, como eliminar usuarios, ordenar la lista de usuarios, etc., antes de volver a guardar una versión modificada de la lista cargada previamente.

Cambio del formato de registro de usuarios y contraseñas utilizado

Uno de los principales cambios que se puede implementar es la utilización de un tamaño variable para el nombre del usuario, utilizando un tipo string en vez de un tipo char[]. En este caso, los registros son de longitud variable.

Una variante adicional con registros de longitud variable es almacenarlos en formato XML.

Para trabajar con alguno de estos nuevos formatos se puede desarrollar un nuevo programa de generación de ficheros de contraseñas en formato binario, texto y xml.

Seguidamente, desarrollar un programa para la verificación de contraseñas para cada formato.

8. Almacenar usuarios en formato XML

Una variación importante del formato del fichero de contraseñas es utilizando XML. A continuación se muestra un posible formato del fichero de usuarios y contraseñas en XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LISTA>
  <USUARIO>
    <NOMBRE>Antonio</NOMBRE>
    <SALT>ogFkdTvfQJW5bwnC5dRWAA==</SALT>
    <RESUMEN>cCgA4SQE1GA9lyTiKhJ4MfBX/zkbA5ratZibGEX1FVQ=</RESUMEN>
  </USUARIO>
  <USUARIO>
    <NOMBRE>Benito</NOMBRE>
    <SALT>RsiOO1Irt9YTsWs1MuvuMg==</SALT>
    <RESUMEN>eP4rozMSTu1xVmlt4qD910ZbronyAqZYrQYfr89usrs=</RESUMEN>
  </USUARIO>
  <USUARIO>
    <NOMBRE>Carlos</NOMBRE>
    <SALT>RQRxGaaz0lahOA+xG5Dfkg==</SALT>
    <RESUMEN>4hQSQVSwAtXnNqFg+pB4DE1DqyrR3M5h5+uT/vriI+U=</RESUMEN>
  </USUARIO>
  <USUARIO>
    <NOMBRE>David</NOMBRE>
    <SALT>Uy9vTVqnLGRdxd+1fhbPHg==</SALT>
    <RESUMEN>oIzShlMvkLvyBUOXN0nZRoq4QYwCRkJp9w/PBZdxzU0=</RESUMEN>
  </USUARIO>
  <USUARIO>
    <NOMBRE>Eduardo</NOMBRE>
    <SALT>+mGB07CCH7pjECFJVugn7w==</SALT>
    <RESUMEN>7EiERsPGIceSckUP1WRqa00bykUGKhyJLGaBav88Hu4=</RESUMEN>
  </USUARIO>
</LISTA>
```

Utiliza la solución desarrollada al principio de la práctica para crear el fichero de contraseñas. Añade el método **GuardaListaXml**(string **NombreFich**), que almacena los tres datos de cada usuario (Nombre, Salt, ResuContra) en el fichero indicado, por ejemplo zz_Usuarios.xml.

Para escribir un fichero como el anterior, usar un objeto de la clase **XmlWriter** encadenado con otro objeto de la clase **FileStream**. Al crear el XmlWriter utilizar un objeto de la clase **XmlWriterSettings**, en el que al menos conviene configurar la propiedad **Indent = true** y elegir una codificación, por ejemplo **Encoding.UTF8**.

Utiliza los métodos **WriteStartElement()** y **WriteEndElement()** para escribir las etiquetas XML. Utiliza un bucle para recorrer la lista de usuarios escribiendo el nombre con el método **WriteElementString()** y después el Salt y el Resumen con el método **WriteBase64()**.

Crea una **nueva solución** para la verificación de un usuario y su contraseña usando un fichero de contraseñas con el formato XML definido previamente.

Reutiliza el código de la solución para la verificación de una contraseña almacenada en un fichero de texto y modifica la lectura del método Verifica().

Para leer un fichero como el anterior, usar un objeto de la clase **XmlReader** encadenado con otro objeto de la clase **FileStream**. Al crear el XmlReader utilizar un objeto de la clase **XmlReaderSettings**, en el que al menos conviene configurar la propiedad **CheckCharacters = true** y la propiedad **DtdProcessing = DtdProcessing.Prohibit**.

Hay dos opciones para leer secuencialmente los elementos del fichero XML.

Opción detallada

Se utiliza solo el método **Read()** de **XmlReader**.

Hay que leer la declaración inicial, el salto de línea el elemento <LISTA> y el salto que le sigue.

Después empieza el bucle do { } while(), en el que hay que leer todos los elementos, empezando por <USUARIO> y terminando con el salto que sigue al último elemento </USUARIO>.

Tras el bucle, lee el elemento </LISTA> para comprobar que la lectura termina correctamente. PERO esto solo tiene sentido hacerlo si se verifica la última contraseña del fichero, por lo que hay que comentar esta sentencia.

Si no se tiene experiencia con el procesamiento de XML, se recomienda que justo después de leer cada elemento se muestren sus propiedades **LocalName** y **Value** para comprobar que se va leyendo correctamente el fichero XML. De la propiedad Value hay que extraer el nombre y los caracteres en Base64 del Salt y el Resumen de la Contraseña. Después, se pueden obtener los bytes del Salt y el Resumen usando el método estático **FromBase64String()** de la clase **Convert**.

Opción compacta

Se utilizan métodos de lectura específicos para cada elemento del fichero XML, en la medida que sea posible.

Comenzar leyendo la declaración inicial del fichero XML con el método general Read().

Después utilizar la pareja de métodos:

LecX.ReadStartElement(“LISTA);

. . .

LecX.ReadEndElement(); // de LISTA (Esta sentencia habrá que comentarla)

En medio de estas dos sentencias colocar el bucle do { } while() con esta estructura:

```
LecX.ReadStartElement("USUARIO");
        LecX.ReadStartElement("NOMBRE");


        LecX.ReadEndElement(); // de NOMBRE
        Idem para SALT y RESUMEN
LecX.ReadEndElement(); // de USUARIO
```

Se puede leer el Nombre, la cadena de caracteres del Salt y la cadena de caracteres del Resumen usando el método ReadString() de XmlReader, y después si es necesario, convirtiendo la cadena de caracteres en Base64 a bytes usando la clase Convert.

Otra posibilidad es realizar la lectura usando los métodos ReadElementContentAsString() y ReadElementContentAsBase64(). Utilizar la ayuda para desarrollar el programa.

9. Almacenar datos de usuarios mediante serialización XML

En esta variante de la práctica se propone usar la técnica de la serialización de objetos para almacenar las credenciales de los usuarios en un fichero y la deserialización para cargar las credenciales desde un fichero.

En .NET se pueden usar varios tipos de serialización: binaria, XML y JSON. La serialización JSON solo está disponible en .NET Core. En esta sección se usará la serialización XML por ser la más común y la más sencilla de usar.

La serialización se aplica a los objetos de una determinada clase. Por ello, es muy conveniente definir la clase a serializar en un fichero separado que se pueda pasar directamente al programa en el que se realiza la deserialización. También es muy conveniente que esta clase sea lo más sencilla posible, lo que se consigue usando la clase solo para contener los datos de los usuarios, sin incluir ningún método para procesarlos.

Solución VS para crear un almacén de usuarios

Crea una nueva solución de Visual Studio. Crea las clases a serializar en un fichero separado. En el explorador de soluciones selecciona el proyecto y pulsa el botón derecho del ratón. En el menú emergente selecciona: Agregar > Nuevo elemento… > clase y, por ejemplo, usa el nombre Credenciales.cs para el fichero.

Cambia el espacio de nombres: namespace Credenciales.

Descarta la clase que ha generado el asistente.

Define la clase pública Almacen que solo contiene el campo público Lista del tipo List<Usuario>. Utiliza el operador new en la declaración del campo, para que al crear el objeto se cree también automáticamente el campo. Delante de public class pon el atributo [Serializable].

A continuación, define la clase pública Usuario, que puede comenzar con la declaración de dos variables de tipo int, de solo lectura, estáticas y privadas: **LongiSalt** y **LongiResuContra**, y asigna la longitud en bytes de ambos campos. Seguidamente, define tres campos públicos:

string Nombre

byte[] Salt

byte[] ResuContra

En los dos últimos campos utiliza el operador new para reservar el espacio correspondiente al objeto creado. Delante de public class pon el atributo [Serializable].

En el fichero principal del proyecto (Program.cs) hay que declarar el espacio de nombres en el que están declaradas las clases Almacen y Usuario. Declara: using Credenciales.

En la clase Program, tras el método estático **Main()**, genera el método estático **CreaUsuario()** que recibe dos string, con un nombre y una contraseña. En el código crea el objeto **NuevoUsu** de la clase Usuario, asigna el valor apropiado a sus campos y retórnalo.

A continuación, genera el método estático **VerLista()** que recibe un objeto de la clase Almacen y muestra en la consola los nombres de la lista de usuarios contenida en el almacén.

Ahora, integra en el método **Main()** el código necesario para crear el objeto **Alma** de la clase Almacen y añade al campo **Lista** de **Alma** objetos de la clase Usuario devueltos al llamar al método **CreaUsuario()**. Carga en **Alma** los 5 nombres y contraseñas de los usuarios típicamente usados en esta práctica: Antonio, conA ··· Eduardo, conE.

Invoca al método **VerLista()** para comprobar que el código funciona correctamente.

Ahora integra el código necesario para guardar el almacén **Alma** en un fichero utilizando la serialización XML. Comienza integrando en el fichero del proyecto la declaración:

```
using System.Xml.Serialization
```

En la clase Program crea el método estático **SerializaAlmacenXML()** que recibe un objeto de la clase Almacen y un string con el nombre de un fichero, por ejemplo "AlmacenUsuarios.xml". Utiliza una sentencia using() en la que debes crear un FileStream. En el cuerpo { } de using() crea el objeto **SerXML** de la clase XmlSerializer, pasándole como argumento el tipo de la clase Almacen usando el operador typeof(). Invoca el método **Serialize()** de **SerXML** pasándole el FileStream y el objeto de la clase Almacen a serializar.

Aunque la deserialización del objeto de clase Almacen se hará en un programa de verificación de usuarios y contraseñas, es conveniente hacerla también en este programa para comprobar la corrección de la serialización realizada.

En la clase Program crea el método estático **DeserializaAlmacenXML()** que recibe un string con el nombre del fichero que contiene el almacén serializado, y devuelve un objeto de la clase Almacen. Utiliza una sentencia using() en la que debes crear un FileStream. En el cuerpo { } de using() crea el objeto **SerXML** de la clase XmlSerializer, pasándole como argumento el tipo de la clase Almacen usando el operador typeof(). Invoca el método **Deserialize()** de **SerXML** pasándole el FileStream. Haz un cast a la clase (Almacen) al objeto devuelto por Deserialize(). Finalmente, la clase DeserializaAlmacenXML() debe retornar este objeto.

En el método Main() prueba los métodos de serialización y deserialización:

Invoca al método SerializaAlmacenXML() .

Declara Alma2 y cárgale el objeto devuelto por el método DeserializaAlmacenXML().

Invoca VerLista() para ver los usuarios de Alma2.

Con la serialización XML se pueden ver los objetos serializados. Al hacer doble clic en el fichero AlmacenUsuarios.xml deberá abrirse una pestaña en Visual Studio mostrando el contenido del fichero. Observa la estructura del almacén de credenciales y comprueba que los strings se han integrado directamente mientras que los arrays de bytes se han integrado codificándolos en Base64.

Solución VS para verificar usuarios

Una solución para verificar usuarios basada en la deserialización de un Almacen, funciona de forma diferente a las soluciones anteriores desarrolladas en esta práctica.

Esta solución reconstruye el Almacen (lista de usuarios) completamente en la memoria, y luego se busca el usuario a verificar en la lista completa que ya reside en la memoria.

Las soluciones previas leen los datos de un usuario de un fichero, y tras leerlo, comprueban si coincide con el usuario a verificar. Si no coincide se continúa leyendo usuario tras usuario hasta encontrar el usuario a verificar o alcanzar el final del fichero.

Crea una nueva solución de Visual Studio. Copia el fichero Credenciales.cs en esta solución y agrégalo al proyecto en el Explorador de soluciones.

Debajo del método Main(), copia en la clase Program los métodos **DeserializaAlmacenXML()** y **Verlista()**.

Crea el método estático y público **Verifica()** que recibe dos string, NombreIn y ContraIn a verificar y un objeto de la clase Almacen. Devuelve un número entero: 1 (nombre desconocido), 2 (contraseña incorrecta) y 0 (usuario y contraseña validados).

Se puede comenzar el método declarando 3 variables locales con las que referenciar los datos de cada usuario a comprobar.

```
string Nombre;
byte[] Salt;
byte[] ResuContra;
```

Realiza un bucle do{ }while() en el que se carga un usuario de la Lista en las variables anteriores y se comprueba si NombreIn coincide con Nombre. El bucle continua mientras que no se encuentre el nombre Y no se alcance el final de la lista.

El código necesario después del bucle es el mismo que en las versiones anteriores de verificación.

En el método Main(), lee de la consola un nombre de usuario y su contraseña, por ejemplo utilizando **Console.ReadLine()**. Después, deserializa el almacén de credenciales invocando al método DeserializaAlmacenXML() y muestra su contenido invocando al método Verlista(). Después, invoca el método Verifica(). Finalmente, utiliza una sentencia **switch** para mostrar en la consola un mensaje para cada valor devuelto por el método Verifica().

Recuerda que antes de ejecutar el programa de verificación tienes que copiar el fichero AlmacenUsuarios.xml generado previamente en el directorio \bin\Debug\ de la solución.