



## Curs 5

Codificarea datelor

  
itacad  
your technology

## Overview

- ▶ Arhivare
- ▶ Criptare
  - ▶ Simetrica
  - ▶ Asimetrica
- ▶ Serializare



Microsoft  
2.NET

11/7/2014

2

□ In cursul 5 vom analiza metode de codificare a datelor, printre care:

- ❖ Arhivarea datelor si dezarhivarea acestora in mod programatic
- ❖ Criptarea datelor pe care vrem sa le transmitem in siguranta
- ❖ Serializarea si deserializarea unor date

## Arhivarea datelor ( I )

- ▶ Namespace-ul **System.IO.Compression** ofera clasele:
  - ▶ DeflateStream (.NET 2.0)
  - ▶ GZipStream (.NET 2.0)
  - ▶ ZipArchive (.NET 4.5)
  - ▶ ZipArchiveEntry (.NET 4.5)
  - ▶ ZipFile (.NET 4.5)
  - ▶ ZipFileExtensions (.NET 4.5)

11/7/2014

3

❑ Arhivarea datelor reprezinta procesul de creare a unei colectii de documente sau informatii stocate intr-un singur fisier: arhiva.

❑ Platforma .NET ofera namespace-ul "System.IO.Compression", colectie care contine urmatoarele clase:

- ❖ DeflateStream (.NET 2.0 - .NET 4.5) : ofera metode si proprietati pentru compresia si decompresia stream-urilor de date folosind algoritmul Deflate
- ❖ GZipStream (.NET 2.0 - .NET 4.5) : ofera metode si proprietati folosite sa comprime si sa decompime streamuri de date.
- ❖ ZipArchive (valabila doar din versiunea .NET 4.5) : reprezinta un pachet de fisiere comprimate intr-o arhiva format .zip
- ❖ ZipArchiveEntry (valabila doar din versiunea .NET 4.5) : reprezinta un fisier comprimat dintr-o arhiva .zip
- ❖ ZipFile (valabila doar din versiunea .NET 4.5) : ofera metode statice pentru crearea, extragerea si deschiderea arhivelor format .zip
- ❖ ZipFileExtensions (valabila doar din versiunea .NET 4.5) : ofera metode extinse pentru clasele ZipArchive si ZipArchiveEntry.

❑ Algoritmul Deflate este o combinatie intre 2 algoritmi, **LZ77** si **Huffman**. Mai multe

detalii puteti gasi la [1].

[1] <http://en.wikipedia.org/wiki/DEFLATE>

## Arhivarea datelor ( II )

### ► Crearea unei arhive:

- Din unul sau mai multe fisiere
  - **ZipFile.Open** pentru deschiderea unei arhive la o cale specificata
  - **CreateEntryFromFile** – metoda a clasei ZipArchive ce creeaza o intrare in arhiva
  - **Dispose** pentru a elibera resursa
- Dintr-un director (folder)
  - **ZipFile.CreateFromDirectory(String, String);**

11/7/2014

4

### ❑ Crearea unei arhive

- ❖ O arhiva se poate crea in doua moduri: selectand unul sau mai multe fisiere sau selectand un director.
- ❖ Pentru crearea unei arhive dintr-un director se poate folosi metoda statica "CreateFromDirectory" a clasei ZipFile. Metoda primeste doi parametri, ambii de tip String. Primul reprezinta calea catre directorul sursa, pe cand cel de-al doilea reprezinta calea(inclusiv numele) arhivei care urmeaza sa fie creata.
- ❖ Pentru crearea unei arhive din mai multe fisiere se poate folosi un cod asemanator cu cel de mai jos

```
ZipArchive zip = ZipFile.Open(archiveName, ZipArchiveMode.Create);
```

```
foreach(string file in files)
```

```
{
```

```
    zip.CreateEntryFromFile(file, Path.GetFileName(file), CompressionLevel.Optimal);
```

```
}
```

```
zip.Dispose();
```

- ❖ Se poate observa ca ultima instructiune este apelarea metodei **Dispose** care elibereaza resursa folosita. Nu uitati ca platforma .NET are componenta de GarbageCollector care elibereaza memoria nefolosita, **insa** acesta nu elibereaza alte resurse in afara de memorie (exemplu: file handler).

## Arhivarea datelor ( III )

- ▶ Citirea elementelor din arhiva:
  - ▶ **ZipFile.OpenRead(fileName)** pentru deschiderea arhivei in modul de citire
  - ▶ Obținerea elementelor din arhiva folosind proprietatea **Entries**
    - ▶ Colectie ce retine obiecte de tip **ZipArchiveEntry**

11/7/2014

5

❑ Pentru a citi elementele dintr-o arhiva trebuie urmati pasii:

- ❖ Deschiderea arhivei folosind metoda **OpenRead** a clasei ZipFile
- ❖ Folosirea proprietatii **Entries** a obiectului rezultat la pasul anterior pentru a obtine toate elementele din arhiva
- ❖ Iterarea prin colectia de elemente si prelucrarea acestora.


```
ZipArchive zip = ZipFile.OpenRead(fileName)
```

```
foreach (ZipArchiveEntry entry in zip.Entries)
```

```
{
```

```
    // print entry.FullName
```

```
}
```



## Arhivarea datelor ( IV )

- ▶ Extragerea elementelor dintr-o arhiva:
  - ▶ Se dezarhiveaza intr-un director cu ajutorul metodei **ExtractToDirectory** a clasei **ZipFile**. Metoda primeste ca parametri:
    - ▶ **CaleaCatreArhivaSursa**
    - ▶ **CaleaCatreFolderulDestinatie**

Microsoft .NET

11/7/2014

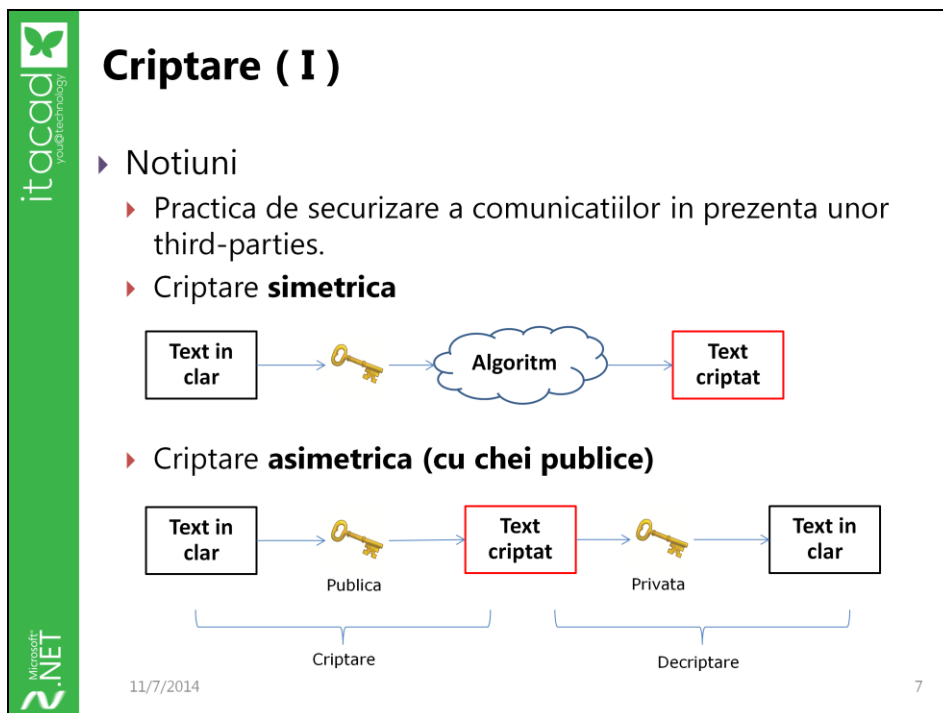
6

❑ Pentru a se extrage fisierele dintr-o arhiva este folosita metoda **ExtractToDirectory** a clasei ZipFile.

- ❖ Metoda primeste 2 parametri – calea catre arhiva ce urmeaza a fi dezarhivata si calea catre folderul destinatie

**ZipFile.ExtractToDirectory(archivePath, folderPath);**







❑ Notiunea de criptare se refera la practica tehnicilor de securizare a comunicatiilor in prezenta unor “third parties” care incearca interceptarea acestor date. Criptarea utilizeaza chei de criptare pe care sender-ul si receiver-ul le folosesc pentru a decripta aceste date.

❑ Exista mai multi algoritmi de criptare si doua mari categorii:

- ❖ Criptarea simetrica – foloseste aceeasi cheie atat la receiver cat si la sender. Sender-ul o foloseste pentru a cripta, receiver-ul pentru a decripta mesajul. Un mare dezavantaj este management-ul sistemului de chei private care trebuie distribuite in mod secret celor doua parti din comunicatie.
- ❖ Criptare asimetrica (sau criptare cu chei publice) – foloseste o pereche de chei, una publica si alta privata. Cele doua chei au o legatura matematica si sunt generate ca o pereche. Avantajul acestui sistem este ca cheia privata nu poate fi ghicita computational din cea publica. Pentru ca sender-ul sa comunice cu receiver-ul, el trebuie sa creeze mesajul cu cheia publica a receiver-ului, dupa care sa trimita mesajul catre acesta. Receiver-ul, pentru decriptare, va folosi cheia sa privata. Pentru comunicarea inversa se procedeaza similar.



## Criptare ( II )

- ▶ Namespace-ul **System.Security.Cryptography**
- ▶ Criptare simetrica in .NET
  - ▶ **SymmetricAlgorithm**
    - ▶ Clasa abstracta
    - ▶ Metode: **Create, CreateDecryptor, CreateEncryptor**
    - ▶ Proprietati: **Key, IV**
    - ▶ Clase derivate: **Aes, DES, RC2, Rijndael, TripleDES**

11/7/2014 8

☐ In platforma .NET exista un namespace ce ofera suport pentru criptarea si decriptarea datelor folosind atat algoritmi cu chei simetrice cat si algoritmi cu chei asimetrice. Acest namespace se numeste **System.Security.Cryptography**

☐ Pentru criptarea simetrica, platforma .NET ofera clasa abstracta "SymmetricAlgorithm" pe care toate clasele concrete trebuie sa o implementeze. Aceasta clasa ofera, printre altele, metodele:

☐ **Create** – creeaza un obiect criptografic; metoda statica

☐ **CreateDecryptor / CreateEncryptor** – creeaza un obiect decryptor/encriptator simetric

☐ Clasa ofera si urmatoarele proprietati:

☐ **Key** – Citeste sau seteaza cheia secreta pentru algoritmul simetric

☐ **IV** – Citeste sau seteaza vectorul de initializare pentru algoritmul simetric

☐ Clase derivate (in functie de algoritmul folosit):

☐ **Aes**

☐ **DES**

☐ **RC2**

☐ Rijndael☐ TripleDES☐ **Exemplu** de criptare a unui element dintr-un XML:

1. Generarea unei chei simetrice folosind un algoritm concret, in cazul nostru **Rijndael**.

```
RijndaelManaged key = null;
```

```
key = new RijndaelManaged();
```

2. Crearea unui obiect **XmlDocument** pentru incarcarea fisierului xml in program.

```
XmlDocument xmlDoc = new XmlDocument();
```

```
xmlDoc.PreserveWhitespace = true;
```

```
xmlDoc.Load("test.xml");
```

3. Identificarea elementului din xml pe care vrem sa il criptam. Vom cauta elementul cu numele "password"

```
XmlElement elementToEncrypt =
```

```
Doc.GetElementsByTagName("password")[0] as XmlElement;
```

4. Criptarea elementului folosind o instanta a clasei **EncryptedXml**. Ultimul parametru al metodei EncryptData se refera la modul de criptare a elementului: pus pe **true** metoda va cripta doar continului elementului, pe cand pus pe **false** metoda va cripta intreg elementul.

```
EncryptedXml eXml = new EncryptedXml();
```

```
byte[] encryptedElement = eXml.EncryptData(elementToEncrypt,  
key, false);
```

5. Vom crea un obiect al clasei **EncryptedData** care va reprezenta inlocuitorul elementului de criptat (in clar text). Dupa creare trebuie populat cu un URL ce permite receiverului (celui care decripteaza) sa detecteze faptul ca acest element este unul criptat.

```
EncryptedData edElement = new EncryptedData();  
edElement.Type = EncryptedXml.XmlEncElementUrl;
```

6. Vom crea o metoda de criptare si o vom seta obiectului creat la pasul anterior. In functie de dimensiunea cheii pe care vrem sa o folosim, putem folosi diferite proprietati ale clasei EncryptedXml.


```
string encryptionMethod = EncryptedXml.XmlEncAES128Url;  
edElement.EncryptionMethod = new  
EncryptionMethod(encryptionMethod);
```

7. Adaugam elementul criptat la pasul 4 instantei "edElement"

```
edElement.CipherData.CipherValue = encryptedElement;
```


8. Inlocuim elementul original cu cel criptat

```
EncryptedXml.ReplaceElement(elementToEncrypt, edElement,  
false);
```

  
itacad  
your technology

## Criptare ( III )

- ▶ Namespace-ul **System.Security.Cryptography**
- ▶ Criptare asimetrica in .NET
  - ▶ **AsymmetricAlgorithm**
    - ▶ Clasa abstracta
    - ▶ Metode statice: **Create()**, **Create(String)**
    - ▶ Proprietati: **KeySize**, **LegalKeySizes**
    - ▶ Clase derivate: **DSA**, **RSA**

  
Microsoft  
.NET

11/7/2014 9

❑ Pentru criptarea asimetrica (cu chei publice) platforma .NET ofera in cadrul aceluiasi namespace, clasa abstracta **“AsymmetricAlgorithm”** ce ofera:

❑ Metodele:

❑ **static Create()** – creeaza un obiect pentru criptarea asimetrica cu algoritmul default de tip “RSA”

❑ **static Create(String)** – creeaza un obiect pentru criptarea asimetrica cu algoritmul specificat prin parametru

❑ Proprietatile:

❑ **KeySize** – seteaza sau citeste marimea cheii de criptare (in biti)

❑ **LegalKeySizes** – citeste marimile disponibile pentru cheile de criptare ale algoritmului

❑ Singurele implementari disponibile pentru aceasta clasa abstracta sunt realizate de clasele concrete **DSA** si **RSA** din cadrul aceluiasi namespace.


## Serializare ( I )

- ▶ Procesul de transformare a unui obiect sau a unei structuri de date intr-un format care poate fi stocat si reconstruit mai tarziu
  - ▶ Fișiere
  - ▶ Stream-uri
- ▶ Namespace-ul **System.Xml.Serialization**
- ▶ Namespace-ul **System.Runtime.Serialization**

11/7/2014

10

- ☐ Notiunea de serializare se refera la procesul de transformare a unui obiect sau a unei structuri de date intr-un format care poate fi stocat (exemplu: fisier, buffer-ul memoriei sau transmis pe un canal de comunicare) si reconstruit mai tarziu.
- ☐ Platforma .NET ofera un namespace pentru serializare/deserializare folosind fișiere xml, namespaceul **System.Xml.Serialization**. Acesta ofera clase folosite la serializarea si deserializarea obiectelor in formate XML sau in stream-uri.
- ☐ Namespace-ul **System.Runtime.Serialization** ofera clase si interfete pentru serializarea si deserializarea obiectelor in si din streamuri binare.



## Serializare ( II )

- ▶ XML
  - ▶ **System.Xml.Serialization.XmlSerializer**
  - ▶ Constructori: **XmlSerializer(Type)**, **XmlSerializer(Type, String)**
  - ▶ Metode:
    - ▶ **Serialize(Stream, Object)**
    - ▶ **Serialize(TextWriter, Object)**
    - ▶ **Deserialize(Stream)**
    - ▶ **Deserialize(TextReader)**
    - ▶ **CanDeserialize(XmlReader)**

11/7/2014

11

❑ Clasa principala din acest namespace este **XmlSerializer**, clasa folosita pentru serializarea si deserializarea obiectelor in si din fisiere XML. Clasa ofera:

❑ Constructorii:


- ❑ **XmlSerializer(Type)** – parametrul reprezinta un tip de clasa folosit pentru serializare
- ❑ **XmlSerializer(Type, String)** – cel de-al doilea parametru va seta namespaceul default pentru toate elementele XML

❑ Metodele:

- ❑ **Serialize(Stream, Object)** – va serializa obiectul (parametrul al doilea) folosind stream-ul primit (primul parametru)
- ❑ **Serialize(TextWriter, Object)** – se foloseste un TextWriter pentru scrierea in fisier
- ❑ **Serialize(XmlWriter, Object)** – se foloseste un XmlWriter pentru scrierea in fisier
- ❑ **Deserialize(Stream)** – deserializeaza folosind stream-ul primit ca parametru


- ☐ Deserialize(Reader) – se foloseste un Reader
- ☐ Deserialize(XmlReader) – se foloseste un XmlReader
- ☐ CanDeserialize(XmlReader) – metoda folosita pentru a se verifica daca un anumit obiect xml poate fi deserializat
- ☐ Pentru metodele de deserializare este nevoie de o conversie (cast) la obiectul in care vom stoca datele citite din fisier



  
itacad  
your technology

## Serializare ( III )

- ▶ Streamuri binare
  - ▶ **System.Runtime.Serialization.ISerializable**
  - ▶ Metoda **GetObjectData(SerializationInfo, StreamingContext)**
  - ▶ Clasa **Formatter** si derivatele sale ofera metodele **Serialize** si **Deserialize** care vor apela automat metoda **GetObjectData** a obiectului

  
Microsoft  
.NET

11/7/2014 12

- ☐ In acest namespace se gaseste o interfata care permite obiectelor sa controleze modul de serializare si de deserializare. Aceasta interfata se numeste **ISerializable** si are o singura metoda:
  - ☐ **GetObjectData(SerializationInfo, StreamingContext)** – populeaza primul parametru cu datele necesare serializarii obiectului. Cel de-al doilea parametru reprezinta destinatia pentru aceasta serializare.
- ☐ In metoda de mai sus trebuie adaugate valorile clasei pentru a le serializa.
- ☐ Pentru deserializare se poate crea un constructor care sa primeasca un obiect de tip **SerializationInfo** care va contine valorile campurilor clasei.
- ☐ Atat pentru serializare cat si pentru deserializare este nevoie de un obiect de tip **Formatter** ce va performa serializarea si deserializarea obiectului:
  - ☐ La serializare se va apela automat metoda **GetObjectData** a obiectului
  - ☐ La deserializare se va apela automat constructorul care primeste **SerializationInfo** ca parametru.
- ☐ Exemplu:

Fie clasa **Student** cu campurile **name**, **universityName** ce implementeaza interfata **ISerializable**. Implementand aceasta metoda, clasa este nevoita sa ofere implementare metodei **GetObjectData**. Aceasta implementare poate fi (unde **info** este un obiect de tip **SerializationInfo**):

```
if (info == null)

    throw new System.ArgumentNullException("info");

info.AddValue("nume", "StudentX");

info.AddValue("univName", "Politehnica");
```

Prin implementarea metodei de mai sus ne-am asigurat ca vom putea serializa acest obiect. Pentru deserializare trebuie creat un constructor care primeste 2 argumente : **SerializationInfo** si **StreamingContext** .

```
if (info == null)

    throw new System.ArgumentNullException("info");

name_value = (string)info.GetValue("nume", typeof(string));

univ_value = (int)info.GetValue("univName", typeof(int));
```

Serializare (s = instanta a clasei Student):

```
BinaryFormatter binaryFmt = new BinaryFormatter();

FileStream fs = new FileStream ("Person.xml",
FileMode.OpenOrCreate);

binaryFmt.Serialize(fs, s);

fs.Close();
```

Deserializare:

```
fs = new FileStream ("Person.xml", FileMode.OpenOrCreate);
```

```
Student s2 = (Person)binaryFmt.Deserialize(fs);
```

## Review

- ▶ Arhivare
- ▶ Criptare
  - ▶ Simetrica
  - ▶ Asimetrica
- ▶ Serializare

