### Modulul 14. Interacțiunea cu fișierele în C#.

#### Modelul Stream în C#. Librăria System.IO.

Librăria System.IO în .NET reprezintă o colecție de metode și proprietăți care ajută programatorilor să citească și să înregistreze date într-o sursă externă(fișier, bază de date, server etc.). Caracterele **IO** pot fi descifrate ca Input/Output, ceea ce vorbește deja despre scopul librăriei.

**Observație: Vom include librăria IO atunci când avem nevoie să interacționăm cu surse externe.**

#### Clasa Stream

Clasa [Stream](https://docs.microsoft.com/en-us/dotnet/api/system.io.stream?view=netcore-3.1) este clasa de bază din System.IO care este folosită pentru citirea și înregistrarea datelor în surse externe.

Stream este o clasă **abstractă,** motivul pentru acest lucru este faptul că multiple clase sunt derivate de la aceasta (în dependență de tipul de date ex, FileStream, MemoryStream etc.)

Este important să menționăm că clasele **Stream** includ clase Reader și Writer, care adaugă funcționalitate de citite și înscriere în fișiere.

Clasa Stream, de asemenea, include funcționalitate ce permite căutarea unor valori în datele care le primim, retragerea unor caracteristici precum lungimea șirului de date, proprietățile șirului(dacă poate fi doar citit sau și modificat etc.)

**Adițional:** Puteți crea personal clase derivate de la clasa abstractă Stream, dar în acest caz va trebui să implementați metodele: Read, Write, ReadByte, WriteByte, CanRead, CanSeek, CanWrite, Flush, Length, Position, Seek, SetLength.

#### Clasa FileStream

Clasa FileStream este o clasă derivată din clasa Stream, care ne ajută să efectuăm operațiuni de bază precum crearea fișierelor, înscrierea sau citirea în ele.

**Exemplu:**

|  |
| --- |
| namespace FileExample  {  class CreateFile  {  static void Main()  {  string path = "D:\\csharpfile.txt";  FileStream fs = new FileStream(path, FileMode.Create);  fs.Close();  Write("Fisierul a fost creat si are adresa: {0}", path);  }  }  } |

În exemplul de mai sus am folosit clasa FileStream pentru a crea un fișier nou pe adresa path, care am definit-o într-o variabilă string locală.

Probleme pentru rezolvare.

### **Program 1:**

1. Pe baza exemplului de mai sus creați un program care va cere să scrieți denumirea fișierului la consolă, apoi va crea fișierul dat.

#### Citirea Stream-urilor și Stream Reader

Pentru citirea informației dintr-un stream vom folosi un **Stream Reader**. Un Stream Reader este un nivel intermediar între sursa externă și programul pe care îl scrieți, în esență acesta:

1. Extrage informația din sursa externă
2. O transformă în date binare
3. O transformă în tipul de date cerut de programator
4. Execută logica cerută de programator(afișează datele, execută operații cu ele etc.)

**Exemplu:**

|  |
| --- |
| namespace StreamReaderExample  {  class StreamReaderClass  {  static void Main(string[] args)  {  string path = "D:\\Example.txt";  using (StreamReader sr = File.OpenText(path))  {  string s = "";  while ((s = sr.ReadLine()) != null)  WriteLine(s);  }  }  }  } |

În exemplul de mai sus am demonstrat cum poate fi citit un stream de date de tip text. Atrageți atenția că în cazul dat trebuie să indicăm sursa fișierului care va fi citit, apoi vom folosi sintaxa **using(StreamReader sr = File.OpenText(path)) (using nu este necesar)** pentru a deschide fișierul indicat în formă textuală. Folosind un ciclul while vom afișa fiecare rând din fișier până când vom ajunge la sfârșit.

Probleme pentru rezolvare.

### **Program 1:**

1. Pe baza exemplului de mai sus deschideți un alt tip de fișier(diferit de .txt) și afișați conținutul acestuia.

### **Program 2:**

1. Pe baza exemplului de mai sus creați un program care va citi conținutului unui fișier .txt (care va conține mai multe rânduri) dar va afișa doar rândul 1.

### **Program 3:**

1. Pe baza exemplului de mai sus creați un program care va citi conținutului unui fișier .txt dar afișați doar literele vocale.

#### Înscrierea în Stream-urilor și Stream Writer

Funcționalitatea opusă citirii stream-urilor este înscrierea în ele și aceasta o vom executa cu ajutorul unui **Stream Writer**.

**Exemplu:**

|  |
| --- |
| namespace StreamWriterExample  {  class StreamWriterClass  {  static void Main()  {  string path = "D:\\Example.txt";  using (StreamWriter sr = File.AppendText(path))  {  sr.WriteLine("\nStream Writer");  sr.Close();  }  WriteLine(File.ReadAllText(path));  }  }  } |

În exemplul de mai sus am utilizat același fișier textual, dar de data aceasta am adăugat un text la conținutul acestuia cu ajutorul unui StreamWriter și a metodei AppendText.

Atrageți atenția că atunci când am finisat operația am închis StreamWriterul cu ajutorul metodei Close().

Deși în cazul dat închiderea Stream-ului nu afectează execuția programului în unele cazuri omiterea acestei operații poate aduce la unele greșeli neașteptate(de ex. ultimul rând nu va fi înscris sau citit etc.).

Probleme pentru rezolvare.

### **Program 1:**

1. Modificați exemplul de mai sus ca să rescrie conținutul fișierului(adică textul care îl introducem să înlocuiască conținutul deja existent).

### **Program 2:**

1. Scrieți un program care va conține o structura InformatiiUtilizator, aceasta va include ID-ul utilizatorului, nickname-ul, parola, numărul de comentarii și alte date(la alegere).
2. Adăugați un constructor cu parametri pentru a seta valorile structurii.
3. În metoda Main() creați 10 obiecte de tip InformatiiUtilizator.
4. Stocați informația despre utilizatori într-un fișier textual în formă structurată cu ajutorul StreamWriter.
5. Afișați informația despre utilizatori la consolă cu ajutorul StreamReader.

#### Utilizarea claselor Directory și DirectoryInfo pentru operații cu fișierele

Clasa **Directory** în C# permite utilizatorilor să execute operații asupra mapele precum crearea, ștergerea, mutarea, accesarea proprietăților unui fișier etc.

**Crearea fișierelor**

1. **Directory.CreateDirectory** - creează un fișier cu adresa indicată
2. **Directory.Exists** - verifică dacă fișierul cu adresa indicată există(returnează true/false)

**Exemplu**:

|  |
| --- |
| namespace DirectoryExample  {  class CreateFolder  {  static void Main()  {  if (!Directory.Exists("D:\\newFolder"))  {  Directory.CreateDirectory("D:\\newFolder");  WriteLine("Fisier creat cu succes");  }  else  WriteLine("Fisierul deja exista");  }  }  } |

În exemplul prezentat am folosit metoda Directory.Exists pentru a verifica dacă un fișier deja există, în cazul în care nu există îl creăm, dacă da atunci afișăm la consolă mesajul respectiv.

Probleme pentru rezolvare.

### **Program 1:**

1. Pe baza programului de mai sus creați un program care va crea 3 fișiere.
2. Fișierul 1 va avea adresa în discul D.
3. Fișierul 2 va fi locat în interiorul fișierului 1.
4. Fișierul 3 se va afla în fișierul 2.

**Ștergerea fișierelor**

1. **Directory.Delete** - șterge fișierul cu adresa indicată

**Exemplu**:

|  |
| --- |
| namespace DirectoryExample2  {  class DeleteFolder  {  static void Main()  {  string path = "D:\\newFolder";  if (Directory.Exists(path))  Directory.Delete(path);  }  }  } |

Probleme pentru rezolvare.

### **Program 1:**

1. Pe baza problemei precedente verificați dacă toate cele 3 fișiere create există, dacă da ștergeți fișierul 3.

**Mutarea fișierului**

1. **Directory.Move** - mută un fișier dintr-o adresă inițială spre altă adresă finală.

**Exemplu**:

|  |
| --- |
| namespace MutareaFisierelor  {  class DeleteFolder  {  static void Main()  {  string adresa = "D:\\Exemplu";  string destinatie = "D:\\1\\Exemplu";  Directory.Move(adresa, destinatie);  }  }  } |

Probleme pentru rezolvare.

### **Program 1:**

1. Folosiți programului de mai sus și implementați logică adițională care verifică dacă fișierul inițial există, dacă nu creați-l.
2. Verificați dacă fișierul 1 există, dacă nu creați-l.
3. Efectuați mutarea.

**Extragerea tuturor fișierelor dintr-un fișier**

1. **Directory.EnumerateDirectories** - extrage toate fișierele dintr-un fișierul de bază

În exemplul prezentat am folosit metoda Directory.Exists pentru a verifica dacă un fișier deja există, în cazul în care nu există îl creăm, dacă da atunci afișăm la consolă mesajul respectiv.

**Exemplu**:

|  |
| --- |
| namespace EnumeraFisierele  {  class EnumerateFolders  {  static void Main()  {  string docPath = "D:\\";  List<string> dirs = new List<string>(Directory.EnumerateDirectories(docPath));  foreach (var dir in dirs)  WriteLine($"{dir.Substring(dir.LastIndexOf(Path.DirectorySeparatorChar) + 1)}");  WriteLine($"{dirs.Count} fisiere gasite.");  }  }  } |

Probleme pentru rezolvare.

### **Program 1:**

1. Folosiți programului de mai sus pentru a găsi fișierul care conține cele mai multe fișiere din discul D.

**Proprietățile DirectoryInfo**

Pe lângă clasa Directory care ne oferă acces la toate operațiile prezentate mai sus, C# ne oferă și clasa **DirectoryInfo,** care conține un set de proprietăți care ne ajută să extragem mai multe informații despre un fișier.

|  |  |
| --- | --- |
| **Proprietatea** | **Descriere** |
| **CreationTime** | Returnează când a fost creat fișierul |
| **FullName** | Returnează numele complet a fișierului inclusiv numele discului și mapele părinți |
| **Name** | Returnează doar numele fișierului |
| **LastAccessTime** | Returnează când a fost deschis fișierul ultima oară |
| **LastWriteTime** | Returnează când a fost modificat fișierul ultima oară |
| **Root** | Returnează numele discului și a mapelor în care se află fișierul |

**Exemplu**:

|  |
| --- |
| namespace DirInfo  {  class Program  {  static void Main(string[] args)  {  DirectoryInfo fi = new DirectoryInfo("D:\\1");  WriteLine("Numele fișierului: {0} ", fi.Name);  WriteLine("Fișierul a fost creat: {0} ", fi.CreationTime.ToLongTimeString());  WriteLine("Fișierul a fost accesat ultima oară: {0} ", fi.LastAccessTime.ToLongDateString());  WriteLine("Fișierul a fost modificat ultima oară: {0} ", fi.LastWriteTime);  WriteLine("Locația fișierului este: {0} ", fi.Root);  }  }  } |

Probleme pentru rezolvare.

### **Program 1:**

1. Afisați când au fost create toate fișierele din discul D care au fost deschise în 2020.

### **Program Final:**

1. Creați o structură Tranzacții care va conține variabilele id, suma, numeClient, timpulEfectuarii și alte variabile. Declarați un constructor cu parametri pentru structură.
2. În metoda Main creați 20 de obiecte de tip Tranzacție, folosiți constructorul cu parametri și folosiți valori aleatorii pentru toate variabilele. (id-ul un număr random, suma - la fel, numele îl puteți selecta aleatoriu dintr-un array prestabilit, timpul random(începând de la data X până la data Y).
3. În metoda Main creați 20 de obiecte de tip Tranzacție, folosiți constructorul cu parametri și folosiți valori aleatorii pentru toate variabilele.
4. Creați o mapă StocareDate în discul D(dacă mapa nu există).
5. Păstrați toate datele despre tranzacții într-un document .txt. Dacă documentul există înlocuiți datele vechi cu cele noi.
6. Afișați la consolă toate informațiile despre tranzacții(din fișierul .txt) precum și când acestea au fost actualizate ultima oară.