### Modulul 17. LINQ.

#### Ce este LINQ?

LINQ sau Language-Integrated Query reprezintă un set de funcționalități în C# și Visual Basic create cu scopul de ajuta să obțină informații rapid și simplu din diferite surse precum unele obiecte, variabile interne(mai multe obiecte, un array de numere etc.) sau externe(baze de date, documente XML etc.

#### Utilitatea LINQ.

Utilitatea de bază a LINQ este faptul că acesta permite programatorilor să extragă informații într-un mod foarte flexibil(putem extrage atât obiecte singulare cât și volume foarte mari de informație) apoi să le stocheze în colecții structurate și să folosească un set larg de proprietăți și metode pentru îndeplini diverse scopuri.

În linii generale LINQ este soluția perfectă atunci când aveți nevoie să extrageți și să prelucrați un set de date.

**Atunci când vom folosi LINQ trebuie să includem librăria using System.Linq;**

#### Ce este un request în LINQ?

1. **Definiția unui request(query)**

Un request LINQ este în esență o cerere de a extrage anumite elemente sau informații care corespun unui anumit criteriu indicat de programator.

**Exemplu:**

|  |
| --- |
| namespace LINQ  {  class Program  {  static void Main()  {  string[] cuvinte = { "salut", "Ion", "Mississippi", "oxigen", "cuvinte" };  var cuvinteLungi = from cuvint in cuvinte where cuvint.Length >= 6 select cuvint;  foreach (var cuvint in cuvinteLungi)  Console.WriteLine(cuvint);  }  }  } |

Un exemplu simplu de request este demonstrat în programul de mai sus care extrage dintr-un masiv de cuvinte doar cuvintele care au lungimea mai mare sau egală cu 6 litere.

Probleme pentru rezolvare.

### **Program 1:**

1. Creați un program care să creeze un masiv cu 10 cifre aleatorii și să extragă din el divizorii lui 3.
2. **Sintaxa requesturilor**

Când vrem să realizăm un request LINQ avem 2 opțiuni legate de sintaxa scrierii acestora:

* **Cu ajutorul expresiei Lamba**

|  |
| --- |
| var cuvinteLungi = cuvinte.Where( w ⇒ w.Length >= 6); |

* **Varianta prelungită**

|  |
| --- |
| var cuvinteLungi = from w in cuvinte where w.Length >= 6 select w; |

Probleme pentru rezolvare.

### **Program 1:**

1. Rezolvați problema precedentă cu ajutorul expresiei Lambda.
2. **Tipuri de requesturi LINQ**

* **LINQ către Obiecte** - în esență permite trimiterea requesturilor către obiectele interne din program, precum un masiv, o colecție etc.
* **LINQ către XML(XLINQ) -** din motiv că fișierele XML sunt de fapt containere structurate pentru date LINQ permite extragerea, filtrarea datelor de care avem nevoie
* **LINQ către DataSet** - precum am discutat la lecția precedentă DataSet-urile sunt tabele structurate de date, din acest motiv putem folosi LINQ
* **LINQ către SQL (DLINQ)** - crează o legătură relațională dintre variabilele programului și o valoare din baza de date. Prin acest principiu folosi LINQ pentru a accesa valorile de pe serverul în care se conține baza de date.
* **LINQ to Entities** -

1. **Operatorii LINQ**
2. **Select -** Un request LINQ este în esență o cerere de a extrage anumite elemente sau informații care corespund unui anumit criteriu indicat de programator.

**Exemplu:**

|  |
| --- |
| namespace LINQSelect  {  public class Employee  {  public int ID { get; set; }  public string Prenume { get; set; }  public string Nume { get; set; }  public int Salariu { get; set; }  public static List<Employee> GetEmployees()  {  List<Employee> employees = new List<Employee>  {  new Employee {ID = 101, Prenume = "Ion", Nume = "Nicolaescu", Salariu = 5000 },  new Employee {ID = 102, Prenume = "Maria", Nume = "Croitor", Salariu = 6000 },  new Employee {ID = 103, Prenume = "Radu", Nume = "Moraru", Salariu = 10000 },  new Employee {ID = 104, Prenume = "Daniel",Nume = "Urechean", Salariu = 25000 },  new Employee {ID = 105, Prenume = "Vasile",Nume = "Varzar", Salariu = 7000 },  new Employee {ID = 106, Prenume = "Petru", Nume = "Soimar", Salariu = 8000 }  };  return employees;  }  }  class Program  {  static void Main()  {  List<Employee> basicQuery = (from emp in Employee.GetEmployees() where emp.Salariu >= 7000 select emp).ToList();  foreach (Employee emp in basicQuery)  Console.WriteLine($"ID: {emp.ID} Nume: {emp.Prenume} {emp.Nume} Salariu: {emp.Salariu} lei.");  }  }  } |

În exemplul de mai sus am folosit operația select în combinație cu un filtru logic where pentru a selecta toți angajații care au un salariu mai mare egal de 7000 lei. Observați că în cazul dat operația select alege un obiectele care coincid cu condiția indicată de filtru.

Probleme pentru rezolvare.

### **Program 1:**

1. Modificați exemplul de mai sus ca să obțineți toți angajații numele cărora începe cu V, și au un salariu între 7000-12000.
2. **Filtrele OfType -** Spre deosebire de filtrele Where care permit execuția unei verificări logice, filtrele OfType verifică datele după tipul acestora

**Exemplu:**

|  |
| --- |
| namespace LINQOfType  {  class Program  {  static void Main()  {  List<object> variabile = new List<object>() { "cuvint", "string", 50075, "masina", 'a', 63, "James", true };  List<int> numere = variabile.OfType<int>().ToList();  foreach (int numar in numere)  Console.WriteLine(numar);  }  }  } |

În exemplul indicat am folosit filtrul OfType pentru a identifica doar numerele dintr-o listă de variabile generice(tip object).

Probleme pentru rezolvare.

### **Program 1:**

1. Modificați exemplul de mai sus ca să obțineți toate cuvintele cu lungimea mai mare egală de 5.

|  |
| --- |
| namespace LINQOfType  {  class Program  {  static void Main()  {  List<object> variabile = new List<object>() { "cuvint", "string", 50075, "masina", 'a', 63, "James", true };  List<string> numere = variabile.OfType<string>().ToList();  numere = numere.Where(x => x.Length >= 5).ToList();  foreach (string numar in numere)  Console.WriteLine(numar);  }  }  } |

1. **Operatorul Distinct -** Permite extragerea elementelor unice fără repetiții.

**Exemplu:**

|  |
| --- |
| namespace LINQDistinct  {  class Program  {  static void Main(string[] args)  {  List<int> intCollection = new List<int>()  {  1,2,3,2,3,4,4,5,6,3,4,5  };  var MS = intCollection.Distinct();  foreach (var item in MS)  Console.Write(item+" ");  }  }  } |

1. **Operatorul Except -** Extrage elementele care sunt în colecția 1 dar nu sunt în colecția 2.

**Exemplu:**

|  |
| --- |
| namespace LINQExcept  {  class Program  {  static void Main(string[] args)  {  List<int> dataSource1 = new List<int>() { 1, 2, 3, 4, 5, 6 };  List<int> dataSource2 = new List<int>() { 1, 3, 5, 8, 9, 10 };  var MS = dataSource1.Except(dataSource2).ToList();  foreach (var item in MS)  Console.Write(item + " ");  }  }  } |

1. **Operatorul Intersect -** Reprezintă operația inversă Except, adică identifică toate elementele care sunt prezente în ambele colecții comparate.

|  |
| --- |
| namespace LINQIntersect  {  class Program  {  static void Main()  {  List<int> dataSource1 = new List<int>() { 1, 2, 3, 4, 5, 6 };  List<int> dataSource2 = new List<int>() { 1, 3, 5, 8, 9, 10 };  var MS = dataSource1.Intersect(dataSource2).ToList();  foreach (var item in MS)  Console.Write(item + " ");  }  }  } |

#### Tipuri de requesturi LINQ

1. **LINQ către Obiecte** - include toate requesturile către obiecte interne din C# precum array-uri, liste etc. Aceste tipuri de requesturi deja vă sunt familiare.
2. **LINQ către XML(XLINQ) -** din motiv că fișierele XML sunt structuri de date, putem utiliza LINQ pentru a efectua o varietate de operații asupra acestora.

Pentru a trimite requesturi LINQ către fișiere XML **trebuie să includem librăria:**

**using System.Xml.Linq;**

Unul din cele mai simple requesturi care le putem utiliza este **citirea documentelor XML cu ajutorul LINQ**.

**Exemplu:**

|  |
| --- |
| namespace LINQToXML  {  class Program  {  static void Main()  {  XElement XMLelement = XElement.Load("exemplu2.xml");  IEnumerable<XElement> employees = XMLelement.Elements();  foreach (var employee in employees)  Console.WriteLine(employee);  }  }  } |

În exemplul de mai sus am folosit librăria XML.Linq și clasa XElement pentru a extrage toate elementele din documentul XML exemplu2.

În continuare am utilizat interfața IEnumerable pentru a transforma toate elementele XML într-o colecție structurată de date pentru ca în final să putem să iterăm prin ele cu ajutorul unui ciclu foreach.

Probleme pentru rezolvare.

### **Program 1:**

1. Folosiți exemplul de mai sus pentru a afișa toate tranzacțiile(din problema de la lecția precedentă).
2. Afișați numărul tranzacțiilor.

**Accesarea unui singur element dintr-un document XML**.

Pe baza principiului folosit în exemplul de mai sus putem folosi un ciclu pentru a itera prin toate produsele și a găsi un anumit câmp cu care vrem să efectuăm o operație.

**Exemplu:**

|  |
| --- |
| namespace LINQToXMLSingle  {  class Program  {  static void Main()  {  XElement xelement = XElement.Load("exemplu2.xml");  IEnumerable<XElement> employees = xelement.Elements();  Console.WriteLine("Pretul tuturor produselor:");  foreach (var employee in employees)  Console.WriteLine(employee.Element("Produs\_pret").Value);  }  }  } |

În exemplul de mai sus am scopul este să extragem prețul fiecărui produs, ceea ce putem face cu ajutorul proprietății **.Element** după care indicăm numele string am câmpului de care avem nevoie și, în final, folosim proprietatea **.Value** pentru a obține valoarea câmpului.

Probleme pentru rezolvare.

### **Program 1:**

1. Folosiți exemplul de mai sus pentru a afișa ID-ul tranzacțiilor cu suma mai mare de X lei.
2. Afișați din lista finală doar tranzacțiile din anul 2019.

**Accesarea unui element dintr-un document XML după atribut.**

În cazurile în care avem nevoie să accesăm unul sau mai multe elemente după atributele incluse în acestea putem folosi proprietatea Attribute pentru a extrage atributul unui anumit element.

|  |
| --- |
| namespace LINQToXMLAtribut  {  class Program  {  static void Main()  {  XElement xelement = XElement.Load("exemplu2.xml");  var enumerare = from element in xelement.Elements("Produs")  where (string)element.Attribute("categorie") == "Igienic"  select element;  Console.WriteLine("Produs cu atribut exemplu:\n");  foreach (XElement element in enumerare)  Console.WriteLine(element);  }  }  } |

**Exemplu cu lista:**

|  |
| --- |
| namespace LINQToXMLAtribut  {  class Program  {  static void Main()  {  XElement xelement = XElement.Load("exemplu2.xml");  var enumerare = from element in xelement.Elements("Produs")  where (string)element.Attribute("categorie") == "Igienic"  select Convert.ToInt32(element.Element("Produs\_pret").Value);  List<int> suma = enumerare.ToList();  Console.WriteLine(suma.Sum());  }  }  } |

Probleme pentru rezolvare.

### **Program 1:**

1. În documentul XML cu tranzacții adăugați la unele tranzacții un atribut tipTranzacție cu valorile PersoanăFizică, PersoanăJuridică.
2. Extrageți informația despre toate transferurile persoanelor fizice. Calculați suma tranzacțiilor de acest tip.
3. Extrageți informația despre toate transferurile persoanelor juridice. Calculați suma tranzacțiilor de acest tip.
4. **LINQ către SQL (DLINQ)** - crează o legătură relațională dintre variabilele programului și o valoare din baza de date. Prin acest principiu folosi LINQ pentru a accesa valorile de pe serverul în care se conține baza de date.

**Accesarea unui singur element dintr-un document XML**.

Pe baza principiului folosit în exemplul de mai sus putem folosi un ciclu pentru a itera prin toate produsele și a găsi un anumit câmp cu care vrem să efectuăm o operație.

**Exemplu:**