CTI - Dezvoltarea Aplicatiilor Web – Laborator 1
Introducere. Mediul de lucru. Extension Methods. Delegates.

Obiective laborator

- Familiarizarea cu mediul de lucru Visual Studio
- Reamintirea convențiilor de cod C#
- Pregătirea bazei pentru laboratoarele următoare
  - *Extension Methods*
  - *Delegates*
- Exerciții laborator – familiarizarea cu aplicații consolă C#

Mediul de lucru - Visual Studio

- Crearea unui proiect de tip Console Application (.NET)
- Structura unui proiect C#
- Fișierul Program.cs
- Rularea aplicației

Demo: creare proiect consolă și rulare exemplu simplu.

Getting started:
https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/
https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-console?view=visualstudio

Convenții de cod C#

Precum în orice limbaj de programare, există convenții atât pentru denumirea identificatorilor, cât și pentru scrierea unui cod cât mai consistent.

Spre deosebire de limbajul C, unde aceste convenții nu sunt impuse de standard și pot varia de la o bibliotecă sau organizație la alta, C# este un limbaj dezvoltat de Microsoft, iar convențiile oficiale reprezintă standardul.

Familiarizarea cu aceste convenții încă de la început este esențială pentru scrierea unui cod lizibil, coerent și aliniat standardelor limbajului.

Coding conventions:
https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions
https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/identifier-names

Extension Methods

Extension methods permit adăugarea de metode noi unor tipuri existente (inclusiv tipuri din .NET) fără a modifica tipul original și fără a folosi moștenirea. LINQ se bazează masiv pe acest mecanism: `Where(), Select(), OrderBy()` sunt metode de extensie definite pe `IEnumerable<T>`. Vom explora în detaliu LINQ în cadrul laboratorului 3.

Un *extension method* poate fi definit doar intr-o clasă `static`, este la rândul lui `static`, iar primul parametru indică tipul extins și este precedat de cuvântul cheie `this`. Din punct de vedere al utilizării, metoda se apelează ca și cum ar fi o metodă de instanță.

Extension methods sunt utile pentru adăugarea funcționalităților auxiliare pentru string-uri, colecții și tipuri primitive, pentru crearea unui API fluent, precum și pentru implementarea modelului utilizat de LINQ.

Exemplu metodă de extensie pentru clasa `string`, pentru a număra cuvintele dintr-un text:

```
public static class StringExtensions
{
    public static int WordCount(this string str)
    {
        return str.Split(' ', StringSplitOptions.RemoveEmptyEntries).Length;
    }
}

// Utilizare
var text = "ana are mere";
int count = text.WordCount();
```

Documentație oficială Microsoft pentru metode de extensie:
https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods

Delegates

Un `delegate` este un tip care poate face referire la o metodă, permițând transmiterea unei funcții ca parametru sau stocarea unei metode într-o variabilă. El definește semnătura unei funcții, adică tipurile parametrilor și tipul valorii returnate, și poate fi asociat doar cu metode compatibile cu această semnătură.

Deși este similar conceptual cu un *function pointer*, delegate-ul este complet type-safe.

În C# modern, delegații sunt utilizați frecvent împreună cu expresii lambda, iar utilitatea lor se regăsește în LINQ, mecanisme de callback, evenimente și în programarea asincronă bazată pe `async / await`.

Exemplu

```
public delegate int Transformer(int x);

static int Square(int x)
{
    return x * x;
}

Transformer t = Square;
int result = t(5); // 25
```

Documentație oficială Microsoft pentru delegates:
https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/

Func și Action

`Func<>` și `Action<>` sunt delegați generici predefiniți care elimină necesitatea definirii unor delegați personalizați.

`Action<T1, …>` reprezintă o funcție care nu returnează nicio valoare (`void`) și primește unul sau mai mulți parametri de tipurile specificate.

`Func<T1, …, TResult>` reprezintă o funcție care returnează o valoare de tip `TResult` și primește unul sau mai mulți parametri de tipurile indicate.

În practică, `Action<>` este utilizat atunci când dorim să efectuăm o acțiune folosind parametrii primiți de delegat, fără a produce un rezultat, în timp ce `Func<>` este utilizat atunci când dorim să evaluăm o expresie sau să calculăm o valoare care va fi ulterior utilizată în program.

De asemenea, `Predicate<T>` este un delegat predefinit care reprezintă o funcție ce primește un singur parametru de tip `T` și returnează o valoare booleană. Acesta este utilizat pentru exprimarea condițiilor logice și apare frecvent în operații de filtrare, fiind echivalent conceptual cu un `Func<T, bool>`, dar mai explicit din punct de vedere semantic.

În exemplul de mai jos sunt ilustrate utilizări pentru delegații predefiniți `Action<>` și `Func<>`. `Action<string>` este folosit pentru a executa o acțiune care primește un parametru de tip `string` și nu returnează nicio valoare, în timp ce `Func<int, bool>` este utilizat pentru a evalua o condiție, returnând o valoare booleană.

```
Action<string> log = s => Console.WriteLine(s);
log("Hello");

Func<int, bool> isEven = x => x % 2 == 0;
bool ok = isEven(10); // true
```

Documentație oficială Microsoft pentru Func, Action, și Predicate:
https://learn.microsoft.com/en-us/dotnet/standard/delegates-lambdas

Exerciții

1. Creați o metodă de extensie pentru tipul `string` care verifică dacă un text este palindrom. **(1p)**
2. Scrieți o funcție care primește ca parametru o listă de numere și o funcție de tip `Func<int, bool>` și returnează doar elementele care respecta condiția. **(1p)**
3. Creați o metodă care primește ca parametru un `string` și un delegat de tip `Predicate<string>` și returnează `true` sau `false` în funcție de rezultatul evaluării. Testați metoda folosind o expresie lambda care verifică dacă lungimea textului este mai mare decât 5. **(1p)**
4. Scrieți o funcție care primește o listă de numere întregi și o funcție de tip `Func<int, int>`, și returnează o listă nouă obținută prin aplicarea funcției primite asupra fiecărui element din listă. **(1p)**
5. Creați o metodă de extensie pentru tipul `List<int>` care returnează suma elementelor pare din listă. **(1p)**
6. Scrieți o metodă care primește o listă de string-uri și o acțiune de tip `Action<string>`, și aplică acțiunea pentru fiecare element din listă. Testați metoda folosind o acțiune care afișează elementele la consolă. **(1p)**
7. Implementați un mecanism simplu de notificare pentru utilizatori, folosind *delegates* și *extension methods*.
   Se consideră un tip de date `User`, care conține cel puțin proprietățile `Name` și `Age`. La înregistrarea unui utilizator (signup), mai multe componente independente trebuie notificate.
   a. Definiți clasa `User`, care conține proprietățile `Name` și `Age` și utilizează un *delegate multicast* de tip `Action<User>` pentru notificare. Clasa trebuie să expună metodele `Subscribe()` și `Unsubscribe()` pentru gestionarea observatorilor. **(1p)**
   b. Definiți o clasă `SignupObservers` cu cel puțin două metode statice care vor acționa ca observatori (de exemplu, trimiterea unui email și logarea evenimentului). **(1p)**
   c. Atașați observatorii delegate-ului multicast folosind metoda `Subscribe()`, notificați utilizatorul la signup, apoi eliminați un observator folosind `Unsubscribe()` și demonstrați efectul asupra notificării. **(1p)**
   d. Definiți clasa statică `UserExtensions`. Implementați o metodă de extensie `Display()` pentru afișarea informațiilor unui utilizator. **(1p)**
   e. În cadrul clasei `UserExtensions`, implementați o metodă de extensie `NotifyAll()` pentru notificarea tuturor utilizatorilor dintr-o colecție `List<User>`. **(1p)**
   f. Iterați lista de delegați folosind `GetInvocationList()` și apelați fiecare handler separat. Ce avantaje oferă această abordare față de apelarea directă a delegate-ului multicast? **(1p)**

Extra docs

Tour of C#:

https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Tipuri de date:

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types

Console I/O:

https://learn.microsoft.com/en-us/dotnet/api/system.console

Array-uri:

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/arrays

Enum-uri:

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum

Clase și OOP:

https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/