**Anonymní metody**

Anonymní metody se definují jako metody, které nemají jméno a nemají na sebe žádný indikátor -> Nemůžeme je přes něj volat.

Action line = () => Console.WriteLine();

Func<int, int, bool> testForEquality = (x, y) => x == y;

var IncrementBy = (int source, int increment = 1) => source + increment;

Console.WriteLine(IncrementBy(5)); // 6

Console.WriteLine(IncrementBy(5, 2)); // 7 testForEquality = (x, y) => x == y;

# Lambda (anonymní metoda)

Lambda je kus kódu, který na vstup dostane proměnnou a další vrátí. Nemusí při tom volat metodu, oni sami jsou jim podobné, akorát nemají jméno.

Lambdy mají určitá pravidla -> Nemohou uvnitř obsahovat proměnné, podmínky či cykly. Mohou být psány více způsoby.

Lambda je v c# řešena pomocí knihovny Linq.

Pro uchování lambda funkce v proměnné potřebujeme importovat System.Linq.

Lambda výraz se dělí na dva typy:

1. Expression Lambda: Tvořena inputem a výrazem.
   1. Příklad:
      1. var square = numbers.Select(x => x \* x);
   2. Syntaxe:
      1. input => expression;
2. Statement Lambda: Tvořena inputem a více příkazy které budou vykonány
   1. Příklad:
      1. Action<string> greet = name =>

{

string greeting = $"Hello {name}!";

Console.WriteLine(greeting);

};

greet("World");

// Hello World!

* 1. Syntaxe:
     1. input => { statements };

// C# program to illustrate the

// Lambda Expression

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**namespace** Lambda\_Expressions {

**class** Program {

**static** **void** Main(**string**[] args)

    {

        // List to store numbers

        List<**int**> numbers = **new** List<**int**>() {36, 71, 12,

                             15, 29, 18, 27, 17, 9, 34};

        // foreach loop to display the list

        Console.Write("The list : ");

**foreach**(**var** value **in** numbers)

        {

            Console.Write("{0} ", value);

        }

        Console.WriteLine();

        // Using lambda expression

        // to calculate square of

        // each value in the list

**var** square = numbers.Select(x => x \* x);

        // foreach loop to display squares

        Console.Write("Squares : ");

**foreach**(**var** value **in** square)

        {

            Console.Write("{0} ", value);

        }

        Console.WriteLine();

        // Using Lambda expression to

        // find all numbers in the list

        // divisible by 3

        List<**int**> divBy3 = numbers.FindAll(x => (x % 3) == 0);

        // foreach loop to display divBy3

        Console.Write("Numbers Divisible by 3 : ");

**foreach**(**var** value **in** divBy3)

        {

            Console.Write("{0} ", value);

        }

        Console.WriteLine();

    }

}

}

**Proč lambdu použít?**

Pomáhají filtrovat a získávat data z kolekcí bez zbytečně dlouhého kódu. Jsou převážně krátké a jednoduché na pochopení.

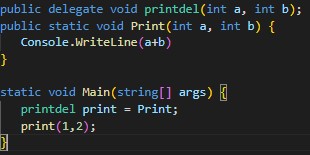
Pozor, nejsou vždy rychlejší! Mohou být i pomalejší.

Můžeme díky nim splnit určitý interface.

# Delegát

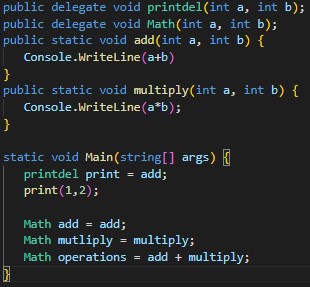
Delegát by se dal popsat jako referenční datový typ. Díky němu můžeme odkazovat na nějakou metodu skrze proměnnou. U delegátů obecně se řeší tři kroky – deklarace, nastavení reference a vyvolávání delegáta. Díky delegátům dosahujeme čistšího kódu, protože můžeme vytvářet anonymní metody a můžeme je využít v eventech.

Abychom v C# použili delegáta, používáme klíčové slovo delegate. Když ho deklarujeme, stanovujeme, jaký typ metody do něj můžeme ukládat. Metody mají Method signature – jaké vstupní a výstupní data.



V tomto případě máme proměnnou, která ukazuje na funkci.

V C# můžeme udělat multicast – Do jedné proměnné můžeme vložit více metod.



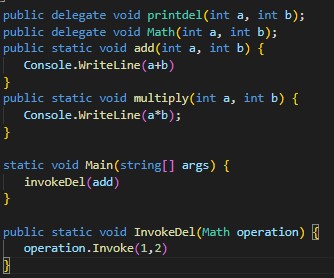
Přes multicast si vytvoříme dva delegáty a poté je spojíme do jednoho. Multicast můžeme volat různými způsoby, třeba operations(1,2), nebo operations.Invoke(1,2).

U multicastu můžeme dokonce i odebírat během běhu programu metody. Můžeme tedy udělat

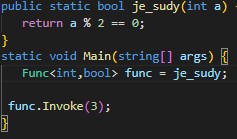
operations -= multiply

A díky tomu odebereme metodu z delegáta. Příklad může být tlačítko, která zabrání sbírání dat. Klikneme na něj a v programu se odebere metoda z delegáta o sbírání dat.

Delegáty můžeme dávat i jako vstupní hodnoty do metod.



**Func** – Přijímá metody, všechny kromě voidu (0-16 vstupních parametrů) Jako první je parametr na vstupu, jako poslední je return typ.



Action – Je void

Predicate – Přijímá jenom bool a jeden parametr a vrací jenom True a False

**Anonymní metody aka Lambda**

Odkaz od Jindry: <https://itnext.io/delegates-anonymous-methods-and-lambda-expressions-5ea4e56bbd05>

Anonymní metody používají delegáty (převážně slovíčko **Func<>**, nebo když nechceme výstup, tak se používá **Action<>**)

Deklaruje se takto:

**Func<>** jméno **=** **(vstupní proměnné) => chtěná akce**

také dokáže mít při deklaraci tělíčko pro rozsáhlejší akce

**Func<>** jméno **= (vstupní proměnné) =>**

**{**

váš kód

**}**

Dělá se skrz anonymní metody LINQ

Výraz (**a) => a \* a** je tzv. **Lambda výraz.** Tento **“=>”** se nazývá **Lambda operátor.** V tomto příkladu je **a** vstupní parametr k anonymní funkci a vrací jako výstupní hodnotu **a \* a**.

Takže když je zavolán ObsahCtverce se vstupní proměnnou **6**, tak nám to vrátí číslo **36.**

**Obsah obrázku text

Popis byl vytvořen automaticky**

Jak zde můžeme vidět **Func<int, int>**, první **int** je **vstupní** proměnná a druhý **int** (vždy ten poslední) je **výstupní** proměnná

Anonymní metoda se dá rozepsat i do tělíčka, jak bylo zmiňováno předtím. Použiju jako příklad obsah obdelníka. Ten má 2 vstupy a jeden výstup, Takže se **Func** **musí** **zvětšit** o jednu vstupní proměnnou.

První je zjednodušený zápis a druhý je zápis s tělíčkem metody, můžete si všimnout, že se výsledky nemění, páč je to stejné

Obsah obrázku text

Popis byl vytvořen automaticky

Lambda operátory lze použít i při psaní **getů/setů**

Lehká ukázka třídy student:

**Obsah obrázku text

Popis byl vytvořen automaticky**

**Obsah obrázku text

Popis byl vytvořen automaticky**

Toto berte to jako dodatečný příklad s použitím **Action<>**