Podprogramy

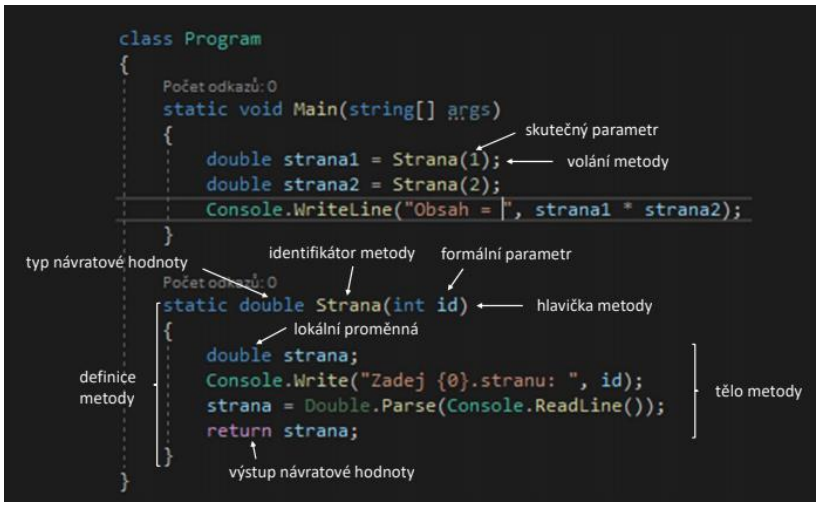
Podprogramy, funkce, delegáty, lambda operátory

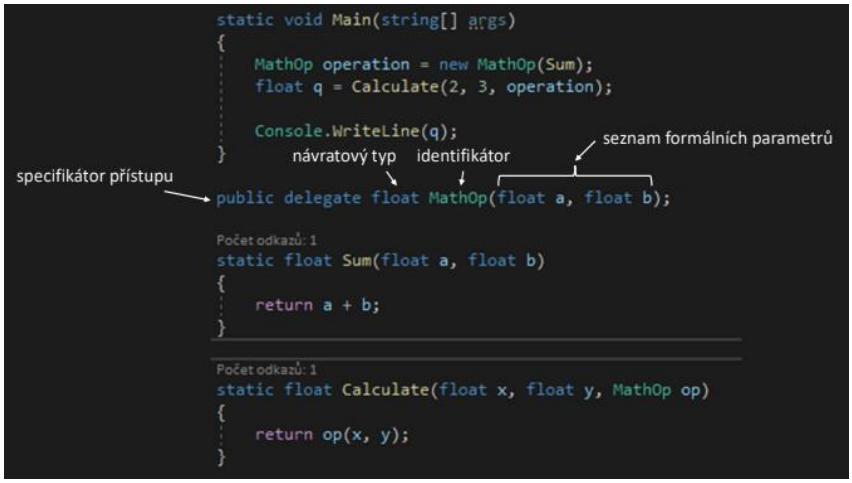
Joseph Soukup – P4

Podprogramy

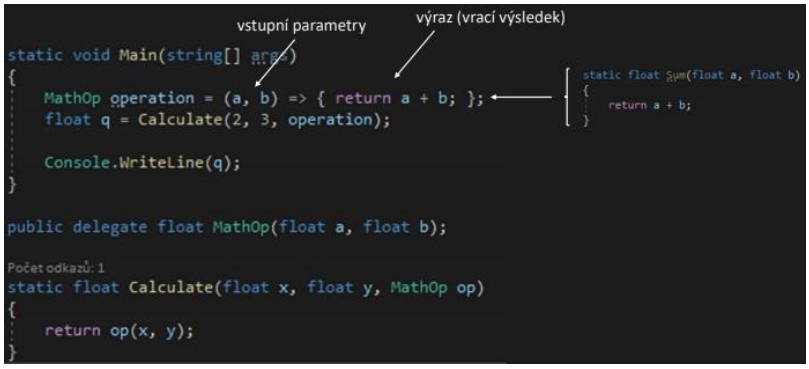
- Část počítačového programu, kterou je možné opakovaně použít v různých místech programu, i z podprogramů
- Může obsahovat parametry, které určují, s jakými daty má pracovat
- Může vracet hodnotu použitelnou při jeho vyvolání z výrazu
- V některých jazycích se rozlišují na **funkce** a **procedury**, podle toho, zda vrací hodnotu, či ne.
- V OOP se podprogramy tříd nazývají **metody**
  - ```
    public void Podprogram(string s)
    {
        s.ToLower();
    }
    ```
- v paralelním programování se používají koprogramy (komponenty, které umožňují na rozdíl od podprogramů více vstupních bodů, pozastavení a obnovení výpočtu v jejich místech)
  - ```
    var q := new queue

    coroutine produce
    loop
    while q is not full
    create some new items
    add the items to q
    yield to consume

    coroutine consume
    loop
    while q is not empty
    remove some items from q
    use the items
    yield to produce
    ```

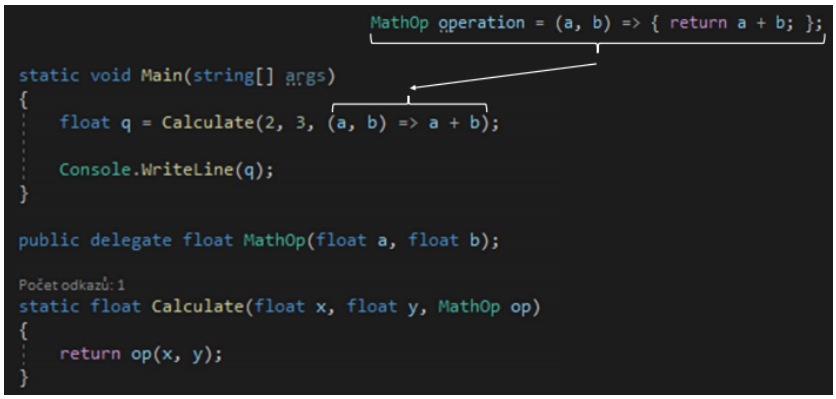
Funkce (metody)

- V OOP (C#, Java, …) - **METODY**
- V non OOP (C, Pascal, …) - **FUNKCE**
- Blok kódu, který obsahuje množinu příkazů, kterou lze volat z jiných částí programu
- Šetří paměť, zpřehledňují program
- Jsou deklarovány ve třídě, struktuře nebo rozhraní
  - ```
    static void Main(string[] args)
    {
        Vypis();
    }

    Počet odkazů: 1
    static void Vypis()
    {
        Console.WriteLine("Metoda");
    }
    ```
    ```

- 

Delegáty

- Datový typ, který představuje odkazy na metody s konkrétním seznamem parametrů a návratovým typem
- Pokud vytvoříme instanci delegátu, můžete příslušnou instanci přidružit s jakoukoli metodou s kompatibilním podpisem a návratovým typem
- Metodu můžeme vyvolat (nebo volat) prostřednictvím instance delegátu
- Používají se pro předávání metod jako argumentů jiným metodám
- Ovladače událostí nejsou nic jiného než metody, které jsou vyvolány prostřednictvím delegátů
- Můžete vytvořit vlastní metodu a konkrétní třída, jako je ovládací prvek Windows, může volat vaši metodu, pokud dojde k určité události
- Deklerace delegátu:
  - ```
    public delegate int PerformCalculation(int x, int y);
    ```
- Delegátu lze přiřadit jakoukoliv metodu z jakékoli přístupné třídy nebo struktury odpovídající typu delegátu
- Metoda může být buď statická, anebo se jedná o metodu instance
- Tato flexibilita znamená, že může programově měnit volání metod nebo připojit nový kód do existujících tříd
- Odkazovat na metodu jako parametr – ideální pro definování metod zpětného volání
  - Metoda, která porovnává dva objekty ve vaší aplikaci
  - Tuto metodu lze použít v delegátu pro algoritmus řazení
    - Vzhledem k tomu, že je kód porovnání oddělen od knihovny, může být metoda řazení obecnější
    ```

- 

Lambda operátory

- Tvoření anonymní funkce
- Pomocí **operátoru => deklarace lambda** oddělte seznam parametrů lambda od jeho těla
- Výraz lambda může mít některou z následujících dvou forem:
  - **Výraz lambda výrazu**, který má výraz jako tělo
    - `(input-parameters) => expression`
  - Výraz **lambda**, který má blok příkazu jako jeho tělo
    - `(input-parameters) => { <sequence-of-statements> }`
- Chcete-li vytvořit výraz lambda, zadejte vstupní parametry na levé straně operátoru lambda a výraz nebo blok příkazu na druhé straně
- Vycházejí z anonymních delegátů
  - Delegate (int x) {return x +1}
- Rozšiřuje delegáty
- Jednoduchý způsob, jak psát funkce, které lze předávat jako argumenty k vyhodnocení
- Lambda výraz lze přeložit
  - Do podoby kódu (jako delegát)
- Díky Expresion Tree můžeme překládat dotazy na SQL výrazy
- 

- 
MathOp operation = (a, b) => { return a + b; };
static void Main(string[] args)
{
float q = Calculate(2, 3, (a, b) => a + b);
Console.WriteLine(q);
}
public delegate float MathOp(float a, float b);
Počet odkazů: 1
static float Calculate(float x, float y, MathOp op)
{
return op(x, y);
}