PV178 – Lab 05


Kahoot!

Agenda

- Garbage Collector
- Dispose
- Delegate
- Func a Action
- Events
- Lambda
- Anonymní typy

Garbage Collector

- Proces, který běží paralelně s programem
- Automaticky spravuje paměť (heap) – alokuje/uvolňuje paměť
- O spuštění „úklidu“ rozhoduje engine garbage collectoru
- Spolupracuje s operačním systémem
- Stárá se o ni **managed memory –** objekty vytvořené přes *new*
- vs. unmanaged memory – musíme se starat my (streamy - soubory, databázové spojení apod.)
- Dá se spustit explicitně příkazem *GC.Collect* – NEDĚLEJTE !!!

Dispose

- Čištění unmanaged paměti, kterou nespravuje Garbage Collector
- Obrázky, grafika, streamy, soubory
- Proč dispose? Garbage collector nemá o zdrojích přehled – neví jestli zdroj stále někdo používá
- Když není metoda Dispose zavolána způsobí memory leak
- `Using` za nás provádí Dispose automaticky

Dispose - demo

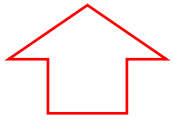
Samostatná práce

- Najděte alespoň 3 třídy, které implementují IDisposable

Delegát – *delegate*

- “předpis signatury metody – jak mají metody vypadat“
- Instance delegáta je reference (odkaz) na metodu – bezpečný ukazatel na metodu
- Umožňuje předávat metodu jako parametr jiné metodě

```
public delegate int MyDelegate(int numberA, int numberB);
```



Toto je pouze předpis, ne instance delegáta!

Delegát – příklad

```
public delegate int MyDelegate(int numberA, int numberB);


static void Main(string[] args)
{
        MyDelegate test;
        test = delegate (int a, int b) { return a + b; }; // Možnost 1
        test = (a, b) => a - b; // Možnost 2
        test = MyDivide; // Možnost 3

        test(3, 5); // Volání metody
}


public int MyDivide(int a, int b)
{
        return a / b;
}
```

Func a Action

- „Obalený delegát“
- Action
  - Delegát pro funkce bez návratové hodnoty (návratová hodnota je *void*)
  - 17 různých delegátů Action s různým počtem parametrů (Action<T1, T2 … Tx >)
- Func
  - poslední parametr udává typ návratové hodnoty
  - 17 různých delegátů Func s různým počtem parametrů (Func<T1, T2 … Tx>)

Func a Action - příklady

```
Func<string> MyFunc1 = … ;  // žádný parameter, návratová hodnota (string)

Func<int, bool> MyFunc2 = … ;  // 1 parameter (int), návratová hodnota (bool)

Func<int, string, bool> MyFunc3 = … ; // 2 parametery (int, string), návratová hodnota (bool)
```

…

```
Action MyAction1 = … ;  // žádný parametr, návratová hodnota (void)

Action<int> MyAction2 = … ;  // 1 parameter (int), návratová hodnota (void)

Action<int, float> MyAction3 = … ; // 2 parametery (int, float), návratová hodnota (void)
```

…

Lamba Výrazy

- Anonymní funkce (nemá jméno)
- Lze je přiřadit do delegáta
- Zapisuje se pomocí operátoru =>
- Používají se v LINQu


```
... => ...
```

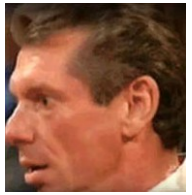
Vstupní parametry

Tělo lambda výrazu

Lamba Výrazy – příklad 2

```
numbers.Where((int i) => { return i > 52; });
```

```
numbers.Where((i) => { return i > 52; });
```



```
numbers.Where(i => { return i > 52; });
```

```
numbers.Where(i => i > 52);
```

Samostatná práce

- Úkoly v projektu DelegatesAndStuff

Lokální funkce

- Funkce uvnitř funkcí
- Méně používané
- Nemají modifikátor viditelnosti
- Znesnadňují čtení kódu

```
private void Do(int n)
{
        // Some work

        PrintToConsole("Some Text");

        void PrintToConsole(string text)
        {
                Console.WriteLine(text);
        }
}
```

Samostatná práce

- Úkoly v projektu LocalFunctions

Events

- Umožňují notifikovat objekty o změně
- Klíčové slovo event
- Používá se například když uživatel klikne do formuláře, zmáčkne tlačítko
- Na událost může reagovat vícero objektů – objekty se registrují k odběru události

Events - demo

Samostatná práce

- Úkoly v projektu Events

Anonymní typy

- Využití v LINQ (uvidíme v příštím cvičení
- Vytvořené kompilátorem (silně typované)

```
var anonymousPerson = new
{
        FullName = "Tom Smith",
        Salary = 1500
}
```

Zde může být libovolný název

```
var name = anonymousPerson.FullName;
var salary = anonymousPerson.Salary;
```