# **Konstanty, Statické metody a vlastnosti, Anonymní metody (Lambda)**

## **Statické metody a vlastnosti**

Nejprve si řekneme, co vlastně **static** znamená. Znamená to, že **nemusí** být vytvořena instance třídy(třeba *new Auto()*), aby se se dokázal static zavolat.

Když chceme zavolat static tak to jde **pouze** přes

**[jméno třídy].[jméno statické proměnné]**

Slovíčka static lze dát skoro před všechny proměnné jako jsou:

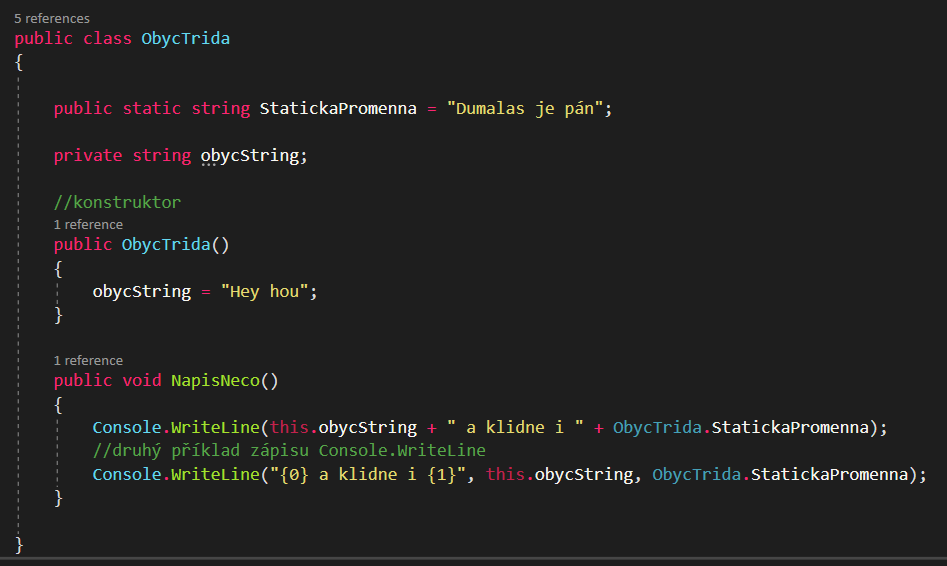
pole, metody, vlastnosti, operátory, eventy a konstruktory

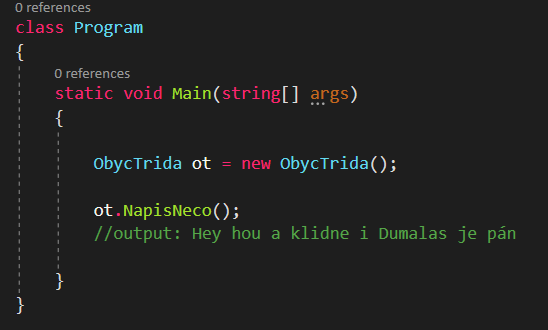
Při deklarování statické třídy, lze do třídy uplatňovat jen proměnné a metody, která jsou také static

****

Static patří ke třídě ale né k instanci třídy. Z **[jméno třídy].** nelze přistupovat k objektovým proměnným ani metodám, lze jen přistoupit k statickým

**Instance** třídy **dokáže pracovat** se statickou proměnnou i metodou, ale **naopak to** **nejde**

****



## **Konstanty**

Konstanty jsou neměnné hodnoty. Chová se jako **statická proměnná**. (Takže se volá jako JmenoTřídy**.JménoKonstanty**) Jsou známy při kompilování programu a **po celou dobu běhu programu se nemění a nedá se změnit**. Konstanty mohou být jen basic typy jako jsou (int, long, string, char,..). Nemůže se do konstant ukládat složité objekty, pole a třídy a uživatelem vytvořené entity.

Metody a ani eventy **nemohou být** konstanty

Deklaruje se slovíčkem ***const*** a pojmenovávají se většinou **velkými písmeny**

Dále se na slovíčko přibaluje:

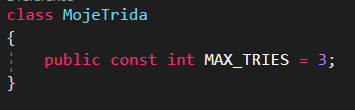
**[public/private]***const* **[datový typ]**jméno **= hodnota typu**

Konstanty se volají buďto normálně přímo ve třídě což je:

**JménoKonstanty**

nebo přes:

Třída**.JménoKonstanty**

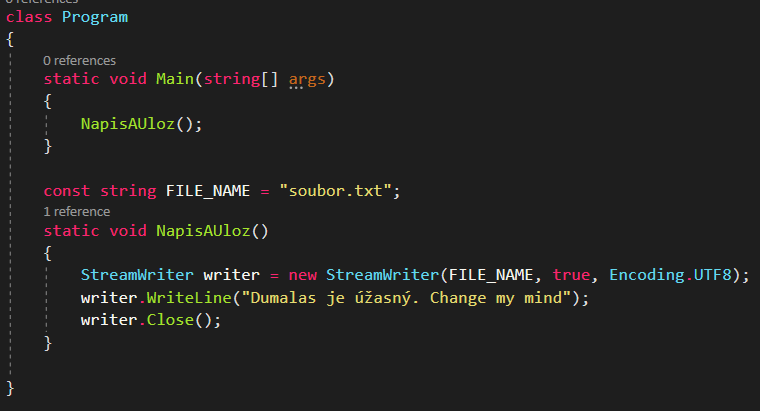


Třeba při returnu



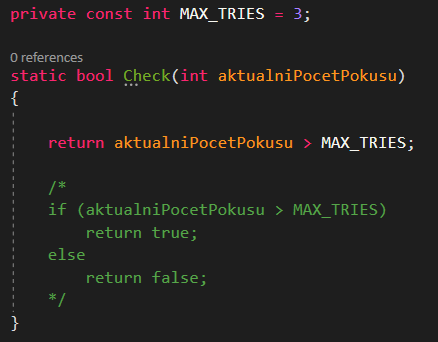
**Příklad konstant:**

*Představte si, že máte danou neměnnou cestu k souboru a chcete vždy do souboru zapsat a uložit.*



Konstantou máš danou počet maximálních pokusů (docela věc, co se nemění) a můžete začít kontrolovat

PS: lepší to je zapsat tak jak to je, ale zakomentované také není špatně



**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) =>**

**{**

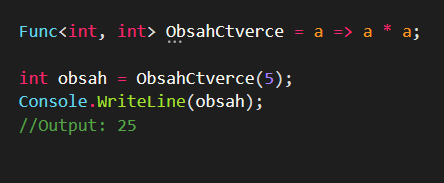
váš kód

**}**

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**.

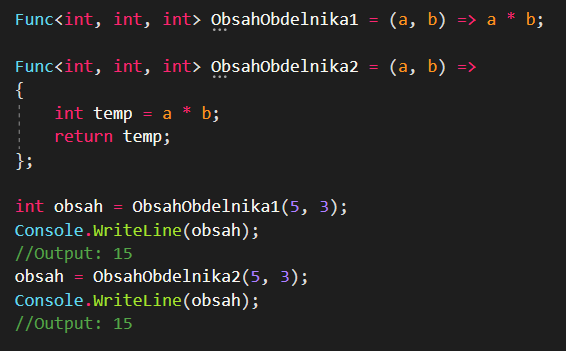
Takže když je zavolán ObsahCtverce se vstupní proměnnou **6**, tak nám to vrátí číslo **36.**

****

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

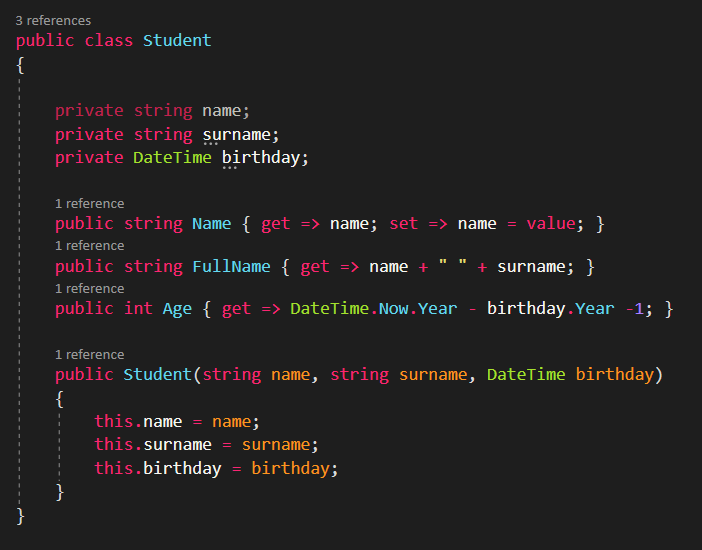
Anonymní metoda se dá rozepsat i do tělíčka, jak bylo zmiňováno předtím. Použiju jako příklad obsah obdelníka. Ten má 2 vstupy a jeden výstup, Takže se **Func** **musí** **zvětšit** o jednu vstupní proměnnou.

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



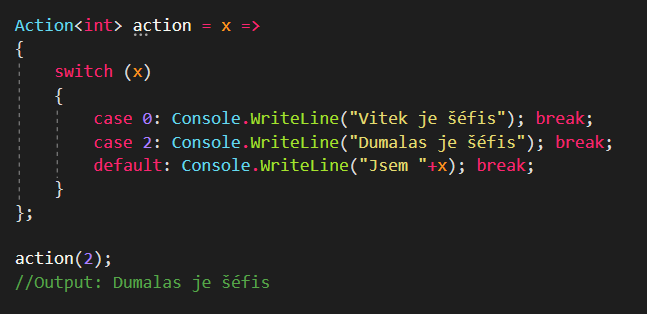
Lambda operátory lze použít i při psaní **getů/setů**

Lehká ukázka třídy student:

****

****

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

****