EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Základy programování a algoritmizace

Třídy, struktury a referenční typy

Erik Král

2020

Strategický projekt UTB ve Zlíně, reg. č. CZ.02.2.69/0.0/0.0/16_015/0002204

Informace o autorech:

Ing. et Ing. Erik Král, Ph.D.
Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Nad Stráněmi 4511
760 05 Zlín
ekral@utb.cz

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OBSAH

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

1 ÚVOD

V tomto materiálu probereme třídy, struktury, hodnotové a referenční typy.

1.1 Třída

Kromě zabudovaných typů, jako je například *int* nebo *double* můžeme pomocí klíčového slova *class* deklarovat vlastní typ – třídu [1], která může být složená z více různých typů.

Třída může mít následující členské prvky:

- Fieldy (atributy, členské proměnné)
- Metody (členské funkce)
- Konstruktor – speciální metoda sloužící k inicializaci proměnné.

Fieldy představují členské proměnné deklarované přímo ve třídě nebo struktuře [2]. Například třída Student má níže nadefinované dva fieldy, *id* a *body*. Fieldy by v reálném kódu neměli být většinou *public*, v tomto případě jde jen o ukázkový příklad.

```
class Student
{
    public int id;
    public double body;
}
```

V klientském kódu si potom můžeme nadefinovat proměnnou typu reference na třídu Student. Hodnota *null* znamená, že proměnná nemá zatím přiřazenou žádnou hodnotu.

```
Student s = null;
```

Této proměnné potom přiřadíme reference na novou instance třídy Student alokovanou na haldě pomocí operátoru *new*. Halda je část paměti RAM, která je k dispozici pro všechny programy a program může operační systém požádat o to, aby mu část této paměti exkluzivně přiřadil. Pokud bychom, například na mikropočítači, neměli operační

systém a pouze jeden program, tak bychom mohli alokovat celou paměť pro zásobník a využívat jen zásobník. V případě operačního systému s multitaskingem ale více programů, a i operační systém sdílejí jednu paměť a musí se o ni podělit.

```
s = new Student();
```

Definici reference i vytvoření instance můžeme zapsat na jeden řádek.

```
Student s = new Student();
```

K fieldům potom přistupujeme pomocí *member access* operátoru.

```
s.id = 1;
s.body = 50;

Console.WriteLine($"{s.id} {s.body}");
```

Metody představují blok příkazů, které se provedou, pokud program metodu zavolá [3]. Hlavička metody obsahuje typ návratové hodnoty, název metody a parametry metody. V následujícím kódu jsou nadefinovány tři metody *Obsah*, *Obvod*, *Zmen* a konstruktor.

```
class Obdelnik
{
    public int a;
    public int b;

    public Obdelnik(int a, int b)
    {
        this.a = a;
        this.b = b;
    }

    public int Obsah()
    {
        return a * b;
    }
```

```
    public int Obvod()
    {
        return 2 * (a + b);
    }

    public void Zmen(int a, int b)
    {
        this.a = a;
        this.b = b;
    }
}
```

Metoda *Obsah*, stejně jako metoda *Obvod* vrací jako hodnotu typ *int* a nemá žádné parametry. Hodnotu z metody vracíme pomocí klíčového slova *return*, které ukončí běh metody.

```
public int Obsah()
{
    return a * b;
}
```

Metoda *Zmen* má návratový typ *void*, to znamená, že nevrací žádnou hodnotu, ale i tak bychom mohli použít *return* bez hodnoty pro ukončení metody. Metoda má dva **parametry** *a*, *b* typu *int*.

```
public void Zmen(int a, int b)
{
    this.a = a;
    this.b = b;
}
```

Speciálním případem metody je konstruktor, který se zavolá kdykoliv, když je vytvořena instance třídy nebo struktur. Pokud není nadefinovaný žádný konstruktor, použije se implicitní konstruktor, který nastaví všechny členské prvky na výchozí hodnoty [4].

Konstruktor je metoda, jejíž název je stejný jako název třídy a nemá žádnou návratovou hodnotu.

V našem příkladu máme parametrický konstruktor, tedy konstruktor, který má dva parametry *a* a *b*. Klíčové slovo *this* představuje reference na instance třídy pro kterou byla zavolána metoda. Většinou se vynechává, ale v tomto případě pomocí klíčového slova *this* odlišujeme fieldy od parametrů, protože se jmenují stejně.

```
public Obdelnik(int a, int b)
{
    this.a = a;
    this.b = b;
}
```

Metody se volají podobně jako fieldy pomocí *member access* operátoru. V následujícím příkladu voláme všechny probrané metody.

```
Obdelnik o1 = new Obdelnik(2, 3);

o1.Zmen(5, 6);
Console.WriteLine($"a: {o1.a} b: {o1.b} obsah:{o1.Obsah()} obvod:{o1.Obvod()}");
```

Posledním typem, který probereme je **Property**, který souvisí se zapouzdřením. Zapouzdření je myšlené ve dvou významech, za prvé sloučení stavu (fieldů) a chování (metod) do jednoho typu. Druhý význam je skrývání dat, kdy skrýváme vnitřní stav objektu (fieldy), tak aby k nim mohli přistupovat pouze metody stejné třídy, ale nemohli být změněny v klientském kódu. Důležité je, že **data skrýváme před vývojáři** a cílem je zpřehlednění kódu a zabránění chybám, tak aby bylo jasné, které fieldy a metody má vývojář volat když používá třídu ve vlastním kódu a které metody používat nemá.

Pro skrývání používáme následující klíčová slova:

- `public` - přístupné z vnějšku třídy i metodám odvozené třídy.
- `private` - nepřístupné z vnějšku třídy a nepřístupné metodám odvozené třídy. Přesto je ale odvozená třída má, ale mohou k nim přistupovat pouze metody základní třídy.
- `protected` - nepřístupné z vnějšku třídy, ale přístupné metodám odvozené třídy, je to „nejdůležitější“ modifikátor v rámci dědičnosti.
- `internal` - přístupné z jakéhokoliv kódu ve stejné assembly, ale nepřístupné z jiné assembly.

My se zaměříme na klíčová slova *public* a *private*.

Nejprve si ukážeme příklad na *public*. Na levé straně je definice třídy a na pravé straně klientský kód v metodě *Main*. Pokud je field *vek* a *jmeno public*, tak k nim můžeme přistupovat v klientském kódu.

```
class Student
{
    public int vek;
    public string jmeno;
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Student student1 = new Student();
        student1.jmeno = "Tomas";
        student1.vek = 22;
    }
}
```

Pokud je field *vek* a *jmeno private*, tak k nim nemůžeme přistupovat v klientském kódu.

```
class Student
{
    private int vek;
    private string jmeno;
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Student student1 = new Student();
        student1.jmeno = "Tomas";
        student1.vek = 22;
    }
}
```

Pro přístup k private fieldům se často používají speciální *public* metody začínající slovem *Get* nebo *Set*, které slouží k přístupu k prvkům z vnějšku třídy. Tyto metody nazýváme gettery a settery. Pomocí getterů a setterů se snažíme skrýt (často i do budoucna) konkrétní implementaci. Počítáme tedy s tím, že v budoucnu můžeme například přidat ověření, že jméno není prázdný řetězec nebo logování změn.

Property představuje zabudovanou podporu pro gettery a settery přímo v jazyce C#.

V kódu se k propertám přistupuje stejně jako k fieldům. Díky tomu je kód kratší a přehlednější. V rámci .Net frameworku prakticky žádné třídy nevystavují fieldy ale vždy jen property. Díky propertám můžeme například při reflexi odlišit metody od getterů a setterů.

Property obsahuje 2 části:

- get část obsahuje implementaci getteru
- set část obsahuje implementaci setteru

Platí pravidlo, že metody *get* a *set* musí být skoro stejně rychlé jako přímý přístup k fieldu. V property bychom neměli provádět časově náročné operace jako například přístup k databázi nebo složité výpočty.

Property by neměli mít vedlejší efekty (side effects), kromě změny backing fieldu, případně logování nebo notifikaci atd. by neměli měnit stav jiných proměnných. Vyjímkou jsou například property v MVVM, kde někdy chceme zaregovat na změnu stavu property.

Na levé straně je uveden příklad getterů a seterů a na pravé straně použití property.

```
private int vek;

public int GetVek()
{
    return vek;
}

public void SetVek(int vek)
{
    this.vek = vek;
}

private string jmeno;

public string GetJmeno()
{
    return jmeno;
}

public void SetJmeno(string jmeno)
{
    this.jmeno = jmeno;
}
```

```
private int vek;

public int Vek
{
    get
    {
        return vek;
    }
    set
    {
        vek = value;
    }
}

private string jmeno;

public string Jmeno
{
    get
    {
        return jmeno;
    }
    set
    {
        jmeno = value;
    }
}
```

Property se používají stejně jako fieldy, dle jmenné konvence začínají velkým písmenem.

```
static void Main(string[] args)
{
    Student student1 = new Student();
    student1.Vek = 22;
    student1.Jmeno = "Tomas";
    string jmeno = student1.Jmeno;
    int vek = student1.Vek;
}
```

Pro triviální případy, kdy metody get a set jen zapouzdřují backing filed, je možné od C# 3.0 využít tzv. **auto-implemented property**, kdy překladač automaticky vytvoří anonymní backing field. V budoucnu, až budeme například chtít třeba v metodě set kontrolovat, zda je value platná hodnota, tak doplníme těla metod getteru a setteru.

```
class Student
{
    public int Vek { get; set; }
    public string Jmeno { get; set; }
}
```

```
static void Main(string[] args)
{
    Student student1 = new Student();
    student1.Vek = 22;
    student1.Jmeno = "Tomas";
    string jmeno = student1.Jmeno;
    int vek = student1.Vek;
}
```

1.2 Hodnotové a referenční typy

Zatím jsme si ukazovali příklad na třídu, která je referenční typ a instace si alokují paměť na haldě a paměť potom uvolňuje Garbage Collector [6]. Struktura má prakticky skoro stejnou syntaxi jako třída, ale jde o hodnotový typ, instance se tedy ukládají na zásobník.

V jazyce C# tedy rozlišujeme typy do dvou základních kategorií, **hodnotové typy** (například *int*) a **refereční typy** (například třída)

Hodnotové typy (Value types) – přiřazením se kopíruje hodnota:

- Struktury
- Výčtový typ (Enum)
- Numerické typy (int, double atd.) a bool

Referenční typy (Reference types) – přiřazením se kopíruje reference:

- Třída (class) – i pole v .NET jsou třídy
- Rozhraní (interface)
- Delegate
- String – speciální případ, referenční typ, který se chová jako hodnotový
- Pointer types – v C# se běžně nepoužívají

I hodnotové typy můžeme předávat jako reference, a to pomocí klíčového slova *ref* [7].

SEZNAM POUŽITÉ LITERATURY

[1] Classes - C# Programming Guide | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 05.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/classes

[2] Fields - C# Programming Guide | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 06.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/fields

[3] Methods - C# Programming Guide | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 06.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods

[4] Constructors - C# Programming Guide | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 06.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constructors

[5] Properties - C# Programming Guide | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 06.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/properties

[6] Fundamentals of garbage collection | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 06.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals

[7] ref keyword - C# Reference | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 06.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/ref

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY