LEKCIA 3: ZLOŽKY TRIED

Vieme že trieda sa v princípe skladá z dát (**fields**) a operácií (**methods**). To je však veľmi hrubozrnné delenie.

ČLENY TŘÍDY

- metody (methods),
- datové položky (fields) a konstanty,
- vlastnosti (properties),
- události (events),
- indexery (indexers)

VLASTNOSTI

Vlastnosti (*properties*) je hybridem mezi datovou položkou a metodou. Pracuje se s nimi jako s datovou položkou, ale pro čtení hodnoty i její nastavení jsou volány metody (tzn. přístupové metody).

V C# se vlastnosti zapisují podobně jako datové položky, ale za jejich jménem následuje ještě definice přístupových metod pomocí klíčových slov `get` a `set`. Přístupové metodě `set` je nová hodnota vlastnosti předána pomocí klíčového slova `value`.

```
private int age;
public int Age {
  get { return age; }
  set { age = value; }
}
```

```
dog.Age = 10;  // nastavení věku pomocí vlastnosti
dog.Age += 7; vs dog.SetAge(dog.GetAge() + 7);   // srovnání vlastností a tradičních přístpových metod
```

Vlastnost může obsahovat jen přístupovou metodu `get` nebo `set`, taková vlastnost je pak jen pro čtení nebo jen pro zápis. U každé přístupové metody lze take specifikovat vlastní modifikátor přístupu. Jmenná konvence říká, že názvy vlastností se píší PascalCase, a názvy datových položek nesoucích hodnotu vlastnosti jsou camelCase.

Pokud vlastnost pouze zapozdřuje datovou položku bez dalších kontrol nebo výpočtů (jako `Age` výše), lze ji zapsat zkráceně jako *automaticky implementované vlatnost*. Přístupové metody takových vlastností nemají uvedena těla, protože kompilátor si je i s datovou položkou vytvoří sám. Aby taková vlastnost měla smysl musí mít obě přístupové metody. Stále na ně lze aplikovat modifikátor přístupu zvlášť.

```
public int Age { get; private set; } // automatická vlastnost, jen ke čtení
```

INDEXERY

Indexery dovolují třídě používat hranaté závorky `[]` podobně jako mají pole. Indexer je konstrukce podobná vlastnostem, ale její název je `this` a po něm následují parametry indexeru. S indexery se můžete setkat o hlavně u kolekcí (hranaté závorky u pole nejsou nic speciálního – to je taky indexer).

MODIFIKÁTORY PRÍTUPU

Pre uplatnenie princípu zapuzdrenia využijeme tzv. modifikátory prístupu.

- **public -** bez obmedzenia viditeľnosti a prístupu,
- **protected -** prístup obmedzený na triedu a jej podtriedy,
- **internal -** prístup obmedzený na danú assembly,
- **private -** prístup obmedzený na danú triedu.

Popis na MSDN: Access Modifiers

MODIFIKÁTORY

MODIFIKÁTORY PARAMETRŮ

V C# existují 3 modifikátory parametrů. Modifikátor `ref` vynutí předání parametru odkazem místo předání hodnotou. Pokud se parametr předaný odkazem v metodě změní dojde ke změně objektu ve volající metodě. Obdobně funguje modifikátor `out`, ten slouží k navracení hodnot z metody a kompilátor vás donutí do takového parametru něco přiřadit. Pro oba tyto modifikátory platí, že je musíte uvéset při deklaraci parametru i v místě použití).

```
int number;
string str = Console.ReadLine();
if( int.TryParse(str, out number) ) {
  Console.WriteLine("Double of the number {0} is {1}", number, number * 2);
}
else {
  Console.WriteLine("{0} is not a number", str);
}
```

VIRTUÁLNE ČLENY A OVERRIDE

Kľúčové slovo `virtual` označuje člena, ktorého može dediaca trieda prepísať (`override`). Používa sa na implementovanie polymorfizmu v jazyku C#.

```
public class Animal {
    public virtual void PerformTrick() {
        //animal trick
    }
}
```

Klúčovým slovom `base` môžeme zavolať z potomka pôvodnú implementáciu predka.

```
public class Dog : Animal {
    public override void PerformTrick() {
        //další trik, ktorý vie iba pes
        base.PerformTrick(); // animal trick
    }
}
```

Opakem `override` je kličové slovo `new`, které kompilátoru říká, že metoda nemá s metodou v bázové třídě nic společného a zabraňuje polymorfismu, resp. vyjadřuje úmysl programátora, že nechce metodu překrýt.

STATICKÉ ČLENY

Klíčové slovo `static` popisuje člen třídy, který je společný všem instancím. Ke statickým členům lze přistupovat jen pomocí jména typu, nikoliv přes instanci.

Datové typy mohou mít statický konstruktor, který je zavolán před prvním použitím typu. Statický konstruktor má na rozdíl od konstruktoru instance před názvem klíčové slovo `static` a nemá modifikátor viditelnost. Pokud třída obsahuje jen statické členy, je možné ji celou označit klíčovým slovem `static`. U takové třídy pak kompilátor zajistí, že z ní nelze dědit, nelze vytvořit její instanci a neobsahuje žádné nestatické členy. Například třídy `Console` nebo `Math` jsou statické třídy.

ROZŠIŘUJÍCÍ METODY (EXTENSION METHODS)

Rozšiřující metody dovolují rozšířit existující třídu o nové metody bez zásahu do původní třídy nebo dědičnosti. Ve skutečnosti jde o další syntaktický cukr kompilátoru: rozšiřující metody jsou statické metody, které kompilátor dovoluje volat na jiné třídě stejným zápisem jako metody rozšiřované třídy – v postfixové notaci.

Rozšiřující metody se deklaruje téměř stejně jako obyčejná statická metoda, jen první parametr metody musí mít modifikátor `this` a určuje třídu, která je rozšiřována. Do prvního parametru je pak dosazen objekt, na kterém se metoda volá. Rozšižřující metody musí být ve statické třídě a pro použití musí být importován příslušný jmenný prostor deklarací `using`.

```
public static class Int32Utils
{
  public static bool IsEven(this int number) { return i % 2 == 0; }
}

int i = 10;
bool isEven1 = i.IsEven();           // volání IsEven jako rozšiřující metody
bool isEven2 = Int32Utils.IsEven(i); // volání IsEven jako statické metody
```

SEALED A PARTIAL

Trieda môže byť definovaná ako `sealed` čo znamená že iné triedy z nej nemôžu dediť (chceme zabrániť rozširovaniu, zneužitiu dedičnosti, zvýšiť bezpečnosť alebo výkon). Napr. String je `sealed`.

```
public class MyString : System.String
{
  //chyba!
}
```

Často narazíte na klúčové slovo `partial` u definície triedy. Takáto trieda môže byť definovaná vo viacerých súboroch *.cs v jednom projekte. Väčšinou sa jedná o triedy generované nástrojmi.

KONSTANTY A READONLY DATOVÉ POLOŽKY

Konstanta je datová položka nebo proměnná, jejíž hodnotu nelze změnit. Značena je klíčovým slovem `const` a je automaticky statická. Konstanta může být pouze jednoho ze vestavěných typů (vč. řetězců a `null`) a její hodnota musí být známá už při kompilaci.

Pro datovou položku lze použít i klíčové slovo `readonly`, které zajistí, že hodnota je do ní přiřazena jen jednou a po té do ní nelze znovu přiřadit. Přiřazení lze provést jen při její inicializaci, nebo v konstruktoru třídy, ve které je deklarována. Datové položky s modifikátorem `readonly` nejsou automaticky statické.

```
private const int SquirrelLimitPerPark = 200;
private static readonly DateTime TypeCreated = DateTime.Now;
private readonly IList<string> squirrelNames = new List<string> { "Chip", "Dale" };
squirrelNames = null;      // chyba kompilace; datová položka už byla nastavena
squirrelNames.Clear();     // nicméně tady pouze měním stav, takže to udělat lze
```

DELEGÁTI A UDÁLOSTI (DELEGATES AND EVENTS)

Datový typ *delegát* je třída, která obsahuje odkaz na metodu. Tuto metodu pak lze prostřednictvím delegáta zavolat. V případě nestatických metod si delegáti pamatují i objekt, na kterém mají metodu vyvolat.

Delegát se deklaruje použitím klíčového slova `delegate` s popisem návratového typu a parametrů (podpisu) stejně jako u metody. Do delegáta je možné přiřadit jen metody, jejich podpisu se shoduje s podpisem delegáta.

Delegáti mají konstruktor, který přebírá název metody, ale kompilátor dovoluje přiřadit do delegáta pouze název metody a konstruktor si domyslí sám.

```
public delegate int BinaryOperationDelegate(int left, int right);            // deklarace delegáta
public static int Add(int left, int right) { return left + right; }          // metoda
BinaryOperationDelegate delegateVariable = new BinaryOperationDelegate(Add); // vytvoření delegáta
BinaryOperationDelegate delegateVariable = Add;                              // zkrácené vytvoření
int sum = delegateVariable(1, 2);                          // vyvolání delegáta s parametry 1 a 2
```

Delegát, do kterého nebyla přiřazena žádná metoda, má hodnotu `null`. To představuje mírnou komplikaci, protože vyvolání prázného delegátu vyvolá výjimku `NullReferenceException` a je tedy potřeba provádět kontrolu.

Jeden delegát může vyvolat více metod a delegáty lze kombinovat. V případě vice metod se metody volají postupně a návratová hodnota delegáta odpovídá výsledku poslední metody. Delegáti se kombinují pomocí operátorů + a -, v praxi se ovšem nejčastěji používají nepřímo jako operátory += a -= pro přidávání a odebírání metody k/z delegáta.

GENERIČTÍ DELEGÁTI ACTION A FUNC

V BCL existuje několik delegátů, za zmínku stojí delegáti `Action` a `Func`. Action je delegát pro funkce bez návratové hodnoty a ve skutečnosti existuje 17 různých delegátů Action s různým počtem generických parametrů: `Action`, `Action<T>`, `Action<T1, T2>`, … Podobně je na tom delegát `Func`, u kterého poslední typový parameter určuje návratovou hodnotu. Jejich nevýhodou je příliš obecný název, který nevystihuje jejich funkci.

ANONYMNÍ METODY A LAMBDA VÝRAZY

Do delegátů lze přiřadit existující metodu, v mnoha případech taková metoda existuje jen kvůli jednomu konkrétnímu delegátu a její existence zbytečně znepřehleňuje třídu. Proto existují 2 typy *anonymních funkcí*, pomocí kterých lze zapsat metodu přímo na řádku přiřazení do delegáta.

Prvním typem jsou *anonymní metody*, které se deklarují pomocí klíčového slova `delegate`.

```
BinaryOperationDelegate add = delegate(int a, int b) { return a + b; }
```

Jejich zjednodušením pak do C# dostaly *lambda výrazy*, které se zapisují pomocí operátoru `=>`.

```
BinaryOperationDelegate add2 = (int a, int b) => { return a + b };
BinaryOperationDelegate add3 = (a, b) => a + b;
Func<int, bool> isEven = n => n % 2 == 0;
Func<DateTime> getCurrentDate = () => DateTime.Now;
```

Pokud má tělo lambda výrazu jednu operaci, lze vynechat složené závorky i klíčové slovo `return`. Typy parametrů si kompilátor dokáže odvodit z kontextu a v případě jednoho parametru je možné vynechat i kulaté závorky.

When to Use Delegates Instead of Interfaces (C# Programming Guide)

UDÁLOSTI (EVENTS)

Událost je mechanizmus komunikace mezi dvěma a více objekty. Jeden objekt (*vydavatel*, *publisher*) informuje o nějaké události druhý objekt (*odběratel*, *subscriber*). Typickým příkladem je kliknutí na tlačítko v aplikaci. Tlačítko (*publisher*) informuje aplikaci (*subscriber*), že na něj bylo kliknuto a aplikace tak na kliknutí může reagovat.

Události v C# jsou implementovány pomocí delegátů, ale dále je omezují (zapouzdřují). K události se lze pouze registrovat nebo odregistrovat pomocí operátorů `+=` a `-=`. Přiřazení a vyvolání události je ale mimo deklarující třídu nemožné (logicky chceme, aby událost vyvolal pouze ten, kdo ji deklaroval).

Deklarace události začíná klíčovým slovem `event`, následovaným typem delegáta a názvem události.

```
public event EventHandler Click;
```

Pro události v C# existuje konvence, který říká, že delegát události by měl mít dva parametry: prvním parameterem je objekt, který událost vyvolal a druhým je třída `EventArgs` nebo z ní odvozená třída, pomocí které se předají metodám registrovaným k události potřebné informace. V BCL existují dva typy delegátů, které se s obvyjke s událostmi používají: `EventHandler` a `EventHandler<TEventArgs>`.

Je také zvykem vytvořit ke každé události chráněnou virtuální metodu, která ji vyvolá s názvem On*NázevUdálosti*.

```
protected virtual OnClick() {
    EventHandler handler = Click;
    if( handler != null )
        handler (this, EventArgs.Empty);
}
```

Events and Delegates, Consuming Events, Raising an Event