Struktura a třída (teorie OOP)

Úvod

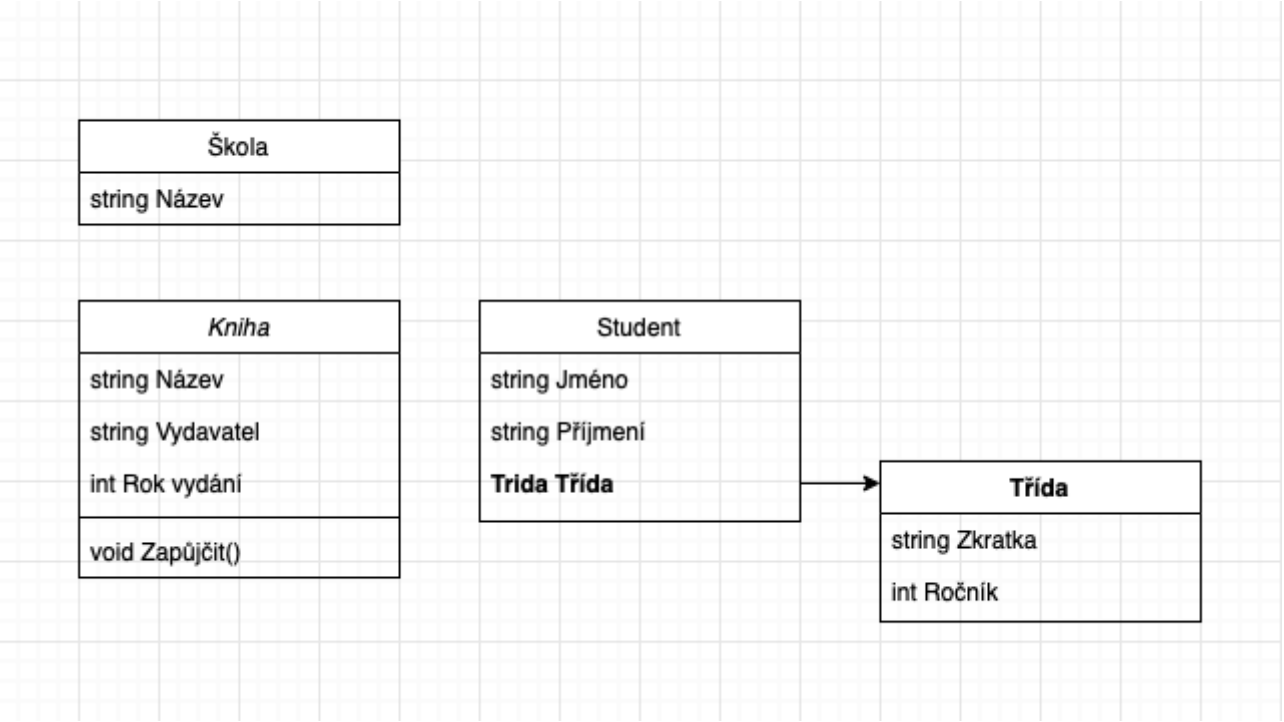
Jazyk C# je stejně jako mnoho ostatních programovacích jazyků založen na objektech, které jsou jeho základními stavebními kameny. Je proto velmi důležité chápat takto jazyk už od samého začátku. Co to ale ten objekt vlastně je? Než si na tuto a řadu dalších otázek odpovíme, uděláme si pro lepší pochopení látky malý výlet do historie a ujasníme si některá fakta.

Objektově orientované programování (OOP)

Nikoho asi nepřekvapí, že vývoj softwaru je velmi mladé odvětví. Porovnejme si ho například se stavebnictvím. Domy stavíme již tisíce let, naproti tomu programovat jsme začali teprve nedávno. S tím souvisí i fakt, že do teď pořádně nevíme, jak software dobře navrhnout a hlavně jak efektivně řídit vývoj velkých projektů. Tak, jako se v průběhu času měnili požadavky na software, museli přicházet i nové metodiky a paradigmata, které si kladli za cíl, jak stále složitější programy vytvářet, řídit a především udržovat. Jedním takovýmto paradigmatem je i objektově orientované programování, které patří do základní výbavy každého dobrého programátora.

> Počátky objektově orientovaného programování sahají do sedmdesátých let minulého století, kdy Steve Jobs (zakladatel společnosti Apple) poprvé navštívil firmu Xerox. Tamní představitelé společnosti mu ukázali tři revoluční produkty. Byli jimi počítačová myš, první grafické uživatelské rozhraní a objektově orientované programování. Všechny tyto tři vynálezy později převratným způsobem změnili celé odvětví a do jisté míry je používáme i dnes.

Objektově orientované programování (OOP – object oriented programming) je paradigma (rozuměj způsob myšlení nebo vzor), které si klade za cíl do programování přenášet objekty a vztahy z reálného světa. Pokud bychom programovali například informační systém pro školní knihovnu, měli bychom objekty jako je **kniha**, **student**, **třída** nebo **škola**. Objekt kniha by obsahoval vlastnosti jako je **název**, **rok vydání** nebo **žánr** a objekt student kromě svých základních identifikačních vlastností může zahrnovat i metodu pro zapůjčení konkrétní knihy. Vše lépe ilustruje diagram níže.



Jednoduchý diagram zobrazující možné třídy (business entity) pro informační systém školní knihovny.

Obsah jednotlivých objektů (přesněji řečeno jejich rozhraní) je čistě na nás. Objekty a vztahy mezi nimi jsou pro člověka lépe představitelné a dovolují tak poměrně snadno řešit danou problémovou doménu. **V odborném jazyce bychom řekli, že OOP vnáší do naší práce více abstrakce.**

> Kromě pojmu **objekt** se můžeme v OOP setkat i s výrazem **instance**. **Objekt a instance jsou synonyma** a znamenají tudíž totéž.

Třída a struktura jako základ objektu

Instance neboli objekt vzniká vždy na základě nějaké třídy nebo struktury. V minulé sekci jsme si základní myšlenku OOP ilustrovali na příkladu s knihovnou. Pojďme se tedy nyní podívat, jak by vypadala výroba obyčejného objektu knihy:

Třída Kniha

```
class Kniha
{


}
```

Jak již bylo zmíněno výše, základem každého objektu je vždy **třída** (`class`) nebo struktura. Strukturu nyní ponecháme stranou a zaměříme se pouze na třídu. V ukázce jsme definovali novou třídu se jménem `Kniha` a prozatím jsme její tělo ponechali prázdné. I takováto prázdná třída by

bohatě postačila jako startovní bod pro vznik nového objektu:

Výroba objektu (neboli instance) ze třídy

```
Kniha mojeKniha = new Kniha();
```

Tímto příkazem jsme vytvořili nový objekt třídy `Kniha` se jménem `mojeKniha`. Odborně bychom řekli, že jsme **vytvořili instanci třídy Kniha**. Pojďme se nyní blíže podívat na celý zápis. Začali jsme definicí proměnné se jménem `mojeKniha`. Samotná třída `Kniha` je zde datovým typem podobně, jako například typy `int` nebo `string`. Kdykoliv tedy vytvoříme novou třídu, tvoříme tím i náš **vlastní nový datový typ**. O proměnné `mojeKniha` bychom tedy mohli říct, že je datového typu `Kniha`. Za operátorem přiřazení (`=`) poté následuje klíčové slovo `new` a poté volání tzv. konstruktoru. **Konstruktor** je speciální typ metody, který nám po svém zavolání vrací onu instanci neboli objekt. Pokud se ale blíže podíváme na implementaci naší třídy `Kniha` tak zjistíme, že zde žádný konstruktor definovaný nemáme. Jak je tedy možné, že konstruktor voláme? Kompilátor si totiž daný konstruktor vytvoří automaticky sám, pokud ve třídě žádný nenajde.

> Konstruktory budou látkou dalšího ročníku a toto je v tuto chvíli vše, co o nich v potřebujeme vědět.

Pro úplnost si ještě ukážeme celý kód viz ukázka níže:

Výroba objektu "mojeKniha"

```
using System;
□□□□□
public class Program
{
□public static void Main()
□{
□□Kniha mojeKniha = new Kniha();
□}
}

class Kniha
{

}
```

Samotná třída `Kniha` může být vytvořena na úrovni namespacu (viz ukázka) nebo může být oddělena do samostatného `.cs` souboru. V praxi se nejčastěji setkáme s druhou variantou, protože programy mohou obsahovat klidně i stovky tříd. Jejich členěním do samostatných souborů (často pojmenovaných stejně jako třída samotná - např. `Kniha.cs`) dosáhneme lepší organizovanosti našeho kódu. Na pořadí nebo způsobu, v jakém definujeme třídy nezáleží, pokud dodržíme

pravidlo, že všechny vytváříme ve stejném namespacu.

Když už nyní máte představu, co to objekt je a jak ho vytvoříme, asi se sami sebe ptáte, k čemu je toto všechno vlastně dobré? Užitečnost objektů si tedy nyní budeme demonstrovat na malé ukázce:

Třída s veřejnými proměnnými

```
class Kniha
{
    public string nazev;
    public int rokVydani;
}
```

Naše instance kniha1 a kniha2 nyní umožňují, aby do nich byl uložen název knihy a její rok vydání. Data v těchto instancích jsou na sobě nezávislá a každá instance může mít svá unikátní data. Musíme si dát pouze pozor na to, abychom k samotným datům objektů přistupovali právě přes jejich instance (proměnné kniha1 a kniha2) a nikoliv přes název jejich třídy (Kniha)! Až se naučíme pracovat s konstruktory, můžeme tento zápis ještě více zkrátit. **Samotná užitečnost objektů tedy tkví v enkapsulaci neboli zapouzdření dat.**

> Enkapsulace je jeden ze čtyř základních pilířů OOP.

Kdybychom OOP neznali, museli bychom náš kód zamořit velkou spoustou nepřehledných proměnných, jako je to vidět na následující ukázce:

Výroba knih bez znalosti OOP

```
kniha1Nazev = “Robinsoe Crusoe”;
kniha1rokVydani = 1797;
kniha2nazev = “Válka s mloky”;
kniha2rokVydani = 1940;
// a tak dále...
```

Naše třída zatím obsahuje pouze název a rok vydání, takže použití několika proměnných jako na ukázce výše se nyní nezdá jako špatný nápad. V reálné aplikaci by ale naše modelová třída `Kniha` mohla obsahovat desítky dalších členů (např. jazyk, autora ilustrací, počet stran atd.) a nemuseli bychom být omezeni pouze na knihy dvě ale i tisíce. S takovými čísli bez použití OOP by byla složitost naší aplikace neúnosná.

Závěr

Objektově orientované programování (**OOP**) patří mezi základní znalosti každého programátora. **Objekty** tvoříme vždy na základě obecných předpisů (tzv. **tříd** - `class`). Z jedné třídy může

vzniknout mnoho objektů, které jsou na sobě nezávislé a mají své vlastní data a vnitřní stav (enkapsulace).

Revision #1
Created 2025-05-21 12:37:54 UTC by Magdalena Dobešová
Updated 2025-05-21 13:09:20 UTC by Magdalena Dobešová