Programování

Opakování OOP

Objektové modelování

- Abstrakce reálných objektů – zjednodušení pro naši potřebu
- Nepodstatné informace jsou zahozeny
- Principy vytváření modelů:
  - **Abstrakce** – umožnit identifikovat podobnosti potřebné v modelu a odstínit se od nedůležitých rozdílů
  - **Formalizace** – umožnit efektivní komunikaci během vývoje nejen v týmu, ale i se zákazníkem (přesné vyjadřování)
  - **Jednoznačnost** – díky formalizaci můžeme přesně popsat jednotlivé části vytvářeného systému (data, funkce, vlastnosti, …)
  - **Zamezení redundancím** – modelování by mělo zamezit výskytu dvojího, vzájemně se vyvracejícímu tvrzení, případně duplicit

Proč vytváříme modely?

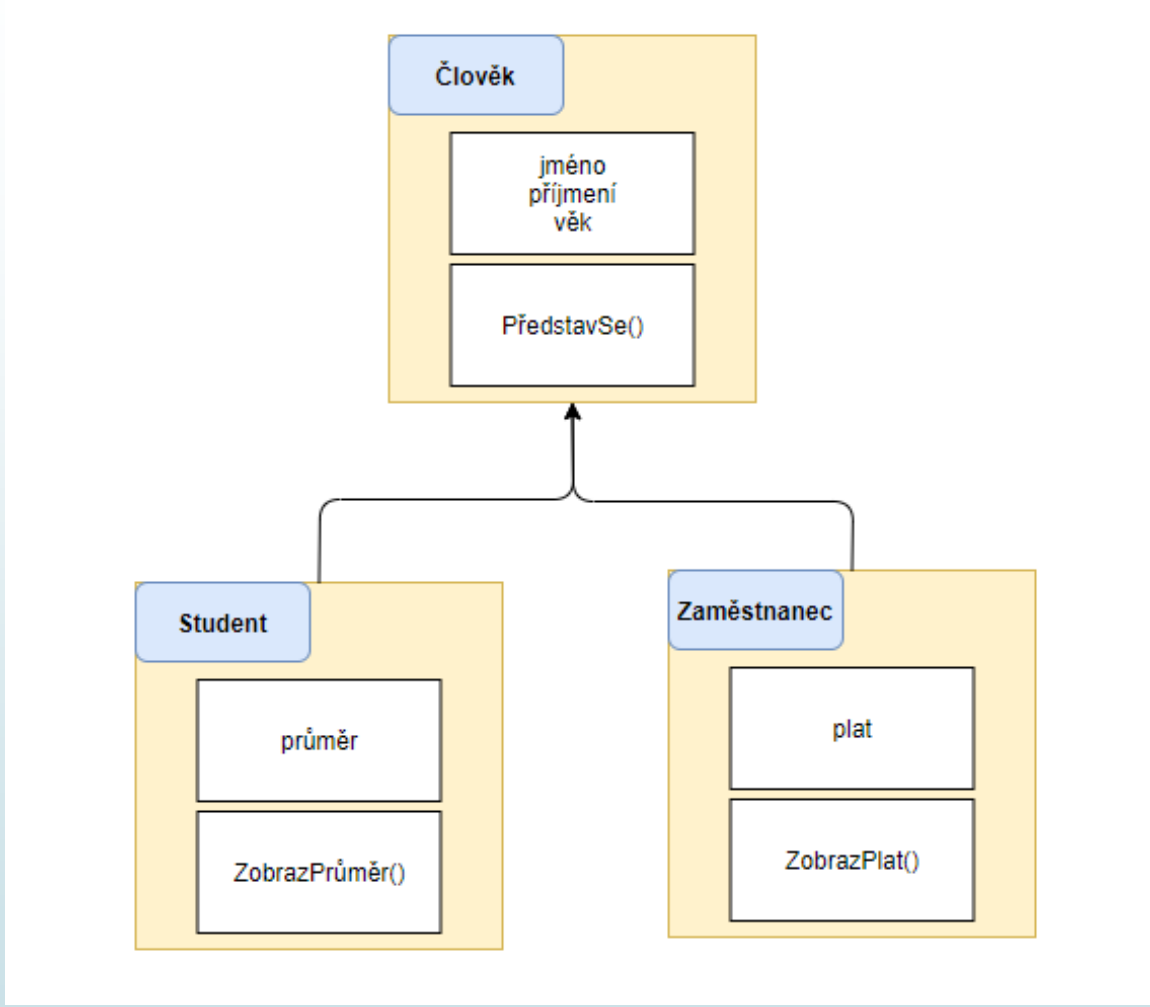
- Vytvářený model je nám šablonou pro nově vytvářené objekty
- Se zjednodušeným modelem umíme lépe pracovat
- Model má svůj vlastní stav, své chování a množinu vlastností
  - V OOP nazýváme vlastnosti **atributy**
  - Chování (co umí) jsou samostatné **metody**
- Objektem chápeme již konkretizovaný model
- V OOP nazýváme tuto šablonu jako **třídu** (class)
- **Vytvářený objekt je tedy konkrétní instancí třídy**

Koncepce OOP

- Objektově orientované programování (OOP) využívá tří základních principů:
  - **Dědičnost**
  - **Zapouzdření**
  - **Polymorfismus**
- Tyto principy z této koncepce dělají silný nástroj pro vytváření kvalitních aplikací
- Jazyky, které nebyly původně objektové také začínají přejímat tyto principy

Dědičnost

- Princip v OOP umožňující vytvářet hierarchii mezi třídami
- Umožňuje podřazeným třídám zdědit atributy a metody nadřazené třídy
- Podobné třídy lze tímto způsobem sjednotit a urychlit jejich implementaci
- Tento princip lze aplikovat s přesahem na návrh databáze
- Nejvyšší třída v C# je třída Object



Zapouzdření

- Princip, který nám umožňuje na vytvořený objekt pohlížet jako na uzavřenou skříňku s vstupy a výstupy
- Možnost skrýt přístup k některým atributům a metodám
- Důvod to má z hlediska bezpečnosti (citlivé údaje) a přístupu (nechceme, aby uživatel měl možnost vidět interní proces)
- Pro zapouzdření využíváme modifikátorů viditelnosti:
  - **public**,
  - **private**,
  - **Package**

Polymorfismus

- Možnost objektu volat metodu s různými vstupními parametry
  - Přetížení funkcí
- Volaní vlastní implementace funkce třídy, která danou metodu zdědila
  - Nadřazená třída nám může vytvářet rozhraní pro třídy odvozené
  - Která implementace se zvolí se rozhoduje až za chodu programu
- Na třídy, které jsou odvozené ze stejné super třídy můžeme nahlížet jako na stejný typ objektu
  - Do konkrétního objektu lze přiřadit objekt, který z původního objektu odvozen
  - Do třídy vozidlo lze vložit objekt motorka, auto, autobus, kolo, …

Vytváření objektů a tříd v C#

- Pro vytvoření objektu je třeba mít nejprve vytvořenou třídu
  - Pro větší projekty ideálně každá třída do samostatného souboru
  - I z třídy, která nemá žádné atributy ani metody lze vytvořit objekt
- Objekt vytváříme za pomocí konstruktoru
  - Konstruktor je metoda se stejným názvem jako sama třída
- Třída může mít libovolný počet konstruktorů, které se od sebe liší počtem vstupních parametrů
- Vytvoření nového objektu je podobné jako deklarace proměnné s rozdílem, že používáme klíčové slovo **new**
- Třídu nemusíme speciálně vytvářet – pohlížíme na ni jako na šablonu nového objektu

Vytvoření třídy

- Takto definovanou třídu můžeme použít pro vytvoření konkrétního objektu
- ***Obdelnik mujObdelnik = new Obdelnik(7, 13);***
- Do proměnné mujObdelnik, která je typu Obdelnik jsme za pomocí konstruktoru vložili nový objekt, kterému budou nastaveny hodnoty A a B podle volaného konstruktoru

```
Počet odkazů: 6
class Obdelnik
{
    Počet odkazů: 4
    public int A { get; set; }
    Počet odkazů: 4
    public int B { get; set; }
    Počet odkazů: 1
    public Obdelnik() { A = 5; B = 8; }
    Počet odkazů: 1
    public Obdelnik(int x) { A = x; B = x; }
    Počet odkazů: 1
    public Obdelnik(int x, int y) { A = x; B = y; }
    Počet odkazů: 1
    public override string ToString()
    {
        return "[" + A + "," + B + "]";
    }
}
```

Dědičnost, polymorfismus, zapouzdření v C#

- Tím, že je C# jazyk postavený na OOP lze využívat tyto principy i v našich projektech
- Dědičnost nám umožňuje vytvářet odvozené třídy, které mohou dědit metody i atributy
- Zapouzdřením docílíme uzavření vytvářené třídy před vnějším kódem
- Polymorfismus nám mimo jiné umožňuje připravit půdu pro naše pokračovatele a jejich vlastní implementaci

Polymorfismus, dědičnost, zapouzdření

- Důležitá klíčová slova, pojící se s použitím těchto vlastností
- Base – odkazuje na nadřazenou třídu
- Override – přepis již existující metody z původní nadřazené
- Virtual – označení funkce u které umožníme přepsání funkcionality
- Při zápisu dědičnosti tříd při popisu třídy užíváme dvojtečky
- **Odvozená třída : Původní nadřazená třída**

```
Počet odkazů: 1
class Soubor
{

    private string nazev;
    private int velikost;
    Počet odkazů: 0
    public Soubor(string nazev)
    {
        this.nazev = nazev;
        this.velikost = nazev.Length * 10;
    }
    Počet odkazů: 0
    public string Nazev { get { return nazev; } set { zmenaNazvu(value); } }
    Počet odkazů: 1
    private void zmenaNazvu(string noveJmeno)
    {
        this.nazev = noveJmeno;
        this.velikost = nazev.Length * 10;
    }

}
```

```
Počet odkazů: 3
class Zvire
{
    Počet odkazů: 5
    public virtual void Promluv() { Console.WriteLine("Mluvím!"); }
}
Počet odkazů: 1
class Pes: Zvire
{
    Počet odkazů: 5
    public override void Promluv() { Console.WriteLine("Haf haf!"); }
}
Počet odkazů: 1
class Kocka : Zvire
{
    Počet odkazů: 5
    public override void Promluv() { Console.WriteLine("Mňau mňau!"); }
}
```