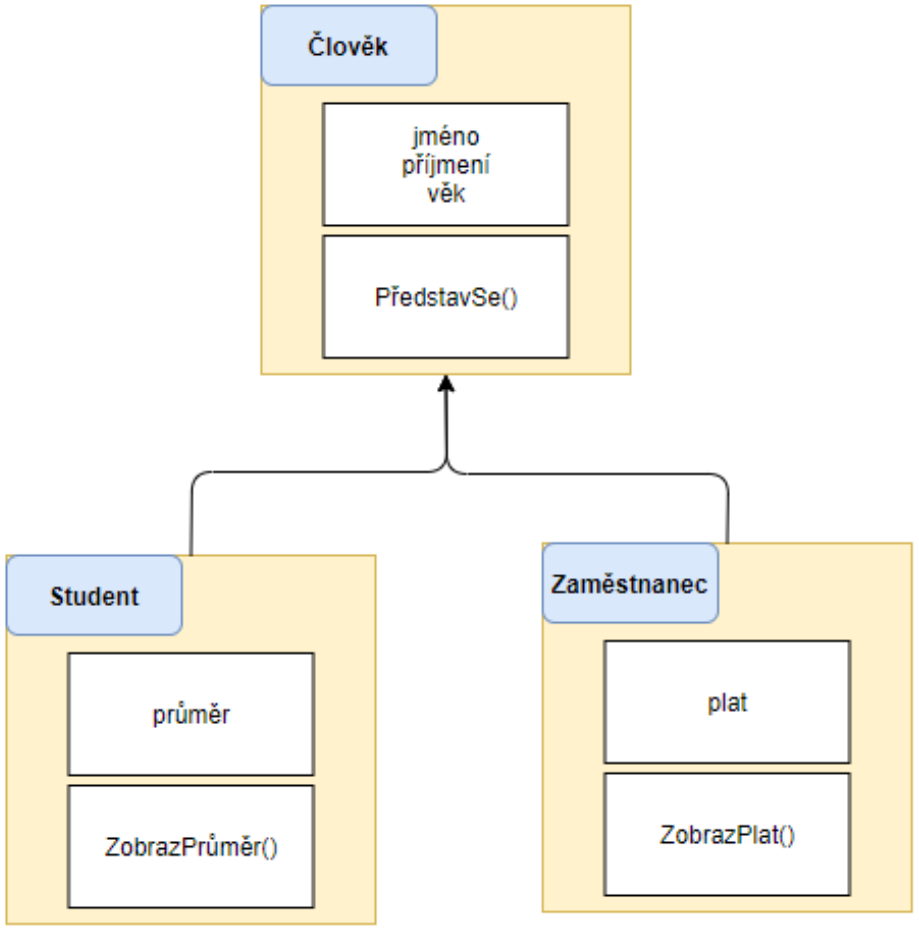
Programování

Dědičnost, zapouzdření a polymorfismus v OOP

Základní pilíře OOP - opakování

- **Dědičnost**
  - Možnost vytváření hierarchií mezi jednotlivými třídami
  - Podtřídy mohou od nadřazené třídy dědit atributy a metody
- **Zapouzdření**
  - Citlivé data jsou pro ostatní část aplikace nebo uživatele schována
  - Třída je reprezentována jako černá skříňka se vstupy a výstupy
  - Využití viditelnosti
- **Polymorfismus**
  - Třídy odvozené ze stejného typu mohou mít vlastní a odlišnou implementaci funkcí
  - Na odvozené třídy lze pohlížet stejně jako na třídu, ze které byly odvozeny

Dědičnost v C#




```
Počet odkazů: 2
class Clovek
{
    private string jmeno;
    private string prijmeni;
    private int vek;
    Počet odkazů: 0
    public void PredstaveniSe() { }
}

Počet odkazů: 0
class Student : Clovek
{
    private double prumer;
    Počet odkazů: 0
    public void ZobrazPrumer() { }
}

Počet odkazů: 0
class Zamestnanec : Clovek
{
    private int plat;
    Počet odkazů: 0
    public void ZobrazPlat() { }
}
```

Dědičnost konstruktoru v C#

- Oproti některým jiným jazykům C# nedědí konstruktory
- Pro naznačení dědičnosti je využito klíčového slova ***base***
- Každá třída tak má vlastní konstruktor
- Ostatní metody a atributy jsou ovšem děděny

```
Počet odkazů: 2
class NadrazenaTrida
{
    private int atributNadrazena;
    Počet odkazů: 1
    public NadrazenaTrida(int attr)
    {
        this.atributNadrazena = attr;
    }
}
Počet odkazů: 1
class PodrazenaTrida : NadrazenaTrida
{
    private int atributPodrazena;
    Počet odkazů: 0
    public PodrazenaTrida(int attrA, int attrB) : base(attrA)
    {
        this.atributPodrazena = attrB;
    }
}
```

Zapouzdření v C#

- Atributy jsou skryté
- Měnit a zobrazit můžeme pouze skrze funkce
- Vnitřní metody funkce jsou rovněž skryté
- Public – přístupné metody
  - Atributy necháváme vždy skryté
- Private – metody a atributy přístupné pouze v konkrétní třídě

```
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;
    }

}
```

Polymorfismus v C#

- U metody třídy, kde očekáváme jinou implementaci uvedeme klíčové slovo ***virtual***
  - Uvádíme u nadřazení třídy
- Metoda, která původní metodu reimplementuje používá označení ***override***
  - Uvádíme u tříd podřazených

```
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!"); }
}
```

Polymorfismus a dědičnost dohromady

- Stejně jako u konstruktoru se můžeme odkazovat na nadřazené metody
- Pro jejich volání využijeme klíčového slova ***base***
- Opět využíváme označení ***override*** a ***virtual*** pro přepsání původní metody a označení metody nadřazené

```
Počet odkazů: 3
class Vozidlo
{
    private double rychlost;
    Počet odkazů: 2
    public virtual bool Zavod(Vozidlo vuz)
    {
        return this.rychlost > vuz.rychlost;
    }
}

Počet odkazů: 0
class Formule: Vozidlo
{
    Počet odkazů: 2
    public override bool Zavod(Vozidlo vuz)
    {
        return  base.Zavod(vuz);
    }
}
```

Revision time!