

## Abstrakce v OOP

- Při modelování a následné implementace tříd se zaměřujeme pouze na důležité věci
- Vytváříme abstrakci skutečného předmětu pro jednodušší práci s daným objektem
- Jazyky podporující OOP využívají abstrakce v plné míře
- Umožňují vytvářet abstraktní metody a třídy
- Abstraktní atribut, metodu nebo třídu uvozujeme klíčovým slovem abstract
- Modifikátor nám říká, že implementace chybí nebo je neúplná

#### Abstraktní třída

- Abstraktní třída je taková třída, která nám slouží pro odvozování dalších tříd
- Z takové třídy nelze vytvořit objekt
- Abstraktní třída může obsahovat běžné i abstraktní metody
- Třídy, které jsou odvozené z abstraktní třídy musí obsahovat implementaci všech abstraktních tříd, které dané třídě předchází

```
abstract class Zivocich
    Počet odkazů: 0
    public abstract string Promluv();
    Počet odkazů: 0
    public void Spi()
        Console.WriteLine("Zzzz...");
```

## Abstraktní metoda

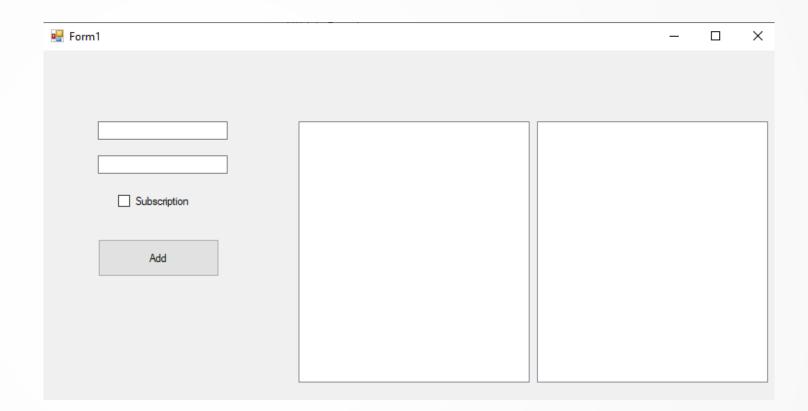
- Modifikátor abstract u metody nám říká, že tato metoda neobsahuje implementaci
- Existuje pouze definice hlavička funkce a implementace se nachází jinde
- Abstraktní metoda je explicitně vedená jako virtuální metoda i bez uvedení tohoto modifikátoru
  - Využití polymorfismu na základě příslušnosti ke třídě se daná funkce může chovat jinak
- Při implementaci abstraktní funkce musíme uvést modifikátor override
- Každá odvozená třída, která vychází z třídy obsahující abstraktní metodu, musí sama obsahovat implementaci této metody

## Proč využívat abstrakci?

- Při návrhu aplikace není třeba řešit samotnou implementaci
- Abstraktní třídy společně s rozumným komentářem nám dokáže přiblížit funkcionalitu jednotlivých metod a využití tříd
- Na plno využíváme vlastností OOP, zejména polymorfismu
- Vytvořením abstraktních tříd hledáme podobnosti mezi jednotlivými objekty a zamezíme tak duplicitám kódu
- Díky abstraktním třídám můžeme schovat určité detaily a zobrazit pouze podstatné informace
- Víme, že daná třída má určité metody a atributy, ale jak se chovají je skryto

## Rozhraní

- Vytvářením formulářových aplikací již vytváříme uživatelské rozhraní
- Z popisků u jednotlivých komponent ví, co daná aplikace obsahuje a co by měla umět
- Samotná implementace je však pro uživatele skrytá



## Interface

- Interface (rozhraní) v programování funguje na podobném principu jako rozhraní zobrazované uživateli
- Popisuje, co daná třída nebo struktura umí a obsahuje
- Interface může obsahovat metody, události, ...
- Interface ovšem neobsahuje implementaci jednotlivých částí, které popisuje
- Obdobně jako u abstraktních tříd, rozhraní může být odvozeno od jiného rozhraní
  - Dědí tedy jednotlivé atributy a metody z nadřazeného rozhraní
- Za pomocí interface dokážeme rovněž dosáhnout jisté míry abstrakce
- Při vytváření rozhraní využíváme klíčového slova interface

# Implementace rozhraní

- Každá třída případně struktura může implementovat jedno či více rozhraní
- Objekt, který je vytvořený z třídy, která implementuje rozhraní může být implicitně (automaticky) převedená na typ daného rozhraní

```
Počet odkazů: 1
interface Zivocich
    Počet odkazů: 1
     string Promluv();
    Počet odkazů: 1
    void Spi();
Počet odkazů: 1
interface Savec
     Počet odkazů: 1
     string TypSrsti();
```

```
class Pes : Zivocich, Savec
    Počet odkazů: 1
    public string TypSrsti()
        return "hladká";
    Počet odkazů: 1
    public string Promluv()
        return "Haf! Haf!";
    Počet odkazů: 1
    public void Spi()
        Console.WriteLine("Zzzz...");
```