

# Objekty

Svět se skládá z objektů!

konkrétní x abstraktní  
hmatatelné x nehmatatelné  
(letadlo) x (chyba v programu)

Objekty mohou obsahovat jiné objekty  
(tělo obsahuje buňky, letadlo součásti).

Objekty URČITÝM ZPŮSOBEM PODOBNÉ můžeme  
považovat za instance jedné TŘÍDY (pes).

# Objekty

Další pokus oddělit

CO x JAK

VENKU x UVNITŘ

INTERFACE x IMPLEMENTACE

Strukturované programování

blok, funkce

Modulární programování

modul, unit

Objektové programování

objekt

# Objekty v programu

Způsob jak izolovat část kódu  
(příkaz-blok-procedura-modul-objekt).

Způsob jak uvažovat o problému

Objekt sdružuje DATA (datové složky, vlastnosti)  
i KÓD (funkce+procedury=METODY)  
= ČLENY (members)

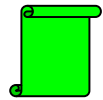
**OBJEKT** = exemplář, instance třídy.

```
class Komplex
{
    public double Re, Im;
    public double AbsolutniHodnota()
    {
        return
            Math.Sqrt(Re * Re + Im * Im);
    }
}

...
Komplex k = new Komplex();
k.Re = 1.00;
k.Im = 1.00;
double y = k.AbsolutniHodnota();
```

# Konstruktor

- metoda volaná při vytváření instance
- slouží k inicializaci objektu
- má důležitý vedlejší efekt (později!).



```
Pes pes = new Pes("alík", 5, 20, 2);  
pes.Stekni();
```

# haf! jmenuju se alík a už jsem pokousal 2 lidi!

# Více konstruktorů

Třída může mít více konstruktorů, musí se lišit parametry (počet, typ).

```
public Pes(string jmeno, int vaha, int vyska,
           long KolikLidiPokousal)
{
    this.jmeno = jmeno;
    this.vaha = vaha;
    this.vyska = vyska;
    this.KolikLidiPokousal = KolikLidiPokousal;
}
public Pes()
{
    this.jmeno = "noname";
}
```

# Dědičnost

ODVOZENÝ datový typ (POTOMEK)  
...DĚDÍ od svého RODIČE (PŘEDKA)

- všechny datové složky
- všechny metody

Může PŘIDÁVAT datové složky a metody.  
Může PŘEPISOVAT metody.



```
class VelkyPes: Pes
{
    public VelkyPes(string jmeno, int vaha,
int vyska, long KolikLidiPokousal)
    {
        this.jmeno = jmeno;
        this.vaha = vaha;
        this.vyska = vyska;
        this.KolikLidiPokousal = KolikLidiPokousal;
    }
    public new void Stekni()
    {
        Console.WriteLine( "HAF! HAF! {1} lidi!",
jmeno, KolikLidiPokousal );
    }
}
```

```
VelkyPes pes2 = new VelkyPes("Rokko", 105, 50, 27);
pes2.Stekni();
```

HAF! HAF! 27 lidi!

# Předefinování metody

- klíčové slovo **new**

```
public new void Stekni()  
{  
    Console.WriteLine(  
        "HAF! HAF! {1} lidi!",  
        jmeno, KolikLidiPokousal );  
}
```

# Volání metody předka

## Pomocí klíčového slova `base`

```
class VelkyPes: Pes
{
    ...
    public new void Stekni()
    {
        base.Stekni();
        Console.WriteLine(" (HAF! HAF!) ");
    }
}
```

```
haf! jmenuju se Hugo a uz jsem pokousal 57 lidi!
(HAF! HAF!)
```

# Problém s předefinovanou metodou



```
VelkyPes pes2 = new VelkyPes("Rokko", 105, 50, 27);  
pes2.Stekni2x();
```

```
haf! jmenuju se Rokko a uz jsem pokousal 27 lidi!  
haf! jmenuju se Rokko a uz jsem pokousal 27 lidi!
```

# Vysvětlení

Metoda `Stekni2x()` volá (dvakrát) metodu `Stekni()`...  
...z třídy `Pes` (jiná třída v té době ani neexistovala).

Třída `VelkyPes` předefinuje metodu `Stekni()`,  
ale metodu `Stekni2x()` dědí od (malého) `Psa`.

*Je tedy vše ztraceno? (odpověď na příštím slajdu)*

# Obyčejné a virtuální metody

## Metody:

- obyč (o jejich volání je rozhodnuto při překladu, "early-binding")
- virtuální (o jejich volání se rozhoduje až v okamžiku volání, "late-binding")



## Syntaxe:

- neřekneme-li nic, je metoda ne-virtuální (obyčejná, BP: „statická“)
- deklarace virtuální metody: `virtual`, `override`

# Obyčejné a virtuální metody 2.

## POZOR:

public virtual void Stekni()

zakládá nový kořen, **NEPŘEPISUJE** starou metodu

(tj. pokud se ta metoda volá nepřímo,

tak se volá ta původní)

# Virtuální metody - jak to funguje

Metoda předka dokáže zavolat metodu potomka,  
jak je to možné?

## VMT: Tabulka virtuálních metod

- 1 pro každou třídu (tj. typ), vytváří překladač
- obsahuje adresy virtuálních metod
- při volání se volá metoda uložená ve VMT
- objekt obsahuje odkaz na VMT
  - kdo ho tam dosadí?  
= **KONSTRUKTOR!**



# Virtuální metody - jak to funguje 2

Co když nezavolám konstruktor?

= V C# nemám možnost jak vytvořit objekt,  
aniž bych zavolal konstruktor

(v Pascalu, C++ etc. to jde  
a pak to končí !@#\$%^&\*() )

# Polymorfismus

- objekty se vytvářejí dynamicky
- proměnné objektových typů jsou jen ukazatelé
- kompatibilita typů (ukazatelů),

VelkyPes je také Pes...

- ...ale vidíme jen to, co má Pes
- opačným směrem nelze  
(bez použití nějakých konverzních metod)

=> volá se metoda

příslušná aktuálnímu TYPU (třídě) objektu.

Ta metoda **MUSÍ BÝT** virtuální.

# Polymorfismus - Příklad 1.

```
Pes pes;
```

```
for (int i = 1; i <= 2; i++)
```

```
{
```

```
    if (i==1)
```

```
        pes = new Pes("alík", 5, 20, 0);
```

```
    else
```

```
        pes = new VelkyPes("Hugo", 5, 20, 57);
```

```
    pes.Stekni();
```

```
}
```

```
...
```

```
haf! jmenuju se alík a už jsem pokoušel 0 lidí!
```

```
HAF! HAF! Už jsem pokoušel 57 lidí!
```

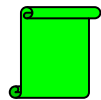
# Abstraktní metoda, abstraktní třída

## Abstraktní třída:

- společný předchůdce jiných tříd
- nebudeme od ní vytvářet instance

## Abstraktní metoda:

- slouží jen k tomu,  
aby mohla být předefinována v potomcích
  - může být předefinována zase abstraktní metodou
  - potomci nemohou volat TUTO metodu předka
- => ta třída je potom také abstraktní



# Statické členy a třídy

## Statické členy (members) (metody, data...)

- jsou alokovány ve třídě a ne instanci.
- jsou přístupné pomocí jména třídy  
(nejsou přístupné pomocí jména instance)

## Statická třída

- obsahuje pouze statické členy
- nelze z ní vytvářet instance pomocí new
- nelze z ní dědit



I třída, která není statická, může mít statické členy.

# Atributy přístupnosti/viditelnosti

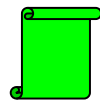
- **public**: přístupné všem
- **protected**: přístupné jen z této třídy/struktury a z potomků
- **internal**: přístupné jen z aktuálního assembly...
- **protected internal**: přístupné jen z aktuálního assembly... nebo z potomků
- **private**: přístupné jen z této třídy/struktury

Když není uvedeno, platí výchozí hodnota,  
pro **class** je to **private**.

# Zapouzdření

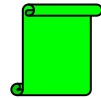
- nenechte nikoho sahat na svá data !  
(veřejné obvykle jenom metody  
a VLASTNOSTI (za chvíli) )
- interface = smlouva,  
to, co používají ostatní části programu

Výhoda zapouzdření:  
objekty jsou vždy v konsistentním stavu



# Properties - vlastnosti

- pro uživatele vypadají jako datová složka
- sestávají z 1 nebo 2 bloků kódu **get-set**
- když chybí **set**, je property **read-only**
- když chybí **get**, je property **write-only**





# Sealed

## třídy

- už od nich nelze dědit

## metody a ostatní members

`public sealed override void DoWork()`

- už dál nebude virtuální
- lze ji přepisovat via `new`

