# Třídy a objekty

Co je objekt

Proč?

Třída, deklarace, použití

## Co je objekt?

- abstraktní **napodobení reality**
- jako součástky
- umožňuje dobře rozložit program na dílčí podproblémy (**dekomponovat**)
- v objektově orientovaném programování se vytváří **objekty** (černé skříňky), které mají pevně definované **rozhraní**, tzn. vstupy a výstupy a vnitřek je ukryt (jakým způsobem je řešen proces uvnitř černé skříňky není pro okolí důležité)
- výhodou je, že objekty se mohou snadno vyměňovat, nahrazovat lepšími, aniž by bylo nutné zasáhnout do okolního programu
- za předpokladu, že má stejné rozhraní (vstupy a výstupy)

## Proč?

- objektově orientovaný program je názornější, srozumitelnější a lze ho lépe spravovat
- pokud je potřeba něco vylepšit, zpravidla se to týká několika objektů a zbytek lze nechat v původním stavu
- to určitě přispívá k větší efektivitě návrhu a tvorby programu
- v objektově orientovaném programování je tedy cílem navrhnout objekty, které řeší dílčí úkoly a definovat vhodné rozhraní objektů pro jejich vzájemnou komunikaci

## Třída

- ve většině objektově orientovaných programovacích jazycích je nutné před tím, než se vytvoří objekt, deklarovat třídu, třída definuje, jak budou nové objekty vypadat a pracovat
- **třída je šablonou pro tvorbu objektů,** je v ní tedy definováno, co by objekt z dané třídy měl obsahovat a jakou činnost by měl provádět
- objekty jsou **stavové**, to znamená, že mají uložený stav, stav objektu je obvykle **skrytý** (soukromý, privátní) a většinou je vyjádřen pomocí **datových členů**, **datových položek** čili **atributů**
- objekt musí mít přesně definované **rozhraní**, pomocí něhož bude komunikovat s ostatními objekty, takové rozhraní představují **veřejné funkční členy**, především **metody**

## Deklarace třídy – datové a funkční členy

- **třída je datový typ**, pomocí něhož lze vytvářet **objekty**
- **objekt je instancí třídy**
- objekt je základním prvkem objektově orientovaného programu
- dříve než můžeme vytvořit objekt, musíme definovat třídu a v ní:

1. **datové členy (atributy), data** specifická pro danou třídu
   1. **datová položka** je každá proměnná, která je deklarovaná na úrovni třídy. Je tedy specifická pro objekty, které se z této třídy budou vytvářet.
   2. **událost**: objekt, který je instancí dané třídy, může vyvolat určitou událost.
2. **funkční členy**: metody, konstruktory, destruktory

## Deklarace třídy – veřejné datové složky

deklarace třídy Zaměstnanec:

```
class Zamestnanec
{
   // deklarujeme veřejné datové složky
   // typu string s nazvem Jmeno:
   public string Jmeno;
    // typu string s nazvem Prijmeni:
   public string Prijmeni;
   // typu int s nazvem cisloPracoviste:
   public int cisloPracoviste;
 }
```

## Přístup k veřejným datovým složkám

```
static void Main()
  { // instance třídy Zamestnanec:
    Zamestnanec z1 = new Zamestnanec();
    // nastavení datové složky na určitou hodnotu:
    z1.Jmeno = "Petr";
    // výpis:
    Console.WriteLine("Jméno: {0}", z1.Jmeno);
  }
```

- k veřejným datovým složkám lze přistoupit z vnějšku třídy pomocí operátoru tečka
- nevýhoda je, že okolní kód může zadat nepřípustnou hodnotu do takové datové složky
- řešením je deklarovat takové datové složky jako **soukromé** a přistupovat k nim pomocí veřejných funkčních členů, jinak řečeno metod

## Funkční členy - metody

- pro jednoduchý výpočet nepotřebujeme objekt, který by obsahoval i nějaké údaje (stav)
- pokud požadujeme metodu, kterou by bylo možné volat bez tvorby objektu, použijeme **statickou metodu**
- statická metoda je metodou třídy a ne objektu
- každý program v jazyce C# obsahuje vždy statickou metodu Main()
- deklarujme třídu např. s názvem Matematika a její statickou metodu ObsahCtverce
- této metodě předáme jeden parametr, kterým je strana čtverce, metoda provede vlastní výpočet a nakonec na místo volání vrátí vypočtenou hodnotu

## Funkční členy - metody

- v mnoha případech má objekt stav a nad tímto stavem pracuje metoda
- v takovém případě **není metoda statická**
- taková metoda se nazývá **metoda instance**, neboť je to metoda objektu (instance třídy) a ne třídy
- k zavolání takové metody je potřeba nejdříve vytvořit objekt

## Funkční členy - metody

```
class Matematika
  {
    public double StranaCtverce;

    public double ObsahCtverce()
    {
      return StranaCtverce * StranaCtverce;
    }
   }
```

## Funkční členy - metody

```
static void Main(){
  Console.Write("Zadejte stranu čtverce: ");
  Matematika mat = new Matematika();
  mat.StranaCtverce = double.Parse(Console.ReadLine());

  Console.WriteLine("Obsah čtverce o straně {0} je {1}.",
  mat.StranaCtverce, mat.ObsahCtverce());
  }
```

- nejdříve je potřeba vytvořit instanci třídy Matematika, která bude mít uloženu hodnotu pro délku strany čtverce (zatím veřejnou, což není správné, jak uvidíme později)
- tuto hodnotu nastavíme na požadovanou a pak zavoláme metodu
- metody nemusí vždy vracet hodnotu, takové metody mají návratový typ **void**

## Funkční členy - konstruktory

- objekt by měl mít v každém okamžiku konzistentní stav, všechny jeho datové složky by měly mít správné hodnoty, které nejsou v rozporu s logikou objektu
- samozřejmě správný stav instance by měl být zachován ihned po jejím vytvoření, k takové **inicializaci** slouží právě **konstruktor**
- konstruktor je metoda volaná při vytváření nové instance dané třídy, má stejný název jako třída a nevrací žádnou hodnotu (nemá typ, ani void)
- pokud není deklarován vlastní konstruktor (tzv. **explicitní konstruktor**), překladač vytvoří svůj konstruktor (tzv. **implicitní konstruktor**). (**inicializuje datovou složku číselného typu na hodnotu 0, datovou složku typu řetězec na prázdný řetězec**)

## Implicitní konstruktor

```
public class Pacient
      {
        public string Jmeno;
        public int Vyska;

        public void VypisZpravu()
         {
           Console.WriteLine("Pacient:");
           Console.WriteLine("Jméno: {0}", Jmeno);
           Console.WriteLine("Vyska: {0}", Vyska);
         }
    }
```

## Implicitní konstruktor

```
static void Main()
        {
           Pacient pacient1 = new Pacient();

           pacient1.VypisZpravu();

           Console.ReadLine();
        }
```

Co program vypíše?

Pacient:
Jméno:
Vyska: 0

## Explicitní konstruktor

```
public class Pacient
      {
         public string Jmeno;
         public int Vyska;

         // explicitní konstruktor bez parametrů:
         public Pacient()
         {
            Jmeno = "Nový pacient";
            Vyska = 180;
          }

         public void VypisZpravu ()
         {
            Console.WriteLine("Pacient:");
            Console.WriteLine("Jméno: {0}", Jmeno);
            Console.WriteLine("Vyska: {0}", Vyska);
          }
      }
```

## Přetížení konstruktoru

```
public class Pacient
      {
         public string Jmeno;
         public int Vyska;

         // explicitní konstruktor bez parametrů:
         public Pacient()
         {
           Jmeno = "Nový pacient";
           Vyska = 180;
          }

         // explicitní konstruktor s parametry (přetížený):
         public Pacient(string jmeno, int vyska)
         {
            this.Jmeno = jmeno;
            this.Vyska = vyska;
          }
        // tady bude opět metoda pro výpis
      }
```

## Přetížení konstruktoru

```
static void Main()
      {
      Pacient pacientA = new Pacient();
      Pacient pacientB = new Pacient("Petr", 185);
      pacientA.VypisZpravu();
      pacientB.VypisZpravu();
       Console.ReadLine();
       }
```

Co program vypíše?

Pacient:
Jméno: Nový pacient
Vyska: 180

Pacient:
Jméno: Petr
Vyska: 185

## Funkční členy - destruktory

- prostředí .NET používá **automatickou správu paměti**, jejíž součástí je nástroj pro automatické odstranění těch objektů, na které již není žádná reference, z paměti
- v takovém případě již není implementace destruktorů tak významná, jako u jazyků bez automatické správy paměti, jakým je například C++

## Funkční členy - vlastnosti

## Funkční členy - vlastnosti

- vystavit stav objektu jako veřejný není dobrý postup, jelikož okolní program (kterýkoliv klient objektu) může tento stav změnit i na takový, který odporuje logice objektu
- objekt sám by měl také stanovit, které jeho údaje budou veřejně viditelné a které budou soukromé
- data (stav) v objektu by tedy měla být vždy **privátní** a přístup k těmto datům by měl vést přes **veřejné funkční členy**
- obecně v objektově orientovaném programu komunikují objekty pomocí zpráv

## Komunikace

zprávy dělíme na:

- **Informační (informative message)** - informují objekt o tom, že se někde v systému změnily údaje o něm a tedy, že by měl aktualizovat svůj stav. Změna v systému mohla být uskutečněna například interakcí s uživatelem v dialogovém okně.
- **Dotazovací (interrogative message)** - systém se dotáže objektu na jeho stav, jelikož tuto informaci potřebuje znát, například pro zobrazení v dialogovém okně.
- **Přikazovací (akční, imperative message)** - systém požaduje od objektu nějakou akci, například výpočet, analýzu apod.

## Dotazovací zpráva

dotazovací zpráva implementuje pomocí metody s prefixem Get

```
public class Pacient
    {
       // Stav je soukromý:
       private string jmeno;      private int vyska;
       // konstruktory
       public Pacient()
       {  this.jmeno = "Nový pacient„;  this.vyska = 180;  }
       public Pacient(string jmeno, int vyska)
       {   this.jmeno = jmeno;   this.vyska = vyska; }

       // Implementace dotazovací zprávy
       public string GetJmeno()
       {         return this.jmeno;      }
       public int GetVyska()
       {         return this.vyska;      }

       public void VypisZpravu()
       {  Console.WriteLine("Pacient:");
          Console.WriteLine("Jméno: {0}", this.jmeno);
          Console.WriteLine("Vyska: {0}", this.vyska); }
    }
```

## Dotazovací zpráva

```
static void Main()
  {
   Pacient pacient1 = new Pacient("Karel Novák", 190);

   Console.WriteLine("{0}, {1} cm",
  pacient1.GetJmeno(),  pacient1.GetVyska());

   Console.ReadLine();
   }
```

## Informační zpráva

- Informační zprávy by se bez vlastností implementovaly pomocí metod prefixovaných **Set**.
- (Jméno pacienta musí být zadané. Pokud se bude zadávat prázdný řetězec, uloží se "Nezadané jméno". Minimální výška je 30 cm, maximální 300 cm)

```
public void SetJmeno(string jmeno)
     {  if (jmeno == "")
                this.jmeno = "Nezadané jméno";
        else
                 this.jmeno = jmeno;
    }

 public void SetVyska(int vyska)
     {   if (vyska > 300)
                this.vyska = 300;
        else if (vyska < 30)
                this.vyska = 30;
        else
                this.vyska = vyska;
     }
```

## Informační zpráva

```
static void Main()
{
  Pacient pacient1 = new Pacient("Karel Novák", 190);
  Console.WriteLine("{0}, {1} cm",
   pacient1.GetJmeno(), pacient1.GetVyska());

   pacient1.SetJmeno("Petr Novák");
   pacient1.SetVyska(195);

  Console.WriteLine("{0}, {1} cm",
   pacient1.GetJmeno(), pacient1.GetVyska());

   pacient1.SetJmeno("");
   pacient1.SetVyska(5000);

   Console.WriteLine("{0}, {1} cm",
   pacient1.GetJmeno(),pacient1.GetVyska());
}
```

## Vlastnosti

- ve třídě Pacient máme soukromý stav, máme řízený přístup k tomuto stavu,
- použití funkcí Set a Get však není tak pohodlné jako dříve, kdy jsme přímo přistupovali k veřejným datovým složkám.
- k tomuto pohodlí právě slouží **vlastnosti**
- vlastnosti se navenek chovají jako veřejné datové složky, ve skutečnosti se však jedná o funkční členy s logikou Get / Set

## Vlastnosti

vlastnost Jmeno

```
public string Jmeno
        {
           get
           {
              return this.jmeno;
           }
           set
           {
              if (value == "")
                 this.jmeno = "Nezadané jméno";
              else
                 this.jmeno = value;
           }
        }
```

- pokud použijeme pouze sekci **set** bude možné pouze hodnotu modifikovat avšak ne číst, v případě užití jen sekce **get** dostáváme vlastnost jen pro čtení

## Konec