Poznámky

k virtuálním metodám – aneb co když zkoušíme překladač zlobit...

- Pokud neuvedeme `override`, vznikne v synu nová (nevirtuální) metoda (a pochopitelně se nezavolá, jak bychom chtěli).
- Pokud neuvedeme `virtual` (a `override ano`), je z toho error (nelze overridovat nevirtuální metodu).
- Zapomeneme-li oboje, je z toho warning (zakrýváme metodu a není jasné, jestli je to to, co chceme).
- V předchozím bodě se warningu zbavíme modifikátorem `new`.

Čistě virtuální funkce a abstraktní třídy

- Co když má rodičovská třída sloužit jen jako vzor (a nechceme dělat instance – a některé metody nechceme ani definovat)?
- Uděláme abstraktního rodiče (modifikátor `abstract`).
- Příklad: `abstract class zvire...`
  a `public abstract void VydejZvuk();`
- Abstraktní (čistě virtuální) metoda je automaticky virtuální.
- Odlišnost od C++: C++ nemá modifikátor `abstract` a do čistě virtuální funkce se přiřadí `NULL`.

Příklad

rodič

```
abstract class zvire {
      string jmeno;
      public abstract void VydejZvuk();
      public zvire():this("Nemam!"){}
      public void zvire(string jmeno)
      { this.jmeno=jmeno;}
      public void KdoTam()
      { Console.WriteLine(jmeno);}
}
```

Příklad

syny

```
class tygr:zvire
{    public tygr(string jmeno):base(jmeno){}
     public override void VydejZvuk()
     {    Console.Writeline("Vrrrrrr-rrum!");}
}
class slepice:zvire
{   public slepice():base(){} //slepice jmena nemaji
    public override void VydejZvuk()
    {    Console.WriteLine("Ko - ko - ko!");}
}
```

Modifikátory přístupu

- public – veřejné prvky, přistoupit smí každý
- protected – přistoupit smí sama třída a potomci
- private (implicitní) – jen já (současná třída)
- internal – jen současné assembly (česky sestavení – zatím ignorujme)
- protected internal – současné assembly a potomci.
- Je zvykem nepouštět si nikoho k datům (zapouzdřenost),
- k nastavování a čtení se tvoří veřejné metody.
- Ač někdy hraničí s objektovým fanatismem, ve větších projektech je to užitečné!
- Metody get a set si lze nechat téměř vygenerovat:

Příklad

metod get a set

```
class zvire
{    private int pocetNohou;
     public int PocetNohou //pozor, velikost!
     {    get
          {    return pocetNohou;}
          set
          { if((value % 2)==0) pocetNohou=value;}
     }
}
```

Použití

```
matysek=new Tygr("Matysek");
matysek.PocetNohou=4;
matysek.PocetNohou=3;
System.Console.WriteLine(matysek.PocetNohou);
```

Zapečetěné třídy a metody

- Modifikátor `sealed`,
- znamená, že z třídy nelze dědit,
- metodu nelze overrideovat (lze jen udělat new).
- Příklad: `sealed class bezdeti{...}`
- Příklad: `public sealed override void cilova_rovinka(){...}`

Struktury

- V jazyce C nebyly objekty, ale byly struktury,
- ovládaly se podobně jako recordy v Pascalu,
- definovaly se pomocí klíčového slova `struct` (syntakticky podobně jako třídy),
- V C# jsou také (ale místo nich se spíše používají objekty).
- Příklad: `struct kompl{public int re,im;};`
- Objekt je referenční typ, struktura je hodnotová.

Pole

jsou úplně snadná, ale chovají se úplně jinak než v Pascalu

- Typ pole definujeme operátorem hranatých závorek za jménem typu:
- `int [] poleintu;`
- Pole se definují předem neznámé délky a dynamicky se alokují:
- `int[] pole=new int[10];`
- Pole se indexují od nuly (do délka - 1 (!))
- Pole některých typů lze rovnou inicializovat:
- `int[] pole=new int[3]{1,2,3};` anebo
- `int[] pole=new int{1,2,3};`

Pole II

- Pole námi definovaného typu:
- `seznobecny[] polesez=new seznintu [10];`
- Toto pole se neinicializuje (je plné nullů)!
- `polesez[0]=new seznintu(10);`
- Délka pole – atribut Length:
  `delka=polesez.Length;`
- Při sáhnutí mimo rozsah pole vypadne IndexOutOfRangeException,
- Při použití neinicializovaného odkazu NullReferenceException.

Pole III

- Vícerozměrná pole: `int [,] pole=new int[2,3];`
- Takové pole je obdélníkové a:
  `pole.Rank==2, pole.Length==6`
- Nepravidelné pole (pole polí):
  `int [][] pole=new int [3][];`
- Následně: `pole[0]=new int[3];`
  `pole[1]=new int[2];...`
- Konstrukce `foreach`:
-
```
int[] pole={1,2,3,4,5};
foreach (int i in pole)
    Console.WriteLine(i);
```

Řetězce

Mají některé ďábelské vlastnosti

- Proměnné typu `string`,
- jsou to objekty, ale lze je inicializovat implicitně:
- `string retez="kus textu!";`
- Porovnání na rovnost (překvapivě) porovnává obsahy
- (mezi jazyky rodiny C je to ale spíše výjimka).
- Délka (atribut Length): `retez.Length`,
- Lze k němu přistupovat jako k poli,
- ale je read-only! Pro zápis použijte StringBuilder.
- Pole lze rozsekat podle oddělovačů:
  `string[] retez.Split(char[])`
- Příklad: `string retez="1 22 3 14";`
  `string[]cisla=retez.Split({' '});`

Allokace poznámky

- Množství dostupné paměti:
  `System.GC.GetTotalMemory(bool)`
- Explicitní volání Garbage-collectoru: `System.GC.Collect()`

Namespace

- Namespace je jmenný prostor,
- syntakticky vypadá podobně jako třída,
- určuje území platnosti identifikátorů,
- lze vnořovat namespace do namespacu,
- nelze vnořit namespace do třídy.

Namespace II

- U nelokálních identifikátorů je nutno říct, ve kterém namespacu jsou,
- chceme-li používat identifikátory z konkrétního namespace častěji, můžeme použít `using`, například: `using System;`
- Příklad správně: `using System;`
  `Console.WriteLine();`
- Příklad správně: `System.Console.WriteLine();`
- Příklad špatně: `using System.Console;`
  `WriteLine();`
  špatně, protože `Console` je statická třída, ne jmenný prostor.

Namespace III

- Pomocí `using` lze vytvářet aliasy tříd:
- `using <alias>=<třída>;`
- Příklad: `using c = System.Console;`
  `c.WriteLine();`
- Nyní už rozumíme celému obrázku, který Visual Studio vygeneruje,
- až na to, že ne nutně víme, co nám jednotlivé namespacy umožňují,
- zájemci to ovšem mohou různými způsoby zjišťovat.

Vstup a výstup

Textový soubor

- Standardní vstup se řídí ze třídy `Console` v namespacu `System`.
- Textové soubory se ovládají velmi podobně pomocí StreamReaderu a StreamWriteru.
- `StreamReader a StreamWriter` jsou v namespacu `System.IO`, tedy:
- `System.IO.StreamReader r= new System.IO.StreamReader(@"c:\temp\file.txt");`
- Instance třídy `StreamReader` resp. `StreamWriter` má stejný charakter jako v Pascalu proměnná typu `text` assignovaná ke konkrétnímu souboru.
- V Pascalu jsme jméno proměnné typu `text` předávali každé funkci, v C# voláme dotyčnému readeru (resp. writeru) metody

Vstup a výstup II

Textový soubor některé metody tříd `StreamReader a StreamWriter`

- `void r.Close() // zavře reader,`
- `string r.ReadToEnd() //přečte soubor do konce,`
- `w.Write(co) //vypíše do writeru,`
- `r.Read() // přečte znak z readeru,`
- `r.EndOfStream // atribut určující konec vstupního streamu`
- @"....." // pevný řetězec – nefungují v něm sekvence $\backslash n, \backslash r, \ldots$.

Vstup a výstup II

Příklad

Zkopírování souboru

```
System.IO.StreamReader r=new
System.IO.StreamReader("soubor.txt");
System.IO.StreamWriter w=new
System.IO.StreamWriter("lacina_kopie.txt");
w.Write(r.ReadToEnd()); w.Close();
```

Vstup a výstup III

Příklad

Zkopírování souboru po znacích `System.IO.StreamReader`

```
r=new System.IO.StreamReader("soubor.txt");
System.IO.StreamWriter w=new
System.IO.StreamWriter("lacina_kopie.txt");
while(!r.EndOfStream) w.Write((char)r.Read());
w.Close();
```

Poznámky

- Metody `ReadLine()`, `resp. WriteLine()`,
- kódování češtiny – lze předat jako druhý parametr konstruktoru Readeru/Writeru:
- Příklad:

```
Encoding e=Encoding.GetEncoding(1250);
Encoding f=Encoding.GetEncoding(852);
System.IO.StreamReader r=new
System.IO.StreamReader("soubor.txt",e);
System.IO.StreamWriter w=new
System.IO.StreamWriter("lacina_kopie.txt",f);
w.Write(r.ReadToEnd()); w.Close();
```

Komplexní čísla

a operace s nimi, aneb přetížené operátory

- Čísla jsou různá: Přirozená, celá, racionální, reálná, komplexní...
- Na všech ale má smysl definovat sčítání, násobení, odčítání a dělení.
- V počítači máme jen čísla celá a necelá. Co s tím?
- Definujeme si vlastní třídu. Tu ale nepůjde sčítat a násobit. Anebo - ze by?
- Ano: Dotyčné třídě přetížíme operátory (přesněji dodefinujeme je).
- Syntakticky se tváříme, jako bychom definovali běžnou statickou metodu, tato funkce se ale bude divně jmenovat.

Příklad

Opět Gaussova celá čísla

```
class kompl
{    public int re,im;
     public kompl(int re,int im)
     {    this.re=re; this.im=im;}
     public static kompl operator + (kompl a,kompl b)
     {    return new kompl(a.re+b.re,a.im+b.im);}
     public static kompl operator * (kompl a,kompl b)
     {    return new kompl(a.re*b.im-a.im*b.im,
               a.re*b.im+a.im*b.re);
     }
}
```

Příklad pokračování

Abychom mohli třídu `kompl` demonstrovat, předefinujeme jí virtuální metodu `ToString` jako minule:

```
public override string ToString()
{    return ""+re+"+ "+im+"i";}
```

A jedeme:

```
kompl a=new kompl(1,0), b=new kompl(0,1),c;
c=a+b;
Console.WriteLine(c);
Console.WriteLine(a*b);
```

Přetížitelné operátory

Přetížit lze mnoho operátorů, konkrétně:
unární `+`, `-`, `!`, `~`, `++`, `--`
a binární `+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `<<`, `>>`...

Konec

Děkuji za pozornost...