Objektově orientované programování

Martin Pergel

2. října 2012

Pragmatické informace

- Volitelný předmět,

Pragmatické informace

- Volitelný předmět,
- zápočet: zápočtový program (s dokumentací), aktivní účast na cvičení (body v CodExu), praktický test,

Pragmatické informace

- Volitelný předmět,
- zápočet: zápočtový program (s dokumentací), aktivní účast na cvičení (body v CodExu), praktický test,
- zkouška: zkoušková písemka na objektový návrh a bude zřejmě výrazně přihlíženo ke způsobu plnění zápočtových povinností.

Pragmatické informace

- Volitelný předmět,
- zápočet: zápočtový program (s dokumentací), aktivní účast na cvičení (body v CodExu), praktický test,
- zkouška: zkoušková písemka na objektový návrh a bude zřejmě výrazně přihlíženo ke způsobu plnění zápočtových povinností.
- Okamžikem udělení hodnocení u zkoušky jsou zápočtové povinnosti zmraženy!

Cíle předmětu

- Programování v C#,

Cíle předmětu

- Programování v C#,
- objektové programování,

Cíle předmětu

- Programování v C#,
- objektové programování,
- moderní vývojové prostředky.

Souvislosti

- Prvácké Programování II s kódem NPRG je výrazně odlišné od Programování 2 s kódem NPRM, resp. NMIN,

Souvislosti

- Prvácké Programování II s kódem NPRG je výrazně odlišné od Programování 2 s kódem NPRM, resp. NMIN,
- proto matematici nejsou schopni navštěvovat prakticky žádné navazující přednášky o programování.

Souvislosti

- Prvácké Programování II s kódem NPRG je výrazně odlišné od Programování 2 s kódem NPRM, resp. NMIN,
- proto matematici nejsou schopni navštěvovat prakticky žádné navazující přednášky o programování.
- Budeme masívně stavět na Pascalu.

Prostředky

- Budeme používat CodEx,

Prostředky

- Budeme používat CodEx,
- využívat budeme buďto Microsoft Visual Studio (v dostupné verzi),

Prostředky

- Budeme používat CodEx,
- využívat budeme buďto Microsoft Visual Studio (v dostupné verzi),
- nebo prostředí MONO (obojí je k dispozici legálně zdarma aspoň v omezené verzi).

Prostředky

- Budeme používat CodEx,
- využívat budeme buďto Microsoft Visual Studio (v dostupné verzi),
- nebo prostředí MONO (obojí je k dispozici legálně zdarma aspoň v omezené verzi).
- Tvořit budeme převážně konzolové aplikace (o psaní formulářových si taktéž něco řekneme).

Literatura

- M. Virius: C# pro zelenáče, Neocortex Praha, 2002,
- E. Gunnerson: Začínáme programovat v C#, Computer Press Praha, 2001,
- J. Kent: Visual C# 2005 bez předchozích znalostí, Computer Press Brno, 2007,
- J. Liberty, D. Xie: Programming C# 3.0, Fifth Edition, O'Reilly Media Inc., 2007.

Objektové programování

aneb proč nebyl C# v prváku?

- Základy objektového programování byly i v Pascalu, všude je to ale podobné.

Objektové programování

aneb proč nebyl C# v prváku?

- Základy objektového programování byly i v Pascalu, všude je to ale podobné.
- Vše je objekt, vše má metody a atributy se všemi důsledky, které z toho plynou.

Objektové programování

aneb proč nebyl C# v prváku?

- Základy objektového programování byly i v Pascalu, všude je to ale podobné.
- Vše je objekt, vše má metody a atributy se všemi důsledky, které z toho plynou.
- I program je objekt (s metodami). Vždy je definováno, jak se s objektem programu zachází.

Objektové programování

aneb proč nebyl C# v prváku?

- Základy objektového programování byly i v Pascalu, všude je to ale podobné.
- Vše je objekt, vše má metody a atributy se všemi důsledky, které z toho plynou.
- I program je objekt (s metodami). Vždy je definováno, jak se s objektem programu zachází.
- Program neběží odnikud nikam, ale spustí se mu metoda `Main`.

Krok stranou

Rodina jazyka C

- B. Kernighan a D. Ritchie napřed navrhli jazyk A...

Krok stranou

Rodina jazyka C

- B. Kernighan a D. Ritchie napřed navrhli jazyk A...
- tým byl spokojen až s jazykem C.

Krok stranou

Rodina jazyka C

- B. Kernighan a D. Ritchie napřed navrhli jazyk A...
- tým byl spokojen až s jazykem C.
- Ač byl tento jazyk navrhován údajně jako recese, velmi se ujal.

Krok stranou

Rodina jazyka C

- B. Kernighan a D. Ritchie napřed navrhli jazyk A...
- tým byl spokojen až s jazykem C.
- Ač byl tento jazyk navrhován údajně jako recese, velmi se ujal.
- Za jeho potomky lze prohlásit C++, Java, C#, Javascript, PHP, Python a další.

Společné vlastnosti

- Jsou case-sensitive,
- mají podobnou syntax,
- u neobjektových jazyků se zpravidla spustí funkce `main` (až na velikost písmen).

Krok stranou

Jazyk C

hello.c

```
#include <stdio.h>
int main()
{     printf("Hallo, world!\n");
}
```

Krok stranou

Jazyk C

k_nicemu.c

```
#include <stdio.h>
void f()
{     printf("Hallo, world!\n");
}
int main()
{     f();
}
```

Syntax jazyka C

definice proměnných

- *Datový typ se píše napřed!*
- `int faktorial(int a)`
- `{  int b=1;`
- `  while(a>1)b*=a--;`
- `  return b;`
- `}`

Nejdůležitější typy

- `void` – prázdný datový typ – zahození hodnoty,
- `char` – obvykle jeden byte – měřák velikosti,
- `short` – celočíselný typ,
- `int` – nejobvyklejší celočíselný typ,
- `long` – celočíselný typ,
- `float` – neceločíselný typ,
- `double` – neceločís. typ s dvojnásobnou (doublovou) přesností,
- **C#** `string` – řetězec **C – ne C#** pointery.

Příklad

- `int a,b,c=10,d=100;`

Příklad

- `int a,b,c=10,d=100;`
- `double x=1.15;`

Příklad

- `int a,b,c=10,d=100;`
- `double x=1.15;`
- `char a='a',b='x';`

Příklad

- `int a,b,c=10,d=100;`
- `double x=1.15;`
- `char a='a',b='x';`
- `string text="nic";`

Příklad

- `int a,b,c=10,d=100;`
- `double x=1.15;`
- `char a='a',b='x';`
- `string text="nic";`
- `void nanic(int a, int b, int c);`

Příklad

- `int a,b,c=10,d=100;`
- `double x=1.15;`
- `char a='a',b='x';`
- `string text="nic";`
- `void nanic(int a, int b, int c);`
- **parametry funkce při definici se oddělují čárkou.**

Příklad

- `int a,b,c=10,d=100;`
- `double x=1.15;`
- `char a='a',b='x';`
- `string text="nic";`
- `void nanic(int a, int b, int c);`
- **parametry funkce při definici se oddělují čárkou.**
- **konstanty modifikátorem** `const: const double pi=3.1415926;`

Základní operátory

- + (binární) sčítání,

Základní operátory

- + (binární) sčítání,
- − odečítání, podobně ∗ a /

Základní operátory

- + (binární) sčítání,
- − odečítání, podobně ∗ a /
- = přiřazení,

Základní operátory

- `+` (binární) sčítání,
- `-` odečítání, podobně `*` a `/`
- `=` přiřazení,
- `==` porovnání na rovnost, `!=`, `>`, `<`, `>=`, `<=` (nerovnosti),

Základní operátory

- + (binární) sčítání,
- - odečítání, podobně * a /
- = přiřazení,
- == porovnání na rovnost, !=, >, <, >=, <= (nerovnosti),
- logické && (and), || (or), ! (not), & (and), | (or), ^ (xor),

Základní operátory

- + (binární) sčítání,
- - odečítání, podobně * a /
- = přiřazení,
- == porovnání na rovnost, !=, >, <, >=, <= (nerovnosti),
- logické && (and), || (or), ! (not), & (and), | (or), ^ (xor),
- první dvě vyhodnocují líně (je-li výsledek jasný, přestanou), 4. a 5. vyhodnocují úplně (takto v C#, v C to bylo trochu jinak).

Základní operátory

- + (binární) sčítání,
- - odečítání, podobně * a /
- = přiřazení,
- == porovnání na rovnost, !=, >, <, >=, <= (nerovnosti),
- logické && (and), || (or), ! (not), & (and), | (or), ^ (xor),
- první dvě vyhodnocují líně (je-li výsledek jasný, přestanou), 4. a 5. vyhodnocují úplně (takto v C#, v C to bylo trochu jinak).
- unární ++, -- (prefixové a postfixové, tedy a++ vs ++a).

Základní operátory

- + (binární) sčítání,
- - odečítání, podobně * a /
- = přiřazení,
- == porovnání na rovnost, !=, >, <, >=, <= (nerovnosti),
- logické && (and), || (or), ! (not), & (and), | (or), ^ (xor),
- první dvě vyhodnocují líně (je-li výsledek jasný, přestanou), 4. a 5. vyhodnocují úplně (takto v C#, v C to bylo trochu jinak).
- unární ++, -- (prefixové a postfixové, tedy a++ vs ++a).
- Pozor na priority!

Základní operátory

- + (binární) sčítání,
- - odečítání, podobně * a /
- = přiřazení,
- == porovnání na rovnost, !=, >, <, >=, <= (nerovnosti),
- logické && (and), || (or), ! (not), & (and), | (or), ^ (xor),
- první dvě vyhodnocují líně (je-li výsledek jasný, přestanou), 4. a 5. vyhodnocují úplně (takto v C#, v C to bylo trochu jinak).
- unární ++, -- (prefixové a postfixové, tedy a++ vs ++a).
- Pozor na priority!
- Řešení jako v Pascalu, tedy závorkovat.

Základní operátory

- + (binární) sčítání,
- - odečítání, podobně * a /
- = přiřazení,
- == porovnání na rovnost, !=, >, <, >=, <= (nerovnosti),
- logické && (and), || (or), ! (not), & (and), | (or), ^ (xor),
- první dvě vyhodnocují líně (je-li výsledek jasný, přestanou), 4. a 5. vyhodnocují úplně (takto v C#, v C to bylo trochu jinak).
- unární ++, -- (prefixové a postfixové, tedy a++ vs ++a).
- Pozor na priority!
- Řešení jako v Pascalu, tedy závorkovat.
- Pozor na porovnání a přiřazení!

Základní operátory

- + (binární) sčítání,
- - odečítání, podobně * a /
- = přiřazení,
- == porovnání na rovnost, !=, >, <, >=, <= (nerovnosti),
- logické && (and), || (or), ! (not), & (and), | (or), ^ (xor),
- první dvě vyhodnocují líně (je-li výsledek jasný, přestanou), 4. a 5. vyhodnocují úplně (takto v C#, v C to bylo trochu jinak).
- unární ++, -- (prefixové a postfixové, tedy a++ vs ++a).
- Pozor na priority!
- Řešení jako v Pascalu, tedy závorkovat.
- Pozor na porovnání a přiřazení!
- Přiřazovací příkazy: +=, -=, *=, /=, &= apod.

Komentáře, bloky

- Jednořádkové `// do konce řádku`

Komentáře, bloky

- Jednořádkové `// do konce řádku`
- Víceřádkové `/* komentář hodně dlouhý dokud ho neukončíme explicitně */`

Komentáře, bloky

- Jednořádkové `// do konce řádku`
- Víceřádkové `/* komentář hodně dlouhý dokud ho neukončíme explicitně */`
- Složené závorky označují blok (jako v Pascalu `begin` a `end`), tedy { `blok; příkazů;`}

Základní řídicí struktury

- `if (podm) příkaz;`

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`
- `if (podm) { blok } [else {blok}]`

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`
- `if (podm) { blok } [else {blok}]`
- `while(podm) příkaz_nebo_blok`

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`
- `if (podm) { blok } [else {blok}]`
- `while(podm) příkaz_nebo_blok`
- `for(init;podm;inkr) tělo`

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`
- `if (podm) { blok } [else {blok}]`
- `while(podm) příkaz_nebo_blok`
- `for(init;podm;inkr) tělo`
- *`init` je inicializační kód, `podm` je podmínka, `inkr` je inkrementační výraz, tedy příklad:*

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`
- `if (podm) { blok } [else {blok}]`
- `while(podm) příkaz_nebo_blok`
- `for(init;podm;inkr) tělo`
- *init je inicializační kód, podm je podmínka, inkr je inkrementační výraz, tedy příklad:*
- `for(a=1;a<10;a++)b+=a;`

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`
- `if (podm) { blok } [else {blok}]`
- `while(podm) příkaz_nebo_blok`
- `for(init;podm;inkr) tělo`
- *init je inicializační kód, podm je podmínka, inkr je inkrementační výraz, tedy příklad:*
- `for(a=1;a<10;a++)b+=a;`
- *Co je tohle?* `if(a=5) a_je_pet();`

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`
- `if (podm) { blok } [else {blok}]`
- `while(podm) příkaz_nebo_blok`
- `for(init;podm;inkr) tělo`
- *init je inicializační kód, podm je podmínka, inkr je inkrementační výraz, tedy příklad:*
- `for(a=1;a<10;a++)b+=a;`
- *Co je tohle?* `if(a=5) a_je_pet();`
- `do příkazy(); while(podm);`

Základní řídicí struktury

- `if (podm) příkaz;`
- `if (podm) příkaz; else příkaz;`
- `if (podm) { blok } [else {blok}]`
- `while(podm) příkaz_nebo_blok`
- `for(init;podm;inkr) tělo`
- *init je inicializační kód, podm je podmínka, inkr je inkrementační výraz, tedy příklad:*
- `for(a=1;a<10;a++)b+=a;`
- *Co je tohle?* `if(a=5) a_je_pet();`
- `do příkazy(); while(podm);`
- *Poznámka:* `a=b=c=1;`

Konstrukce `switch`

ekvivalent `case ... of`

```
switch(i)
{       case 1: i_je_jedna(); break;
        case 2: case 3: i_je_2_nebo_3(); break;
        default: vsechno_je_jinak(); break;
}
```

Break ukončuje jednotlivé větve, v C se vykonával kód, dokud se nenarazí na `break`, C# toto neumožňuje (jednu větev je třeba ukončit před začátkem druhé).

Definice funkcí

```
typ jmeno(parametry)
{ telo }
```

Chceme-li vrátit návratovou hodnotu, použijeme klíčové slovo `return` a za ně umístíme vracený výraz: `return 0;`

Definice funkcí

podrobnosti

- procedura je funkce, která vrací `void`,

Definice funkcí

podrobnosti

- procedura je funkce, která vrací `void`,
- `void` `vracíme` pomocí `return;`,

Definice funkcí

podrobnosti

- procedura je funkce, která vrací `void`,
- `void` `vracíme` pomocí `return;`,
- kulaté závorky jsou operátor zavolání (nebo operátor definice funkce), nelze je tudíž oproti Pascalu vynechat,

Definice funkcí

podrobnosti

- procedura je funkce, která vrací `void`,
- `void` `vracíme` pomocí `return;`,
- kulaté závorky jsou operátor zavolání (nebo operátor definice funkce), nelze je tudíž oproti Pascalu vynechat,
- Parametry jsou implicitně předávány hodnotou. Jazyk C předání referencí neměl (místo toho byly pointery).

Definice funkcí

podrobnosti

- procedura je funkce, která vrací `void`,
- `void` `vracíme` pomocí `return;`,
- kulaté závorky jsou operátor zavolání (nebo operátor definice funkce), nelze je tudíž oproti Pascalu vynechat,
- Parametry jsou implicitně předávány hodnotou. Jazyk C předání referencí neměl (místo toho byly pointery).
- C# umí předávat parametry též referencí a výsledkem:

Definice funkcí

podrobnosti

- procedura je funkce, která vrací `void`,
- `void` `vracíme` pomocí `return;`,
- kulaté závorky jsou operátor zavolání (nebo operátor definice funkce), nelze je tudíž oproti Pascalu vynechat,
- Parametry jsou implicitně předávány hodnotou. Jazyk C předání referencí neměl (místo toho byly pointery).
- C# umí předávat parametry též referencí a výsledkem:
- `void divna(ref int a, out int vysledek);`

Definice funkcí

podrobnosti

- procedura je funkce, která vrací `void`,
- `void` `vracíme` pomocí `return;`,
- kulaté závorky jsou operátor zavolání (nebo operátor definice funkce), nelze je tudíž oproti Pascalu vynechat,
- Parametry jsou implicitně předávány hodnotou. Jazyk C předání referencí neměl (místo toho byly pointery).
- C# umí předávat parametry též referencí a výsledkem:
- `void divna(ref int a, out int vysledek);`
- proměnná `a` je předána referencí, proměnná `vysledek` výsledkem, což je téměř totéž (tedy je řešena též odkazem, ale hodnota proměnné předávané výsledkem není funkci vůbec předána).

Objektové programování

- Objekty jsou instance třídy (jako v Pascalu).

Objektové programování

- Objekty jsou instance třídy (jako v Pascalu).
- Statické metody (resp. atributy) příslušejí třídě (ne jednotlivým objektům).

Objektové programování

- Objekty jsou instance třídy (jako v Pascalu).
- Statické metody (resp. atributy) příslušejí třídě (ne jednotlivým objektům).
- Modifikátory `public`, `private`, `protected` řídí přístup k jednotlivým položkám (může každý / může jen metoda dané třídy / smí současná třída a potomek).

Objektové programování

- Objekty jsou instance třídy (jako v Pascalu).
- Statické metody (resp. atributy) příslušejí třídě (ne jednotlivým objektům).
- Modifikátory `public`, `private`, `protected` řídí přístup k jednotlivým položkám (může každý / může jen metoda dané třídy / smí současná třída a potomek).
- Třídu (v C++, Javě, C# a dalších) vytváříme pomocí klíčového slova `class`.

Program v C#

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{    class Program
     {    static void Main(strings[] args)
          {    Console.WriteLine("Sem se bude psát
ten program!");
          }
     }
}
```

Operátor tečky má podobný význam jako v Pascalu.

Eukleidův algoritmus

```
static void Main(string[] args)
{    Console.WriteLine("Dvě čísla sem:);
     int a=int.Parse(Console.ReadLine());
     int b=int.Parse(Console.ReadLine());
     while(a!=b)
          if(a>b) a-=b;
          else b-=a;
     Console.Write("NSD je: ");
     Console.WriteLine(a);
}
```