Jazyk C# 1

3. seminář

Jakub Večeřa

Univerzita Palackého v Olomouci

11. 10. 2024

Reakce na úkoly

- Použití `StringBuilder`
- (Ne)optimalizace
- Dodržení názvů, výstupů
- Kdo vyřešil elegantně?

Pole (1/2)

- Pole je datová struktura, která obsahuje několik elementů stejného datového typu
- Fixní délka!
- Deklarace, inicializace a přiřazení hodnot: `int[] pole = new int[] {2,4,6};`
- Případně lze i bez `new`: `int[] pole = {1, 3, 4, 6, 7};`
- Indexováno od nuly, přístup pomocí operátoru `[]`

```
1 int prvniPrvek = pole[0]; // zjisteni prvniho prvku
2 pole[2] = 64; // nastaveni tretiho prvku na 64
```

- Vytvoření prázdného pole bool délky 20: `bool[] pBool = new bool[20];`
- Přístup k délce pole pomocí vlastnosti `Length` – `pBool.Length`

Pole (2/2)

- Procházení pole pomocí cyklu:

```
1 for (int i=0; i<pBool.Length; i++) {
2    bool prvek = pole[i];
3 }
```

- Procházení, pokud nepotřebujeme znát index

```
1 foreach (bool prvek in pBool) {
2 if (prvek)
3 {
4 ...
5 }
6 }
```

Vícedimenzionální pole (1/2)

- Více dimenzionální(rank) pole: `int[,,] pole3d = new int[3,6,9];`
- Přístup pomocí indexeru s čárkou: `int[2,1,3]`
- Inicializace `int[,] pole2d = new int[3, 2] {{1, 2}, { 3,4}, { 5,6}};`
- Zjištění počtu dimenzí: `pole2d.Rank` $\Rightarrow$ 2
- Zjištění délky jednotlivých dimenzí `pole3d.GetLength(1)` $\Rightarrow$ 6
- Pouze pro „obdélníková pole“. Kolik může být dimenzí?
- `foreach` přes prvky

Vícedimenzionální pole (2/2)

- „neobdélníková pole“ jako pole polí: `int[][] polePoli = new int[3][];`
- Naplnění řádky různých velikostí:

```
1 polePoli[0] = new int[] {1, 2, 3, 4, 5};
2 polePoli[1] = new int[] {6, 7, 8};
3 polePoli[2] = new int[] {1};
```

- Jaká bude proměnná ve `foreach`?

Objektově orientované programování

- Co znáte z Paradigmat programování 3?

Objektově orientované programování

- Hlavní paradigma v C# (vedle imperativního)
- Vychází z (přirozeného) náhledu na svět – objekt má:
  - Nějaké vlastnosti
  - Nějak se chová navenek
  - Nějak může fungovat uvnitř
  - Může být složen z menších objektů
- Rozdělení programu na ucelené funkční celky
- Pevné dodržení vnějšího chování objektů umožňuje
  - Rozdělit implementaci programu mezi více programátorů
  - Snadnější pozdější úpravy

Hlavní prvky OOP

- Zapouzdření

Hlavní prvky OOP

- Zapouzdření
- Polymorfismus
- Dědičnost
- Rozhraní objektů

Třídy v C#

- Třída = formální popis objektu – jaké má mít vlastnosti, co má umět a jak má fungovat
- Nepsané pravidlo – každá třída definována ve svém souboru
- Prvky třídy:
  - *Fields* (sloty) – datové prvky, „proměnné“
  - *Konstanty* – datové prvky spjaty se třídou, sdílené pro všechny instance
  - *Metody* – funkce spjaté s danou třídou
  - *Properties* (vlastnosti) – funkce pro manipulaci s privátními sloty
  - *Konstruktory* – metody volané automaticky při vytváření instance, stejné jméno, bez návratové hodnoty
  - *Destruktor* – metoda volaná při mazání objektu z paměti
  - *Podtřídy* – Ve třídě se může definovat interně definovaná třída (neviditelná „z venku“)

Deklarace třídy

- Klíčové slovo `class` + název + blok kódu

```
1  public class HardDiskDrive {
2  public const string Name = "Hard disk drive";
3  private uint capacity;
4  private string manufacturer;
5  private string partNumber;
6  public uint BufferSize;
7  public uint WriteSpeed;
8
9  public string StoreName => manufacturer + partNumber;
10
11 public uint GetFreeSpace() { ... }
12 public bool WriteNumber(int number, uint position) { ... }
13 public int ReadNumber(uint position) { ... }
14 public void ParkReadHead() { ... }
15
16 }
```

Konstruktor

- Metody pojmenované stejně jako třída, bez návratové hodnoty

```
1 public HardDiskDrive(uint capacity, string manufacturer,
2 string partNumber, uint bufferSize, uint writeSpeed)
3   {
4       this.capacity = capacity;
5       this.manufacturer = manufacturer;
6       this.partNumber = partNumber;
7       BufferSize = bufferSize;
8       WriteSpeed = writeSpeed;
9    }
```

- Vytvoření nových instancí:

```
1 HardDiskDrive disk1 = new HardDiskDrive(1000000, "WD",
    "SX04210345",1000, 10000);
2 HardDiskDrive disk2 = new HardDiskDrive(50000, "SEAGATE",
    "725a22s5",333, 8547);
```

Volání metod

- Metody voláme pomoci tečky `disk1.WriteNumber(42, 42468);`
- Třída může obsahovat více metod se stejným názvem, ale jinými parametry - „přetěžování metod“

```
1 public void ParkReadHead(int delay) {
2 }
```

- Může obsahovat i více konstruktorů
- Našeptávač potom umožňuje vybrat verzi metody

Statické metody

- Metody, které nejsou závislé na konkrétní instanci
- Měly by se vždy chovat stejně
- Zpřístupňují (nějakou) funkcionalitu spjatou se třídou
- Např: `Math.Sqrt(25)` $\Rightarrow$ vypočítá odmocninu z 25
- Při deklaraci se použije klíčové slovo `static`
- Nejznámější statickou metodou je `main`

Counter v C#

- Ukázka kódu

Úkol (1/2)

- Navrhněte třídu `IntMatrix`, která bude sloužit k reprezentaci číselné 2D matice
- Interně si bude pamatovat své rozměry (zadané v konstruktoru)
- Implementujte metody:
  1. `public int NRows()` – vrátí počet řádků
  2. `public int NCols()` – vrátí počet sloupců
  3. `public int GetVal(int x, int y)` – vrátí hodnotu na souřadnicích `x,y`
  4. `public void SetVal(int x, int y, int val)` – nastaví hodnotu na souřadnicích `x,y` na `val`
  5. `public int NonZero()` – vrátí počet nenulových prvků matice
  6. `public int SumOfVals()` – vrátí součet všech prvků matice

Úkol (2/2)

- Dále implementujte metody:
  7. `public IntMatrix AddMatrix(IntMatrix mat)` – vrátí novou matici, která je součtem původní matice a matice `mat`
  8. `public IntMatrix MulMatrix(IntMatrix mat)` – vrátí novou matice, která je součinem původní matice a matice `mat`
  9. `public void Print()` – vypíše matici do konzole
- A Implementujte statickou metodu:
  10. `public static IntMatrix Ones(int nRows, int nCols)` – vytvoří matici plnou jedniček o velikosti $nRows \times nCols$
- Vhodně ošetřete velikosti matic vstupujících do operací

Úkol – příklad (1/2)

```
1 IntMatrix prvni = new IntMatrix(2,2);
2 prvni.SetVal(0,0,2); prvni.SetVal(0,1,4);
3 prvni.SetVal(1,0,6); prvni.SetVal(1,1,8);
4 prvni.Print();
```

==>

```
2 4
6 8
```

```
1 IntMatrix druha = new IntMatrix(2,2);
2 druha.SetVal(0,0,1); druha.SetVal(0,1,3);
3 druha.SetVal(1,0,5); druha.SetVal(1,1,7);
4 druha.Print();
```

==>

```
1 3
5 7
```

Úkol – příklad (2/2)

```
1 IntMatrix treti = prvni.AddMatrix(druha);
2 treti.Print();
```

```
===>
    3  7
    11 15
```

```
1 Console.WriteLine(treti.SumOfVals()); ==> 36
2 Console.WriteLine(treti.NonZero()); ==> 4
3 treti.SetVal(0,0,0);
4 Console.WriteLine(treti.NonZero()); ==> 3
5 IntMatrix dalsi = IntMatrix.Ones(3, 3);
6 dalsi.Print();
```

```
==>
1 1 1
1 1 1
1 1 1
```