Jazyk C# 1

7. seminář

Jakub Večeřa

Univerzita Palackého v Olomouci

8. 11. 2024

Reakce na úkoly

- Kontrolujte si zprávy v chatu – u neohodnocených úkolů čekám na reakce od studentů
- $\Rightarrow$ V případě nejasností mě můžete kontaktovat emailem nebo přijít na seminář
- `CompareTo()==0` vs. `Equals()`
- Generika – být si jistý, co má být návratovou hodnotou a co má být parametrem
- Nevyužití možnosti `foreach`
- Enumerátor

Výčtový datový typ – enum (1 / 3)

- Bývá potřeba mít vlastní datový typ, který může nabývat pouze určitých hodnot
- $\Rightarrow$ Výčtový datový typ `enum`
- Deklarace:

```
1 public enum ComputerFormFactor {
2 Atx,
3 BigTower,
4 Laptop,
5 MiniPC,
6 Embeded
7 }
```

- Poté můžeme náš datový typ normálně používat, přiřazení hodnot staticky typované

```
1 ComputerFormFactor ff = ComputerFormFactor.BigTower;
2 Console.WriteLine(ff);
```

Výčtový datový typ – enum (2 / 3)

- Formálně je „na pozadí“ enumu datový typ `int`
- Hodnoty jsou pojmenované konstanty, číslované od nuly
- Funguje přetypování (na obě strany)

```
1 ComputerFormFactor dalsi = (ComputerFormFactor)2;
2 int a = (int) dalsi;
3 Console.WriteLine(dalsi);
```

- Datový typ „na pozadí“ můžeme změnit, stejně jako specifikovat hodnoty

```
1 public enum UsbConnector : short {
2 mini = 1,
3 micro = 3,
4 c = 4,
5 standardA = 7
6 }
```

Výčtový datový typ – enum (3 / 3)

- Lze přiřadit více hodnot – bitová maska
- Musíme použít *atribut* `Flags`
- Přiřazení více hodnot pomocí bitového součtu `|`

```
1 [Flags]
2 public enum DaysOfWeek {
3 Monday = 0x1, Tuesday = 0x2, Wednesday = 0x4, Thursday = 0x8,
4 Friday = 0x10, Saturday = 0x20, Sunday = 0x40,
5 Weekend = Saturday | Sunday, Workday = 0x1f
6 }
7 DaysOfWeek d = DaysOfWeek.Workday;
8 Console.WriteLine(d);
```

- Přístup k názvům: `Enum.GetNames(typeof(DaysOfWeek))`
- Přístup k hodnotám `Enum.GetValues(typeof(daysOfWeek))`

Extensions methods (1 / 2)

- Máme nějakou třídu, chceme rozšířit její funkcionalitu
  - Můžeme vytvořit novou a podědit z předchozí
  - Tím ale nedostaneme funkcionalitu k instancím původní
  - Původní třída může být `sealed` a nejde z ní dědit
  - Mohli bychom si vytvořit statické metody, které by jako první argument měly onu třídu

Extensions methods (1 / 2)

- Máme nějakou třídu, chceme rozšířit její funkcionalitu
  - Můžeme vytvořit novou a podědit z předchozí
  - Tím ale nedostaneme funkcionalitu k instancím původní
  - Původní třída může být `sealed` a nejde z ní dědit
  - Mohli bychom si vytvořit statické metody, které by jako první argument měly onu třídu
- Vše takové polovičaté $\Rightarrow$ Mechanismus Extensions methods

Extensions methods (1 / 2)

- Máme nějakou třídu, chceme rozšířit její funkcionalitu
  - Můžeme vytvořit novou a podědit z předchozí
  - Tím ale nedostaneme funkcionalitu k instancím původní
  - Původní třída může být `sealed` a nejde z ní dědit
  - Mohli bychom si vytvořit statické metody, které by jako první argument měly onu třídu
- Vše takové polovičaté ⇒ Mechanismus Extensions methods
- Mějte statickou třídu a v ní napišme statickou metodu
- Přidáním slova `this` u prvního parametru rozšíříme danou třídu o naši metodu

```
1 public static class StatickeMetody {
2   public static int GetWordCount(this string s) {
3     return s.Split().Length;
4    }
5  }
```

Extensions methods (2 / 2)

- Volání je pak možné, klasicky, přes tečku

```
1 string s = "nejaka dlouha veta, kde budu pocitat slova";
2 Console.WriteLine($"Retezec ma {s.GetWordCount()} slov");
```

- Můžeme rozšířit jakýkoliv typ

```
1 public static bool IsBellowZero(this int i) {
2     return i < 0;
3 }
4 int i = 33;
5 Console.WriteLine(i.IsBellowZero());
```

- Používat s rozvahou (nemusí být snadno dohledatelné, kde je daná metoda implementována)
- Nelze přepsat již existující metodu
- Musí být naincludována pomocí `using`

DateTime (1 / 3)

- Hodnotový datový typ určený pro práci s datem
- Rozsah: od půlnoci 1. ledna roku 1, do půlnoci 31. prosince roku 9999
- Interně reprezentováno počtem *tiků*(100ns) od začátku letopočtu v Gregoriánském kalendáři
- Vytvoření:

```
1 DateTime begin = new DateTime(); // pocatek veku
2 DateTime someDate = new DateTime(2023, 9, 6); // specificke datum
3 DateTime now = DateTime.Now; // Nyni (na milisekundy)
4 DateTime today = DateTime.Today; // Dnesni punoc
5 string dateString = "03.05.2023 20:23:55";
6 DateTime parsedDate = DateTime.ParseExact(dateString,
7     "MM.dd.yyyy HH:mm:ss",
8     System.Globalization.CultureInfo.InvariantCulture);
```

DateTime (2 / 3)

- Výpis záleží na použitém jazyku

```
1 Console.WriteLine(begin); // vychozi
```

- Specifický jazyk (Francouzština)

```
1 Console.WriteLine(someDate.ToString(
2 System.Globalization.CultureInfo.CreateSpecificCulture("fr-FR")));
```

- ToString s argumentem formátovacího řetězce
  `https://docs.microsoft.com/en-us/dotnet/api/system.datetime.tostring?view=net-5.0#System_DateTime_ToString_System_String_`

```
1 Console.WriteLine(now.ToString("D"));
2 Console.WriteLine(today.ToString("MM.dd.yyyy HH:mm:ss"));
```

- Spousta přetížených metod jak pro formát, tak pro vytvoření

DateTime (3 / 3) – Operace

- DateTime můžeme sčítat (`Add`), odčítat (`Subtract`)

```
1 Console.WriteLine($"Od zacatku semestru uplynulo
2 {DateTime.Now.Subtract(new DateTime(2023, 9, 18)).TotalSeconds}
      sekund");
```

- Přičítat k nim hodiny (`AddHour`), minuty (`AddMinutes`), ...
- Dotazovat se na faktické údaje
  - Počet dnů v měsíci – statická metoda `DaysInMonth`
  - Je to letní čas? – `IsDaylightSavingTime()`
  - Je daný rok přestupný? – statická metoda `IsLeapYear`
- Vlastnost `Ticks` – počet *tiků* od začátku věků

Výjimky (obecně)

- Co víte o výjimkách z jiných předmětů?

Výjimky (obecně)

- Co víte o výjimkách z jiných předmětů?
- Mechanismus pro zpracování **výjimečných** stavů programu
- Nízkoúrovňový mechanismus měnící tok běhu programu
- Nejčastější použití:
  - Práce se soubory
  - Ošetření vstupů (uživatel)
  - Práce se zdroji (Databáze, procesor, disk, . . .)
  - Práce se sítí
- Obecně: Tam, kde může něco selhat nebo tam, kde nejsme schopni zajistit správnost/existenci vstupu

Výjimky

- Typickým příkladem je například čtení z pole mimo index nebo dělení nulou

```
1 int[] pole = new int[10];
2 pole[0] = 25;
3 pole[10] = 33;
4
5 int x = 0;
6  Console.WriteLine(6 / x);
```

- Třetí řádek zastaví program a vyhodí výjimku
- K dělení nulou ani nedojde

Výjimky – obsluha

- Obsluha = reakce programu na výjimečnou situaci
- Možnosti:
  - Neošetření
  - Ošetření ignorací – žádná reakce
  - Ošetření – reakce uživateli, vynucení nového vstupu, atd.
  - Ošetření a znovu vyvolání – zalogování výjimky a ošetření nechat na vyšších funkcích
- V C# pomocí `try-catch` bloků

```
1 try {
2  ... // kod, ktery muze vyvolat vyjimku
3 } catch (Exception e) {
4  ... // obsluha vyjimky
5 }
```

- `catch` bloků může být více - hledá se shora dolů

Výjimky – třída Exception

- Třída Exception většinou standardní třídou jazyka
- Nosič informace o výjimečné situaci
- Často „lidsky čitelný“ popis chyby
- Stack trace
- Při debugu i přesný řádek výskytu
- Možnost definice vlastních výjimek (dědičnost)

Výjimky – ošetření

- Ošetření ignorací (nevhodné)

```
try { ...} catch (Exception e) {}
```

- Ošetření – reakce uživateli, vynucení nového vstupu, atd.

```
int readInput() {
   try { ... } catch (Exception e) {
      System.out.println("Nezadali jste spravny vstup");
      return readInput();
   }
}
```

- Ošetření a znovu vyvolání – zalogování výjimky a ošetření nechat na vyšších funkcích – vyhození výjimky pomocí `throw`

```
try {
   ...
} catch (Exception e) {
   Logger.log(e); // zalogovani
   throw e; // znovuvyvolani vyjimky
}
```

Výjimky – úklid zdrojů

- I když se nestane výjimečná situace, je potřeba zavřít zdroje
- Konstrukt `try-catch-finally`
- Kód ve větvi `finally` se provede vždy (kromě použití systémových akci v `catch` bloku)

```
1 try {
2    file = openFile(path)
3    lines = getLinesFromFile(file);
4    matrix = parseMatrixFromLines(lines);
5 } catch (FileNotFoundException e) {
6    // obsluha nedostupnosti souboru
7 } catch (InvalidInputException e) {
8   // obsluha spatnych hodnot pro vytvoreni matice
9 } finally {
10   // Tady uklidim zdroje, zavru soubor, ...
11 }
```

Úkol (1 / 2)

1. Přidejte datovému typu `string` metodu `UpperAfterSpace`, která vrátí řetězec, kde každé slovo bude začínat velkým písmenem

```
1 string s = "nejaka dlouha veta, kde budu zvetsovat pismena;
2 Console.WriteLine(s.UpperAfterSpace());
3 => Nejaka Dlouha Veta, Kde Budu Zvetsovat Pismena"
```

2. Navrhněte enum `ChessPiece`, který bude reprezentovat šachové figurky (žádná, pěšec, věž, jezdec, střelec, dáma, král) včetně barev
3. Naprogramujte statickou metodu `GetBoard` která vrátí 2D(8×8) pole s počáteční deskou šachové partie

Úkol (2 / 2)

4. Naprogramujte statickou metodu `DateTime WhenTimesFactor(DateTime son, DateTime father, int factor)`, která odpoví na otázky typu *"Syn má čtyři roky, otec má 31 let, kdy bude otec přesně 3x starší než syn?"*
   - V případě, že výsledek nexistuje, vyhoďte výjimku

```
1 DateTime result = WhenTimesFactor(sonsBirthday, fatherBirthday,
      3);
2 Console.WriteLine("Otec bude presne 3x starsi nez
3                            syn v datu:" + result);
```

5. Modifikujte vaši prioritní frontu tak, že bude vyhazovat výjimku v případě:
   - Že je prázdná a zavoláme na ni pop
   - Již v ní existuje prvek s danou prioritou $\Rightarrow$ vyhozená výjimka bude obsahovat nejbližší hodnoty vyšší a nižší priority