Jazyk C# 2

10. seminář

Večeřa J., Janoštík R.

Univerzita Palackého v Olomouci

2. 5. 2023

Reflexe – motivace

- Jak byste se současnými znalostmi, bez použití hotových knihoven řešili následující situace:
  - Serializace objektů do formátu `JSON`

Reflexe – motivace

- Jak byste se současnými znalostmi, bez použití hotových knihoven řešili následující situace:
  - Serializace objektů do formátu `JSON`
  - Naprogramování vlastního ORM

Reflexe – motivace

- Jak byste se současnými znalostmi, bez použití hotových knihoven řešili následující situace:
  - Serializace objektů do formátu `JSON`
  - Naprogramování vlastního ORM
  - Parametrizování testovacích metod

Reflexe – motivace

- Jak byste se současnými znalostmi, bez použití hotových knihoven řešili následující situace:
  - Serializace objektů do formátu `JSON`
  - Naprogramování vlastního ORM
  - Parametrizování testovacích metod
  - Zjištění, zda má nějaký objekt danou metodu

Reflexe – motivace

- Jak byste se současnými znalostmi, bez použití hotových knihoven řešili následující situace:
  - Serializace objektů do formátu `JSON`
  - Naprogramování vlastního ORM
  - Parametrizování testovacích metod
  - Zjištění, zda má nějaký objekt danou metodu
- Hodil by se nám nástroj na „získávání informací o kódu, z kódu“

Reflexe – motivace

- Jak byste se současnými znalostmi, bez použití hotových knihoven řešili následující situace:
  - Serializace objektů do formátu `JSON`
  - Naprogramování vlastního ORM
  - Parametrizování testovacích metod
  - Zjištění, zda má nějaký objekt danou metodu
- Hodil by se nám nástroj na „získávání informací o kódu, z kódu“
- Tento mechanismus naštěstí v C# existuje $\Rightarrow$ *Reflexe*

Reflexe

- = proces získávání znalostí o *typech* za běhu programu
- $\Rightarrow$ získávání metadat „o kódu“
- Objektově zapouzdřené
- Budeme používat funkcionalitu z namespace `System.reflection`
- Zaměříme se pouze na získávání metadat o objektech, metodách, atributech

Reflexe – typy

- Nejdříve potřebujeme získat informace o typu objektů
- Máme-li (můžeme-li vytvořit) instanci:

```
1 Custommer c = new Custommer() { ... };
2 Type t = c.GetType();
```

- Nemáme-li (nechceme vytvářet) instanci:

```
1 Type t = typeof(Custommer);
```

- Získání typu z řetězce: `Type t = Type.GetType("NejakaTrida");`
- Velké množství `true/false` hodnot – např.:`IsAbstract`, `IsArray`, `IsClass`, `IsCOMObject`, `IsEnum`, `IsInterface`, `IsPrimitive`, `IsNestedPrivate`, `IsNestedPublic`, `IsSealed`, `IsValueType`

Reflexe – typy – informace

- Získání informací o metodách – `GetMethods()`
  - Vrátí pole `MethodInfo`, ze kterého získáme veškeré informace
  - Jméno, typy parametrů, návratová hodnota, přístupnost, ...

```
1 MethodInfo[] mi = t.GetMethods();
2 foreach (MethodInfo m in mi) {
3   Console.WriteLine($"Metoda: {m.Name} vraci {m.ReturnType}");
4 }
```

- Získání informací o slotech

```
1 FieldInfo[] fis = t.GetFields();
2 foreach (FieldInfo fi in fis) {
3   Console.WriteLine(fi.Name);
4 }
```

- Obdobně `GetProperties()`, `GetInterfaces()`
- Obdobně `m.GetParameters()` pro parametry metod

Atributy

- Předchozí metadata generoval kompilátor, měli jsme je přístupné
- Atributy = způsob, jak dovolit programátorům poskytovat metadata o kódu
  - Tato metadata pak budou přístupná „programu“ skrze reflexi
- S atributy jsme s potkali z uživatelského pohledu
- Atributy = anotace k danému typu, (třídy, rozhraní, struktury, . . .) a jejich prvkům (vlastnosti, metody, sloty, . . .)
- Již jsme se mohli setkat s

  `[HttpGet], [HttpPost], [Serializable], [PrimaryKey], ...`

Atributy – čtení

- Čtení atributů pomocí metody `GetCustomAttributes()`

```
1 FieldInfo[] fis = t.GetFields(BindingFlags.NonPublic |
2                    BindingFlags.Instance);
3 foreach (FieldInfo i in fis) {
4   Console.WriteLine(i.Name);
5   object[] atts = i.GetCustomAttributes(false);
6   foreach (object att in atts) {
7     Console.WriteLine(att.GetType());
8   }
9 }
```

- Případně kontrola na daný atribut

```
1 if (m.GetCustomAttributes(typeof(ThreadStaticAttribute), true)
2                                        .Length > 0)) {
3 // metoda ma atribut
4 }
```

Atributy

- Atributy samy o sobě „nedělají nic“, pouze dodávají informace
- Vždy záleží na tom, kdo je zpracuje a co s tím udělá

Atributy

- Atributy samy o sobě „nedělají nic“, pouze dodávají informace
- Vždy záleží na tom, kdo je zpracuje a co s tím udělá
  - Nějaký kód přes reflexi zjistí atributy (a jejich vlastnosti)
  - Na základě těchto získaných dat může manipulovat s objektem
  - Např. `[Serializable]` – Víme, že třída lze serializovat
  - `[NonSerialized]` – tento slot se serializovat nemá

Atributy

- Atributy samy o sobě „nedělají nic“, pouze dodávají informace
- Vždy záleží na tom, kdo je zpracuje a co s tím udělá
  - Nějaký kód přes reflexi zjistí atributy (a jejich vlastnosti)
  - Na základě těchto získaných dat může manipulovat s objektem
  - Např. `[Serializable]` – Víme, že třída lze serializovat
  - `[NonSerialized]` – tento slot se serializovat nemá
- Velmi často nám jako jedinná informace postačí název atributu
  - Ale atributy mohou mít vlastnosti
  - Nastavovatelné přes konstruktur
  - Mohou mít i metody

Vytvoření vlastních atributů

- Pojmenovávací konvence – suffix `Attribute`
  - Suffix není potřeba uvádět do hrantých závorek, pouze první část jména
- Atributu z bezpečnostního hlediska vytvoříme jako `sealed`

Vytvoření vlastních atributů

- Pojmenovávací konvence – suffix `Attribute`
  - Suffix není potřeba uvádět do hrantých závorek, pouze první část jména
- Atributu z bezpečnostního hlediska vytvoříme jako `sealed`
- Budeme dědit ze `System.Attribute`

```
1 public sealed class MyAttribute : Attribute
2 {
3 }
```

- Omezení použití atributu – pomocí atributu `AttributeUsage`
  - Parametrem je maska z enumu: `AttributeTargets`

```
1 [AttributeUsage(AttributeTargets.Class | AttributeTargets.Field)]
2 public sealed class MyAttribute : Attribute
3 {
4 }
```

Vytvoření vlastních atributů

- Atribut je tedy úplně normální třída
- Ale měl by být používán pouze jako nosič dat

```
1  [AttributeUsage(AttributeTargets.Class)]
2  public sealed class MyORMContextAttribute : Attribute {
3      public Type[] types;
4      public string someData;
5      public int version;
6      public MyORMContextAttribute(Type[] types,
7                 string someData, int version) {
8        this.types = types;
9        this.someData = someData;
10       this.version = version;
11     }
12 }
```

Úskalí a doporučení k reflexi

- Reflexi bychom měli používat pro věci, které nejdou udělat jinak
  - Např. bychom přes ni neměli volat běžně metody

Úskalí a doporučení k reflexi

- Reflexi bychom měli používat pro věci, které nejdou udělat jinak
  - Např. bychom přes ni neměli volat běžně metody
- Snižuje výkon – VM nemůže dělat tolik optimalizací

Úskalí a doporučení k reflexi

- Reflexi bychom měli používat pro věci, které nejdou udělat jinak
  - Např. bychom přes ni neměli volat běžně metody
- Snižuje výkon – VM nemůže dělat tolik optimalizací
- Bezpečnost – v některých prostředích může být zakázaná

Úskalí a doporučení k reflexi

- Reflexi bychom měli používat pro věci, které nejdou udělat jinak
  - Např. bychom přes ni neměli volat běžně metody
- Snižuje výkon – VM nemůže dělat tolik optimalizací
- Bezpečnost – v některých prostředích může být zakázaná
- Lze získat hodnoty a volat *non-public* členy tříd

Závěr

- Jak byste nyní řešili:
  - Serializace objektů do formátu `JSON`

Závěr

- Jak byste nyní řešili:
  - Serializace objektů do formátu `JSON`
  - Naprogramování vlastního ORM

Závěr

- Jak byste nyní řešili:
  - Serializace objektů do formátu `JSON`
  - Naprogramování vlastního ORM
  - Parametrizování testovacích metod

Závěr

- Jak byste nyní řešili:
  - Serializace objektů do formátu `JSON`
  - Naprogramování vlastního ORM
  - Parametrizování testovacích metod
  - Zjištění, zda má nějaký objekt danou metodu

Úkol

- Zkuste navrhnout a implementovat vlastní ORM knihovnu

Úkol

- Zkuste navrhnout a implementovat vlastní ORM knihovnu
- Komplexnost a funkcionalitu nechám na vašem zvážení

Úkol

- Zkuste navrhnout a implementovat vlastní ORM knihovnu
- Komplexnost a funkcionalitu nechám na vašem zvážení
- Doporučuji zajistit následující funkcionalitu:
  - Definice, které třídy a které jejich *vlastnosti* se budou ukládat do databáze
  - Vytvoření databázového schémata
  - Vložení, editace, smazání instancí

Úkol

- Zkuste navrhnout a implementovat vlastní ORM knihovnu
- Komplexnost a funkcionalitu nechám na vašem zvážení
- Doporučuji zajistit následující funkcionalitu:
  - Definice, které třídy a které jejich *vlastnosti* se budou ukládat do databáze
  - Vytvoření databázového schémata
  - Vložení, editace, smazání instancí
- Komplexnější funkce, elegantnost návrhu může být odměněna více body