

Jazyk C# 1

6. seminář

Radek Janoščík

Univerzita Palackého v Olomouci

31. 10. 2025

Reakce na úkoly

- Efektivita/zvolená datová struktura
- Čitelnost
- Rozhraní používající konkrétní typ

```
1  interface IIntSet {  
2      void Add(int i);  
3      bool Contains(int i);  
4      void Remove(int i);  
5      int Size();  
6      IntSet Union(IntSet other);  
7      IntSet Intersection(IntSet other);  
8      IntSet Subtract(IntSet other);  
9      bool AreEqual(IntSet other);  
10     bool IsSubsetOf(IntSet other);  
11     bool IsEmpty();  
12 }
```

Generické typy – motivace

- Připomeňme si náš zásobník `IntStack`
- Uměl pracovat pouze s celými čísly
- Co kdybychom chtěli mít zásobník na řetězce?
 - ▶ Hloupý přepis celého kódu
 - ▶ 95% zůstane stejných
- Nebo předěláme `IntStack`, aby bral obecně `object`
- Co takhle datový typ parametrizovat?
- $\Rightarrow$ aparát generických datových typů

Vytvoření generické třídy

- Za jméno třídy můžeme do ostrých závorek přidat typ(y) jako parametr(y)

```
1 public class Stack<T>
```

- Generická mohou být i rozhraní
- Poté můžeme název `T` používat v kódu jako typ

```
1 public Stack(uint initCapacity) {  
2     data = new T[initCapacity];  
3 }  
4 public bool Pop(out T a) {  
5     if (!IsEmpty()) {  
6         T value = data[head]; // zjistíme prvek  
7         head--; // posuneme hlavu  
8         a = value;  
9         return true;  
10    }  
11    a = // tady vznikne problem  
12    return false;
```

Generické typy

- Vznikla potřeba přiřadit nějakou výchozí hodnotu - klíčové slovo default:

```
1 public bool Pop(out T a) {  
2     if (!IsEmpty()) {  
3         T value = data[head]; // zjistíme prvek  
4         head--; // posuneme hlavu  
5         a = value;  
6         return true;  
7     }  
8     a = default // problem solved!  
9     return false;  
10 }
```

Omezení generické typu

- Při deklaraci generické třídy/rozhraní můžeme omezit s jakými typy můžeme pracovat
 - ▶ Např. když potřebujeme použít nějakou metodu určitých typů
- Omezení pomocí `where`, př.: `public class Stack<T> where T : Person`

Formulace	Význam
<code>where T: struct</code>	T musí být hodnotový datový typ
<code>where T: class</code>	T musí být referenční datový typ
<code>where T: IBlah</code>	Typ T musí implementovat rozhraní IBlah
<code>where T: Foo</code>	Typ T musí být potomkem třídy Foo
<code>where T: new()</code>	Typ T musí mít výchozí konstruktor (bez parametrů)
<code>where T1: T2</code>	Typ T1 je potomkem typu T2

Nullable hodnotové datové typy

- Často by se nám mohlo hodit umět dát `null` např. do proměnné typu `int`
- Ke každému hodnotovému typu je k dispozici jeho nullable varianta (s otazníkem)
 - ▶ `bool?`, `int?`, `double?`, ...
- Každý nullable typ je instancí generického typu `System.Nullable<T>`, vlastnosti:
 - ▶ `bool HasValue;` – predikát, jestli obsahuje hodnotu
 - ▶ `T Value` – přístup k hodnotě

```
1 int a = 32;
2 int? b = a;
3 b = null;
4
5 a = b; // Co se stane?
```

Nullable hodnotové datové typy

- Často by se nám mohlo hodit umět dát `null` např. do proměnné typu `int`
- Ke každému hodnotovému typu je k dispozici jeho nullable varianta (s otazníkem)
 - ▶ `bool?`, `int?`, `double?`, ...
- Každý nullable typ je instancí generického typu `System.Nullable<T>`, vlastnosti:
 - ▶ `bool HasValue`; – predikát, jestli obsahuje hodnotu
 - ▶ `T Value` – přístup k hodnotě

```
1 int a = 32;
2 int? b = a;
3 b = null;
4
5 a = (int) b; // Co se stane teď?
```

Nullable hodnotové datové typy

- Musíme vždy kontrolovat, zda v proměnné není `null`

```
1 if (b.HasValue) { // nebo b!=null
2     a = (int) b;
3 }
```

- Nebo použít metodu `GetValueOrDefault()`, která případně vrátí výchozí hodnotu

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/default-values>

- Rozepisování je zdlouhavé $\Rightarrow$ nový operátor

- ▶ `??` – null coalescing operator: pokud levá strana není null, vrátí ji. Jinak vrátí pravou

```
a = b ?? -1;
```

Systémové kolekce

- Většinu datových struktur máme v C# již hotovou – nemusíme je programovat
- Seznam, spojový seznam, seřazený seznam, fronta, zásobník, množiny, slovník, ...
- `IEnumerable` – pouze pro procházení (nutné pro `foreach`)
- `ICollection` – zjištění počtu prvků, kopie do pole
- `IList` – K prvkům lze přistupovat na základě pořadí, mají indexer, přidání, odebrání, vymazání
- Další viz `System.Collections` a `System.Collections.Generic`

Enumerátory

- Pomocné třídy, které nám umožní iteraci přes objekt za pomoci `foreach`
- Rozhraní `IEnumerable` má jedinou metodu pro vrácení enumerátoru `public IEnumerator GetEnumerator()`
<https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable.getenumerator?view=net-5.0>
- Enumerátor je jen struktura, pamatující si, na kterém místě zrovna jsme
- Ukázka – iterovatelný zásobník

Úkol

- 1 Předělejte vaši množinu tak, aby do ní bylo možné vkládat jakékoliv objekty (stejného typu - genericky)
- 2 Předělejte i vaše rozhraní, aby bylo generické
- 3 Dodejte vašemu rozhraní předka `IEnumerable<T>`
- 4 Implementujte potřebné metody a enumerátor
- 5 Ověřte, zda vám na vaší množině funguje `foreach` a zda je možné množinu použít i pro jiné typy než `int`