Jazyk C# 1

5. seminář

Jakub Večeřa

Univerzita Palackého v Olomouci

25. 10. 2024

Reakce na úkoly

- Složitost interní reprezentace
- Hlídání si velikosti
- Duplicity v množině
- `AreEqual` – využití IsSubsetOf, správnost
- Komplikované konstrukty
- Testování

Rozhraní

- formální popis toho, jaké má objekt veřejné metody $\Rightarrow$ co s ním můžeme dělat
- V praxi nás často nezajímá s kterými objekty pracujeme, ale co s nimi můžeme dělat
- Definice rozhraní:

```
1 public interface IStorageDevice {
2    uint GetFreeSpace();
3    bool WriteNumber(int number, uint position);
4    int ReadNumber(uint position);
5 }
```

- Deklarace, že daná třída implementuje rozhraní

```
public class HardDiskDrive : IStorageDevice { ... }
```

- Konvence - názvy rozhraní začínají `I`

Zpět k zásobníku

- Jak by mohla vypadat formální definice rozhraní našeho zásobníku?

Zpět k zásobníku

- Jak by mohla vypadat formální definice rozhraní našeho zásobníku?

```
1 public interface IStack
2    {
3        public int Size();
4        public int Pop();
5        public int Peek();
6        public bool Push(int number);
7        public void Print();
8    }
```

- Poté již stačí deklarovat, že naše implementace splňuje dané rozhraní

```
public class Stack : IStack
```

Referenční a hodnotové datové typy

- Hodnotové typy – hodnota uložena přímo na zásobníku
  - atomické typy – int, bool, float, double,...
  - životnost pouze v aktuálním prostředí
  - jako parametr funkce je kopie hodnoty
- Referenční typy – na zásobníku uložena pouze adresa objektu na haldě
  - paměť na haldě přidělena až po použití operátoru `new`, jinak `null`
  - životnost určuje garbage collector
  - jako parametr funkce je kopie adresy

Referenční a hodnotové datové typy

- Hodnotové typy – hodnota uložena přímo na zásobníku
  - atomické typy – int, bool, float, double,...
  - životnost pouze v aktuálním prostředí
  - jako parametr funkce je kopie hodnoty
- Referenční typy – na zásobníku uložena pouze adresa objektu na haldě
  - paměť na haldě přidělena až po použití operátoru `new`, jinak `null`
  - životnost určuje garbage collector
  - jako parametr funkce je kopie adresy
- Klíčové slovo `ref` – předání parametru pomocí reference(odkazu)
  - Proměnná musí být inicializována, obousměrnost
  - Metoda jej nemusí nastavovat vůbec
- Klíčové slovo `out` – opět předání odkazu
  - Nemusí být inicializována, jednosměrnost (více návratových hodnot)
  - Volaná metoda musí nastavit

Dědičnost

- Co znáte o dědičnosti z Paradigmat?

Dědičnost v C#

- Pouze jednoduchá dědičnost (vs. více rozhraní)
- Každý objekt implicitně dědí z `System.Object`, jeho metody:
  - `public virtual string ToString()`
  - `public virtual int GetHashCode()`
  - `public virtual bool Equals(object other)`
  - `public Type GetType()`
  - `protected object MemberwiseClone()`
  - `public static bool ReferenceEquals(object objA, object objB)`
- Dědičnost je možná i mezi rozhraními

Dědičnost

- Vytvoříme si třídu `Person` reprezentující nějakou osobu

```
1   public class Person
2    {
3       public string Name;
4       public string Surname;
5       private int Id;
6       public Address address;
7
8       public Person(string name, string surname, Address a)
9       {
10          Name = name;
11          Surname = surname;
12          address = a;
13      }
14  }
```

- Ukázka překrytí metod ToString()

Dědičnost - deklarace předka

- Fakt, že třída dědí z nadtřídy specifikujeme podobně jako rozhraní

```
public class Employee : Person
```

- Takto vytvořená třída bude mít přístup ke všem slotům a metodám
- Ukázka funkčnosti metody ToString()
- Ukázka zděděných metod
- Třída může dědit z jedné třídy, ale splňovat více rozhraní (dědičnost vždy první)
- Překrývání metod - předek určuje klíčovým slovem `virtual`, které metody je možné překrýt

Modifikátory přístupu

- Dědičnost nám zkomplikuje modifikátory přístupů ke slotům a metodám
  - `public` – přístup je odkudkoliv z kódu
  - `private` – přístup je pouze z dané třídy
  - `protected` – přístup z dané třídy a jejích potomků
- Navíc modifikátory na základě sestavení
  - `internal` – přístup odkudkoliv ze stejného sestavení
  - `protected internal` – přístup ze stejného sestavení nebo z potomků
  - `private protected` – přístup z dané třídy a potomků ze stejného sestavení

Zjištění typu

- Velmi často budeme potřebovat znát, jakého typu je daná instance
- Operátor `is`:

```
if (e is Person) { }
```

- Stejně funguje dotaz na splnění rozhraní

```
zasobnik is IStack
```

Úkol

1. Formálně definujte rozhraní `IIntSet`, které by měla splňovat vaše množina z minulého úkolu
2. Deklarujte, že jej vaše množina implementuje
3. Implementujte třídu reprezentující prioritní frontu danou rozhraním:

```
1 public interface IIntPriorityQueue {
2   bool IsEmpty();
3   void InsertWithPriority(int value, int priority);
4   bool PopNextValue(out int val);
5   bool PeekNextValue(out int val);
6 }
```

4. Vhodně přepište metodu `ToString()` (v implementaci prioritní fronty)