Programowanie 3 - zaawansowane

Jan Bródka

Wydział Matematyki i Nauk Informacyjnych

Laboratorium - Kolekcje standardowe

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”
współfinansowany jest ze środków Unii Europejskiej
w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu
„NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”,
współfinansowanego ze środków Unii Europejskiej
w ramach Europejskiego Funduszu Społecznego.

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Kolekcje standardowe biblioteki .NET

- kolekcje uogólnione (bezpośredni odpowiednik kontenerów z STL-a, dostępne od wersji C# 2.0) przechowujące elementy wskazanego typu zdefiniowane są w przestrzeni nazw System.Collection.Generic

- kolekcje o elementach typu object (dostępne od wersji C# 1.0) zdefiniowane są w przestrzeni nazw System.Collection

- zalety kolekcji z System.Collections.Generic
  - lepsza kontrola typów w programie
  - lepsza wydajność dla typów bezpośrednich
    - nie ma operacji pakowania niezbędnej dla "zwykłych" kolekcji o elementach typu object
  - większa przejrzystość kodu (nie trzeba ciągle rzutować)

- wniosek: **lepiej używać kolekcji uogólnionych niż kolekcji "starego typu"**

Standardowe interfejsy kolekcji

- oprócz klas kolekcji w wymienionych przestrzeniach nazw zdefiniowane są interfejsy (uogólnione i "zwykłe") implementowane przez kolekcje
  - IEnumerable<T>
  - ICollection<T> - implementowany przez wszystkie kolekcje standardowe, jest interfejsem pochodnym od IEnumerable<T>
  - interfejsy pochodne od ICollection<T>
    - IList<T> - kolekcje z dostępem przez indeksowanie
    - IDictionary<K,V> - kolekcje przechowujące pary klucz/wartość
    - ISet<T> - abstrakcja pojęcia zbioru (od C# 4.0)

- dzięki tym interfejsom wymuszona jest podobna wspólna funkcjonalność wielu kolekcji (w STL-u z C++ wspólna funkcjonalność to tylko umowa, język jej nie wymusza)

- zaleca się do wymiany informacji między obiektami (np. w parametrach metod) używać interfejsów, a nie konkretnych kolekcji

Interfejsy IEnumerable<T> i IEnumerator<T>

- interfejs IEnumerable<T>
  - dziedziczy po IEnumerable
  - definiuje metodę IEnumerator<T> GetEnumerator();
  - klasa implementująca musi jawnie implementować metodę
    IEnumerator IEnumerable.GetEnumerator();
- interfejs IEnumerator<T>
  - dziedziczy po IEnumerator i IDisposable (!)
  - definiuje jedynie właściwość
    T Current { get; }
  - klasa implementująca musi jawnie implementować właściwość
    object IEnumerator.Current { get; }

- taka konstrukcja zapewnia że każdy typ implementujący uogólnione wersje tych interfejsów implementuje również wersje "zwykłe"
- uogólniony interfejs IEnumerable<T> można również implementować za pomocą iteratorów yield ("zwykły" oczywiście też)

Interfejs ICollection<T>

- uogólniony interfejs ICollection<T> zawiera
  - Count - właściwość (do odczytu) - liczba elementów
  - IsReadOnly - właściwość (do odczytu) - czy kolekcję można modyfikować
  - void Add(T e) - wstawienie elementu
  - bool Remove(T e) - usunięcie elementu
  - void Clear() - usunięcie wszystkich elementów
  - bool Contains(T e) - czy kolekcja zawiera wskazany element
  - void CopyTo(T[] tab, int k) - kopiowanie wszystkich elementów kolekcji do tablicy tab, począwszy od indeksu k (do tab[k] będzie skopiowany pierwszy element kolekcji)

- "zwykły" interfejs ICollection zawiera jedynie właściwość Count i metodę CopyTo oraz właściwości związane z aplikacjami wielowątkowymi
- interfejs uogólniony nie dziedziczy po "zwykłym"
- dla kolekcji tylko do odczytu metody Add, Remove i Clear powinny zgłaszać wyjątek NotSupportedException

Interfejs IList<T>

- uogólniony interfejs IList<T> zawiera
  - indeksator z parametrem int (odczyt/zapis)
  - void Insert(int k, T e) - wstawienie elementu e na pozycji k-tej (kolejne elementy są przesuwane do tyłu)
  - void RemoveAt(int k) - usunięcie k-tego elementu (i przesuniecie kolejnych)
  - int IndexOf(T e) - wyszukiwanie elementu e, zwraca indeks (-1 gdy nie znaleziono)

- "zwykły" interfejs IList zawiera te same składowe i dodatkowo metody Add, Remove, Clear, Contains ("brakujące" w zwykłym ICollection)

- Klasa Array (czyli zwykłe **tablice**) implementuje "zwykły" interfejs IList, oczywiście metody zmieniające liczbę elementów generują wyjątek

Interfejs IDictionary<K,V>

- kolekcje implementujące uogólniony interfejs IDictionary<K,V> przechowują pary klucz/wartość (typu odpowiednio K i V)
- uogólniony interfejs IDictionary<K,V> zawiera (między innymi)
  - indeksator z parametrem typu K (klucz) - umożliwia odczyt/zapis wartości
  - metody dodawania/usuwania operujące na parach klucz/wartość
- "zwykły" interfejs IDictionary zawiera analogiczne składowe i dodatkowo metody Add, Remove, Clear, Contains ("brakujące" w zwykłym ICollection)
- kolekcje implementujące IDictionary są odpowiednikami map z biblioteki STL (z unikalnymi kluczami, wartości mogą się powtarzać)

Interfejs ISet<T>

- został wprowadzony w C# 4.0
- implementuje matematyczne pojecie zbioru (bez powtórzeń)
- dostarcza metod realizujących operacje mnogościowe (argument jest typu IEnumerable<T>, metody modyfikują bieżący zbiór)
  - UnionWith - suma zbiorów
  - IntersectWith - iloczyn zbiorów
  - ExceptWith - różnica zbiorów
  - SymmetricExceptWith - różnica symetryczna zbiorów
  - Overlaps - true jeśli zbiory mają wspólny element
  - SetEquals - true jeśli zbiory są równe (mają te same elementy)
  - IsSubset, IsSuperset - true jeśli bieżący zbiór jest podzbiorem (nadzbiorem) zbioru argumentu
  - Add(T) - wstawia element do zbioru, zwraca true/false (zbiór jest bez powtórzeń!)
  - ... i wiele innych

Kolekcje standardowe - przegląd

kolekcje uogólnione	**kolekcje "zwykłe"**	**interfejs**	**Kontener z STL-a**
List<T>	ArrayList	IList<T>	vector<T>
Dictionary<K,V>	HashTable	IDictionary<K,V>	---
SortedList<K,V>	SortedList	IDictionary<K,V>	---
SortedDictionary<K,V>	---	IDictionary<K,V>	map<K,V>
HashSet<T> (C# 3.0)	---	ISet<T>	---
SortedSet<T> (C# 4.0)	---	ISet<T>	set<T>
LinkedList<T>	---	ICollection<T>	list<T>
Queue<T>	Queue	ICollection<T>	queue<T>
Stack<T>	Stack	ICollection<T>	stack<T>
---	BitArray	ICollection	bitset

Kolekcja List<T>

- reprezentuje rozszerzalne tablice (o zmiennej liczbie elementów)
- najważniejsze metody (oprócz wymaganych przez interfejsy)
  - Sort - sortowanie (algorytmem quicksort) - kilka wersji (z domyślnym lub wskazanym przez parametr kryterium sortowania)
  - BinarySearch - wyszukiwanie binarne - zwraca indeks elementu, kolekcja musi być posortowana
  - ForEach - parametrem jest delegacja opisująca czynności do wykonania na każdym elemencie kolekcji
  - Remove - usuwa pierwsze wystąpienie elementu o wskazanej wartości
  - RemoveAt - usuwa element o wskazanym indeksie
  - AddRange, InsertRange - wstawia na końcu/we wskazanym miejscu elementy innej kolekcji (podanej jako parametr)
  - grupa metod Find... - wyszukiwanie elementów spełniających podany predykat
  - ConvertAll<S> - tworzy nową listę powstałą z listy źródłowej w wyniku konwersji każdego z jej elementów z typu T do S (metoda uogólniona)

Kolekcje przechowujące pary klucz/wartość

- Dictionary<K,V>
  - implementowana przy pomocy tablic haszowanych
  - dla przechowywanych obiektów musi być szczególnie starannie zdefiniowana metoda GetHashCode
  - elementy kolekcji nie są posortowane
- SortedDictionary<K,V>
  - implementowana przy pomocy drzew zrównoważonych
  - elementy kolekcji są posortowane
- SortedList<K,V>
  - implementacja tablicowa (analogicznie do kolekcji List<T>)
  - elementy kolekcji są posortowane
- wszystkie mają podobny (ale nie identyczny) zestaw metod (niewiele rozszerzający wymagania interfejsu IDictionary)
- różnią się wydajnością (złożonością obliczeniową) metod
- formalnie zawierają elementy typu KeyValuePair<K,V> (odpowiednik struktury pair<K,V> z STL)

Kolekcja LinkedList<T>

- jest zaimplementowana jako lista wiązana (podwójnie wiązana) składająca się z oddzielnych węzłów (typu LinkedListNode<T>)
  - każdy z węzłów zawiera właściwości
    - Next, Previous - (tylko odczyt) - poruszanie się po liście
    - Value - (odczyt/zapis) - dostęp do wartości elementu
  - wstawianie/usuwanie elementów jest szybkie (czas stały), przetwarzanie sekwencyjne jest wolniejsze

- właściwości First, Last - (odczyt) - zwracają pierwszy/ostatni węzeł listy
- metody
  - AddAfter/AddBefore - dodawanie po/przed wskazanym węzłem
  - AddFirst/AddLast - dodawanie na początku/na końcu listy
  - Remove - dwie wersje przeciążone - usuwanie wskazanego węzła lub pierwszego wystąpienia podanej wartości
  - RemoveFirst/RemoveLast - usuwanie pierwszego/ostatniego elementu listy
  - Find/FindLast - wyszukiwanie węzła zawierającego wskazaną wartość

Kolekcje HashSet<T> i SortedSet<T>

- HashSet<T> została wprowadzona w C# 3.0, a SortedSet<T> w C#4.0

- obie kolekcje dostarczają metod
  - RemoveWhere - usuwa ze zbioru elementy spełniające wskazany predykat
  - oraz metod wymaganych przez interfejsy ISet<T> i ICollection<T>

- SortedSet<T> dodatkowo dostarcza metod i właściwości
  - GetViewBetween - zwraca SortedSet zawierający elementy pomiędzy wskazanymi elementami
  - Reverse - iteruje (zwraca IEnumerable<T>) po zbiorze w kolejności odwrotnej
  - Max, Min - właściwości (do odczytu) zwracające odpowiednio największy/najmniejszy element zbioru

Kolekcje Stack<T> i Queue<T>

- Stack<T> - stos
  - void Push(T e) - wstawia element e na szczyt stosu
  - T Pop() - zdejmuje (pobiera) element ze szczytu stosu
  - T Peek() - "podgląda" element na szczycie stosu
- Queue<T> - kolejka
  - void Enqueue(T e) - wstawia element e na koniec kolejki
  - T Dequeue() - pobiera (i usuwa z kolejki) pierwszy element
  - T Peek() - "podgląda" pierwszy element

- kolekcje te są zaimplementowane jako rozszerzalne tablice
  - jeśli jest jeszcze miejsce na nowy element, to wstawianie jest szybkie
  - w przeciwnym przypadku następuje realokacja elementów
- obie kolekcje implementują "zwykły" interfejs ICollection, ale nie implementują ICollection<T>
- obie kolekcje implementują zarówno interfejs IEnumerable jak i IEnumerable<T>

Kolekcja BitArray

- reprezentuje tablicę bitów
- jest zdefiniowana w przestrzeni nazw System.Collection i nie ma odpowiednika uogólnionego w przestrzeni System.Collection.Generic
- zajmuje znacznie mniej pamięci niż kolekcje o elementach typu bool

- liczne konstruktory (z tablicy bool[], z tablicy byte[], z tablicy int[], wskazanego rozmiaru - inicjowana wartościami false, i inne)
- wybrane metody
  - And, Or, Xor - wykonują odpowiednią operację według schematu BitArray *oper* (arg BitArray) - bieżący obiekt jest modyfikowany
  - BitArray Not() - neguje bity
  - void SetAll(bool b) - ustawia wszystkie elementy
- wybrane właściwości
  - Length - liczba elementów kolekcji - do odczytu i zapisu (zapis zmienia rozmiar kolekcji - złożoność liniowa)
  - Count - to samo, co Length, ale tylko do odczytu

Kolekcje standardowe - inicjalizacja (1)

- w C# 3.0 wprowadzony został uproszczony sposób inicjalizowania kolekcji standardowych, a ściślej obiektów dowolnych typów implementujących interfejs uogólniony ICollection<T> lub "zwykły" IList
- deklaracja z inicjalizacją

```
Kolekcja kol = new Kolekcja<T> {el_1, el_2,..., el_n};
```

jest równoważna sekwencji instrukcji

```
Kolekcja kol = new Kolekcja<T>();
kol.Add(el_1);
kol.Add(el_2);
...
kol.Add(el_n);
```

- możliwa jest też oddzielna deklaracja zmiennej, a później jej inicjalizacja w opisany wyżej sposób (a nawet analogiczna konstrukcja bez jawnej deklaracji zmiennej, np. przy przekazywaniu kolekcji jako parametr)

Kolekcje standardowe - inicjalizacja (2)

- w przypadku kolekcji standardowych przechowujących pary klucz/wartość, w szczególności implementujących interfejs uogólniony IDictionary<K,V>, możliwy jest również następujący sposób inicjalizacji
- deklaracja z inicjalizacją

```
Kolekcja kol = new Kolekcja<K,V>
   { {key1,val1}, {key2,val2}, ... , {keyn,valn} };
```

jest równoważna sekwencji instrukcji

```
Kolekcja kol = new Kolekcja<K,V>();
kol.Add(key1,val1);
kol.Add(key2,val2);
...
kol.Add(keyn,valn);
```

Kolekcje standardowe - inicjalizacja (3)

- w C# 6.0 wprowadzony został nowy uproszczony sposób inicjalizowania kolekcji standardowych przechowujących pary klucz/wartość (ściślej wszystkich kolekcji definiujących indeksator, w szczególności implementujących interfejs uogólniony IDictionary<K,V>)
- deklaracja z inicjalizacją

```
Kolekcja kol = new Kolekcja<K,V>
   { [ind1]=wart1, [ind2]=wart2, ... , [indn]=wartn };
```

jest równoważna sekwencji instrukcji

```
Kolekcja kol = new Kolekcja<K,V>();
kol[ind1]=wart1;
kol[ind2]=wart2;
...
kol[indn]=wartn;
```

- nowa wersja różni się od poprzedniej wykorzystaniem indeksatora zamiast metody Add

Kolekcje standardowe - metody rozszerzające

- w C# 3.0 (ściślej w .NET 3.5) wprowadzono liczne metody rozszerzające kolekcje standardowe
- są one zdefiniowane w klasie Enumerable z przestrzeni nazw System.Linq
- rozszerzają interfejs IEnumerable<T>, czyli mogą być stosowane do wszystkich kolekcji standardowych

- niektóre ("bardzo niektóre") z tych metod to:
  - First/Last - zwraca pierwszy/ostatni element kolekcji
  - Max/Min - zwraca maksymalny/minimalny element kolekcji
  - operacje mnogościowe na dwóch kolekcjach (Union, Intersect, Except)
  - konwersje na konkretne typy kolekcji (ToArray, ToList, ToDictionary)
  - liczne metody związane bezpośrednio z technologią LINQ (np. Select, Where, Join, GroupBy)

- wiele z tych metod jako dodatkowy argument może przyjmować delegację (w praktyce zwykle wyrażenie lambda)

Kolekcje standardowe - uwagi

- wszystkie kolekcje mają konstruktory pozwalające utworzyć daną kolekcję na podstawie innej kolekcji (w tym również całkiem innego typu - mają parametr typu ICollection lub IDictionary)
- podobnie większość kolekcji (oprócz opartych na węzłach) ma konstruktory określające początkową pojemność (chociaż tworzą puste kolekcje)
- kolekcje Sorted... mają dodatkowo konstruktory określające kryterium sortowania
- w przestrzeni nazw System.Collection.ObjectModel jest zdefiniowana kolekcja Collection<T> pomyślana specjalnie jako klasa bazowa dla własnych kolekcji (jest oparta na kolekcji List<T>)

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”
współfinansowany jest ze środków Unii Europejskiej
w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu
„NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”,
współfinansowanego ze środków Unii Europejskiej
w ramach Europejskiego Funduszu Społecznego.

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny