Colecții generice

Introducere

Clasele spațiului de nume System.Collections.Generic

Structurile spațiului de nume System.Collections.Generic

Interfețele spațiului de nume System.Collections.Generic

Clasa List<T>

Clasa Dictionary<TKey, TValue>

Colecții sortate

Cozi și stive

Introducere

-Datele având aceleași caracteristici pot fi tratate mai eficient atunci când grupate într-o colecție. În loc de a scrie cod care să trateze separat fiecare obiect în parte, prin intermediul colecțiilor puteți utiliza același cod pentru a procesa toate elementele care au caracteristici comune.

-Pentru a gestiona o colecție cu un număr fix de elemente se poate utiliza clasa System.Array. Pentru a adăuga, elimina și modifica fie elemente individuale sau o serie de elemente în colecție se pot utiliza clasele din System.Collections, așa cum am procedat în cursurile anterioare, însă și o mulțime de alte clase din spațiul de nume System.Collections.Generic care permit crearea unor colectii puternic tipizate.

-.NET Framework 4 , adauga spațiul de nume System.Collections.Concurrent care furnizează clase "thread-safe", utile atunci când elementele colecției sunt accesate concurent de mai multe fire de execuție.

- Clasele din spațiul de nume System.Collections.Immutable permit lucrul cu colectii care nu pot fi schimbate și care prin modificare creaza noi colecții. Aceste clase au fost adaugate de versiunea .NET 4.5 și vizează aplicații care se adresează desktop-ului, Windows Phone 8, Windows Store.

-Colectiile generice au fost adaugate de versiunea .NET 2.0. In acest sens, a fost adăugat spatiul de nume System.Collections.Generic, care conține mai multe clase colectii care utilizeaza genericele.

-Genericele permit crearea unei colectii “type-safe” la momentul compilarii. Daca o colectie ArrayList poate pastra obiecte de orice tip, in schimb intr-o colectie de tipul List<T> se pot pastra doar obiecte de tipul T.

- Orice tip referință sau tip valoric, care se adaugă la un ArrayList este implicit transformat intr-un obiect de tipul Object. În cazul în care elementele sunt tipuri valorice, acestea sunt impachetate atunci când sunt adăugate in lista, și despachetate când sunt preluate. Operatiile de impachetare si despachetare scad performanta, iar efectul poate fi foarte important în scenarii care utilizeaza colecții mari.

-Din motivele mentionate mai sus, pentru crearea unor tipuri sigure, se recomanda utilizarea colectiilor generice in locul colectiilor non-generice.

Clasele spațiului de nume System.Collections.Generic

Clasa	Descriere
Comparer <T>	Clasă care oferă implementări ale interfeței generice IComparer <T> .
Dictionary<TKey, TValue>	Reprezintă o colecție de chei și valori.
Dictionary<TKey, TValue>.KeyCollection	Reprezintă colecția de chei într-un Dictionary<TKey,TValue>. Aceasta clasa nu poate fi moștenită.
Dictionary <TKey, Tvalue>.ValueCollection	Reprezintă colecția de valori într-un Dictionary<TKey,TValue> . Aceasta clasa nu poate fi moștenită .
EqualityComparer <T>	Clasa care oferă implementări ale interfeței generice IEqualityComparer <T> .
HashSet <T>	Reprezintă un set de valori.
KeyedByTypeCollection <TItem>	Oferă o colecție ale cărei elemente sunt tipuri care servesc drept chei.
KeyNotFoundException	Excepția este lansată , atunci când cheia specificată pentru accesarea unui element într-o colecție nu se potrivește cu nici o cheie din colecție.
LinkedList <T>	Reprezintă o listă dublu legată.
LinkedListNode <T>	Reprezintă un nod în LinkedList <T> . Această clasă nu poate fi moștenită.
List <T>	Reprezintă o listă puternic tipizată de obiecte care pot fi accesate de un index. Oferă metode de căutare, sortare și manipulare a listelor
Queue<T>	Reprezintă o colecție first-in, first-out (FILO) de obiecte.

SortedDictionary <TKey, TValue>	Reprezintă o colecție de perechi cheie / valoare care sunt sortate pe baza cheii.
SortedDictionary <TKey, TValue> . KeyCollection	Reprezintă colecția de chei într-un SortedDictionary <TKey, TValue> . Această clasă nu poate fi moștenită.
SortedDictionary <TKey, TValue> . ValueCollection	Reprezintă colecția de valori într-un SortedDictionary <TKey, TValue> . Această clasă nu poate fi moștenită.
SortedList <TKey, TValue>	Reprezintă o colecție de perechi cheie / valoare care sunt sortate prin punerea în aplicare a interfeței IComparer <T> .
SortedSet <T>	Reprezintă o colecție de obiecte care se menține în ordine sortată.
Stack <T>	Reprezintă o colecție last-in first-out (LIFO) de obiecte de același tip.
SynchronizedCollection <T>	Oferă o colecție thread-safe, care conține obiecte de tipul specificat de parametrul generic.
SynchronizedKeyedCollection <K, T>	Oferă o colecție thread-safe, care conține obiecte de tipul specificat de parametrul generic și care sunt grupate pe chei.
SynchronizedReadOnlyCollection <T>	Oferă o colecție thread-safe read-only , care conține obiecte de tipul specificat de parametrul generic.

Structurile spațiului de nume System.Collections.Generic

- Cu exceptia structurii:

KeyValuePair <TKey, TValue> *Definește o pereche cheie/valoare care poate fi setată sau recuperată*,

restul structurilor sunt enumeratori ai claselor spatiului de nume System.Collections.Generic.

- Fiecare din structurile următoare enumeră elementele din clasa corespunzătoare. Spre exemplu, Dictionary <TKey, TValue> .Enumerator enumeră elementele unui Dictionary <TKey, TValue>. (Metoda GetEnumerator din acesta clasa returneaza o structura Dictionary <TKey, TValue> .Enumerator).

Celelalte structuri ale spațiului de nume System.Collections.Generic sunt:

Dictionary <TKey, TValue>. KeyCollection.Enumerator ;
Dictionary <TKey, TValue>. ValueCollection.Enumerator;
HashSet <T>.Enumerator ;
LinkedList <T> .Enumerator ;
List <T> .Enumerator;
Queue <T> .Enumerator;
SortedDictionary <TKey, TValue> . Enumerator;
SortedDictionary <TKey, TValue> . KeyCollection.Enumerator ;
SortedDictionary <TKey, TValue> . ValueCollection.Enumerator;
SortedSet <T> .Enumerator;
Stack <T> . Enumerator;

Interfețele spațiului de nume System.Collections.Generic

Interfața	Descriere
ICollection <T>	Definește metode pentru a manipula colecții generice (Add, Clear, Contains, CopyTo, Remove).
IComparer <T>	Definește o metodă (Compare) pe care un tip trebuie să o implementeze pentru a compara două obiecte de tipul T.
IDictionary <TKey, TValue>	Reprezintă o colecție generică de perechi cheie / valoare.
IEnumerable <T>	Expune enumaratorul, care realizează o iterare simplă a unei colecții de un tip specificat .
IEnumerator <T>	Realizează o iterare simplă a unei colecții generice.
IEqualityComparer <T>	Definește metode pentru compararea obiectelor.
IList <T>	Reprezintă o colecție de obiecte care pot fi accesate individual printr-un index .
IReadOnlyCollection <T>	Reprezintă o colecție read-only puternic tipizată de elemente.
IReadOnlyDictionary <TKey, TValue>	Reprezintă o colecție read-only de perechi cheie/valoare.
IReadOnlyList <T>	Reprezintă o colecție read-only de elemente care pot fi accesate printr-un index .

Clasa List<T>

Reprezintă o listă puternic tipizată de obiecte care pot fi accesate prin intermediul unui index. Clasa furnizează metode pentru căutare, sortare și manipularea unei liste.

Proprietățile clasei List<T>

Capacity	Obține sau setează numărul total de elemente pe care structura internă de date le poate stoca fără redimensionare.
Count	Obține numărul de elemente cuprinse în List <T>.
this[int]	Obține sau setează elementul la indexul specificat (indexarea clasei List<T>).

Câteva metode ale clasei List <T>

Add	Adaugă un obiect la sfârșitul listei List <T>.
AddRange	Adaugă elementele colecției specificate la sfârșitul listei List<T>.
Clear	Elimină toate elementele din List <T>.
Contains	Determină dacă un element este în List <T>.
ConvertAll <TOutput>	Convertește elementele din lista curentă List <T> la un alt tip (tipul TOutput), și returnează o listă care conține elementele convertite.
CopyTo (T [])	Copie întreaga listă List <T> într-un tablou (Array) compatibil unidimensional. Punctul de start este la începutul matricei țintă.
GetEnumerator	Returnează un enumerator (structura List<T>.Enumerator) care iterează lista List <T> . (Metoda MoveNext() din structura List<T>.Enumerator este apelata automat de bucla foreach)
GetRange	Crează o copie superficială a unor elemente din List <T> .
IndexOf (T)	Caută obiectul specificat și întoarce prima apariție (indexarea se face de la zero) în întreaga listă List <T> .

Insert(int, T)	Inserează un element în List <T> la indexul specificat .
InsertRange	Inserează elementele unei colecții în lista List <T> la indexul specificat .
LastIndexOf (T)	Caută obiectul specificat și întoarce indexul ultimei apariții în întreaga listă List<T> .
Remove(T)	Sterge obiectul specificat la prima apariție a sa în List <T> .
RemoveAt(int)	Elimină elementul avand indexul specificat de List <T> .
Reverse ( )	Inversează ordinea elementelor din întreaga listă List <T> .
Sort ( )	Sortează elementele din întreaga listă List <T> folosind comparatorul implicit.
Sort (IComparer <T>)	Sortează elementele din întreaga listă List<T> folosind comparatorul specificat .

Următoarele două programe demonstrează modul în care se poate utiliza clasa List <T> împreună cu proprietățile și metodele sale, însă și alte aspecte generice: clasa Comparer <T>, interfețele IEquatable<T>, IComparable<T> și IComparer<T>.

-Remarcați în primul program cum se pot utiliza metodele Contains(T) și Remove(T) (ICollection<T>) implementate de List <T> în conjuncție cu interfața IEquatable<T>.

-În cel de-al doilea program, remarcați cum se pot utiliza IComparable<T> și IComparer<T> pentru a sorta o colecție. Comparați programul de față cu un program similar prezentat în cursul 9.

Exemplu List<T>

```
using System;
using System.Collections.Generic;

public class Produs : IEquatable<Produs>   {
    public int produs_ID;
    public string numeProdus;

    public Produs(int produs_ID, string numeProdus)  {
        this.produs_ID = produs_ID;
        this.numeProdus = numeProdus;
    }

    public override string ToString()   {
        return "Produs_ID: " + produs_ID + "   Nume: " + numeProdus;
    }

    public bool Equals(Produs altProdus)  {
        if (altProdus == null) return false;
        return (this.produs_ID.Equals(altProdus.produs_ID));
    }
}
```

```
class Exemplu {
    public static void Main() {
        List<Produs> listaProduse = new List<Produs>();

        // Adauga produse in lista.
        listaProduse.Add(new Produs(1000, "pix"));
        listaProduse.Add(new Produs(1001, "minge"));
        listaProduse.Add(new Produs(1002, "caiet"));
        listaProduse.Add(new Produs(1003, "calculator"));
        listaProduse.Add(new Produs(1004, "joc"));
        listaProduse.Add(new Produs(1005, "camasa"));

        // Scriem in consola elementele din lista de produse
        Console.WriteLine();
        foreach (Produs unProdus in listaProduse) {
            Console.WriteLine(unProdus);
        }
        // Verificam daca in listaProduse se afla produsul cu ID-ul #1003.
        //Metoda Contains apeleaza metoda IEquitable.Equals din clasa Produse.
        Console.WriteLine("\nLista contine produsul cu ID-ul (\"1003\"): {0}",
        listaProduse.Contains(new Produs(1003, "")));
```

```
// Inseram un item nou la pozitia 2.
        Console.WriteLine("\nInseram(2, \"1015\")");
        listaProduse.Insert(2, new Produs(1015, "masina"));
        foreach (Produs unProdus in listaProduse)  {
            Console.WriteLine(unProdus);
        }
//Accesam un element din colectie printr-un index
        Console.WriteLine("\nlistaProduse[3]: {0}", listaProduse[3]);
        Console.WriteLine("\nStergem(\"1003\")");
  // Stergem produsul cu ID-ul 1003 chiar daca numeProdus este diferit.
  // Acest lucru este posibil deoarece metoda Equals verifica doar ID-ul produselor.
        listaProduse.Remove(new Produs(1003, "fructe"));
        Console.WriteLine();
        foreach (Produs unProdus in listaProduse) {
            Console.WriteLine(unProdus);
        }

        Console.WriteLine("\nRemoveAt(3)");
   // Aceasta va sterge produsul cu indexul 3.
        listaProduse.RemoveAt(3);
        Console.WriteLine();
        foreach (Produs unProdus in listaProduse )  {
            Console.WriteLine(unProdus);
        }
    }
}
```

Rezultat:

Produs_ID: 1000 Nume: pix
Produs_ID: 1001 Nume: minge
Produs_ID: 1002 Nume: caiet
Produs_ID: 1003 Nume: calculator
Produs_ID: 1004 Nume: joc
Produs_ID: 1005 Nume: camasa
Lista contine produsul cu ID-ul ("1003"): True
Inseram(2, "1015")
Produs_ID: 1000 Nume: pix
Produs_ID: 1001 Nume: minge
Produs_ID: 1015 Nume: masina
Produs_ID: 1002 Nume: caiet
Produs_ID: 1003 Nume: calculator
Produs_ID: 1004 Nume: joc
Produs_ID: 1005 Nume: camasa
listaProdus[3]: Produs_ID: 1002 Nume: caiet
Stergem("1003")
Produs_ID: 1000 Nume: pix
Produs_ID: 1001 Nume: minge
Produs_ID: 1015 Nume: masina
Produs_ID: 1002 Nume: caiet
Produs_ID: 1004 Nume: joc
Produs_ID: 1005 Nume: camasa
RemoveAt(3)

Produs_ID: 1000 Nume: pix
Produs_ID: 1001 Nume: minge
Produs_ID: 1015 Nume: masina
Produs_ID: 1004 Nume: joc
Produs_ID: 1005 Nume: camasa

```
//Comparati prezentul program cu un program similar prezentat in cursul 9. Notati diferenta dinte interfetele
//IComparable si IComparer, pe de o parte, si  interfetele generice IComparable<T> si IComparer<T>, pe de alta parte.
//Mai precis, daca metodele CompareTo() si Compare() ale interfetelor IComparable si respectiv IComparer
// au ca parametrii obiecte de tip Object, pentru CompareTo() si Compare(), ale interfetelor IComparable<T> si
//IComparer<T>, parametrii sunt de tipul T. In consecinta, NU trebuie sa ne asiguram, asa cum am facut in programul
//din cursul 9, ca parametrii sunt de tipul dorit. Genericele au asa numita facilitate type-safe.
using System;
using System.Collections.Generic;

public class Produs: IComparable<Produs> {
    public int produs_ID;
    public string numeProdus;

    public Produs(int produs_ID, string numeProdus) {
        this.produs_ID = produs_ID;
        this.numeProdus = numeProdus;
    }
    public override string ToString() {
        return "Produs_ID: " + produs_ID + "   Nume: " + numeProdus;
    }
    //Metoda CompareTo(T) din IComparable<T> are ca parametru formal tipul T.
    //Metoda CompareTo(object) din IComparable are ca parametru formal tipul object.
    public int CompareTo(Produs prod) {
        return this.produs_ID - prod.produs_ID;
    }
}
```

```
public class ComparaNumeProduse : IComparer<Produs> {
   public static IComparer<Produs> Default = new ComparaNumeProduse();

//Metoda Compare(T,T) din IComparable<T> are ca parametrii formali tipul T.
//Metoda CompareTo(object, object) din IComparable are ca parametrii formali tipul object.
   public int Compare(Produs p1, Produs p2) {
//Apelam proprietatea Default a clasei Comparer<string> pentru a returna un obiect al acestei
clase.  Prin intermediul acestui obiect utilizam implementarea standard a metodei
Compare(string,string)
      return Comparer<string>.Default.Compare(p1.numeProdus, p2.numeProdus);
   }
}

class Exemplu{
   public static void Main() {
      List<Produs> listaProduse = new List<Produs>();
// Adaugam produse in lista.
      listaProduse.Add(new Produs(1007, "pix"));
      listaProduse.Add(new Produs(1020, "minge"));
      listaProduse.Add(new Produs(1050, "caiet"));
      listaProduse.Add(new Produs(1003, "calculator"));
      listaProduse.Add(new Produs(1000, "joc"));
      listaProduse.Add(new Produs(1005, "camasa"));
```

```
// Scriem elementele din lista de produse (nesortate)
        Console.WriteLine("Elementele colectiei sunt:");
        foreach (Produs unProdus in listaProduse) {
            Console.WriteLine(unProdus);
        }

//Sortam lista cu comparatorul default (dupa ID).
//Metoda Sort() apeleaza automat CompareTo(T).
//In cazul de fata, Sort() apeleaza CompareTo(Produs) implementata de clasa Produs.
        Console.WriteLine();
        Console.WriteLine("Produse sortate cu comparatorul default (dupa ID):");
        listaProduse.Sort();
// Scriem elementele din lista de produse, sortate dupa ID.
        foreach (Produs unProdus in listaProduse) {
            Console.WriteLine(unProdus);
        }
```

```
//Sortam lista (dupa nume) prin intermediul metodei Sort(IComparer<T>). Aceasta apeleaza
//automat metoda Compare(T,T).
//In cazul de fata, Sort(IComparer<Produs>) apeleaza Compare(Produs, Produs)
//implementata de clasa ComparaNumeProduse.
//Aceasta clasa, creaza un obiect care este referit prin intermediul referintei Default de tip
//interfata IComparer<Produs>.

        Console.WriteLine();
        Console.WriteLine("Produse sortate cu comparatorul non-default (dupa nume):");
        listaProduse.Sort(ComparaNumeProduse.Default);
        // Scriem elementele din lista de produse, sortate dupa nume.
        foreach (Produs unProdus in listaProduse) {
            Console.WriteLine(unProdus);
        }

    }
}
```

Rezultat:
Elementele colectiei sunt:
Produs_ID: 1007 Nume: pix
Produs_ID: 1020 Nume: minge
Produs_ID: 1050 Nume: caiet
Produs_ID: 1003 Nume: calculator
Produs_ID: 1000 Nume: joc
Produs_ID: 1005 Nume: camasa

Produse sortate cu comparatorul default (dupa ID):
Produs_ID: 1000 Nume: joc
Produs_ID: 1003 Nume: calculator
Produs_ID: 1005 Nume: camasa
Produs_ID: 1007 Nume: pix
Produs_ID: 1020 Nume: minge
Produs_ID: 1050 Nume: caiet

Produse sortate cu comparatorul non-default (dupa nume):
Produs_ID: 1050 Nume: caiet
Produs_ID: 1003 Nume: calculator
Produs_ID: 1005 Nume: camasa
Produs_ID: 1000 Nume: joc
Produs_ID: 1020 Nume: minge
Produs_ID: 1007 Nume: pix

Clasa Dictionary<TKey, TValue>

Reprezintă o colecție de chei și valori.

Proprietățile clasei Dictionary<TKey, TValue>

Comparer	Returnează (obține) IEqualityComparer<T> care este folosită pentru a determina egalitatea cheilor din Dictionary<TKey,TValue>.
Count	Returnează numărul de perechi cheie/valoare cuprinse în Dictionary <TKey, TValue>.
Keys	Returnează o colecție (Dictionary <TKey, TValue>.KeyCollection) care conține cheile din Dictionary<TKey, TValue>.
Values	Returnează o colecție (Dictionary<TKey,TValue>.ValueCollection) care conține valorile din Dictionary<TKey, TValue>.
this[TKey]	Returnează sau setează valoarea asociată cheiei specificate (indexarea clasei List<T>).

Metodele clasei Dictionary<TKey, TValue>

Add (TKey, TValue)	Adaugă cheia specificată și valoarea în dicționar.
Clear	Elimină toate cheile și valorile din Dictionary <TKey, TValue>.
ContainsKey(TKey)	Determină dacă Dictionary<TKey, TValue> conține cheia specificată.
ContainsValue(TValue)	Determină dacă Dictionary <TKey, TValue> conține o anumită valoare.
GetEnumerator	Returnează un enumerator (structura Dictionary<TKey, TValue>.Enumerator) care iterează dictionarul Dictionary<TKey, TValue> . (Metoda MoveNext() din structura Dictionary<TKey, TValue>.Enumerator este apelata automat de bucla foreach)
GetObjectData	Implementeaza interfata System.Runtime.Serialization.ISerializable și returnează datele necesare pentru a serializa o instanță Dictionary<TKey, TValue> .
OnDeserialization	Implementeaza interfataSystem.Runtime.Serialization.ISerializable și produce un eveniment când deserializarea este completă.
Remove(TKey)	Șterge valoarea, corespunzătoare cheiei specificată, din Dictionary <TKey, TValue> .
TryGetValue (TKey, out TValue)	Returnează valoarea asociată cheiei specificată.

Următorul program demonstrează modul în care se poate utiliza clasa Dictionary<TKey, TValue> împreună cu proprietățile și metodele sale, însă și alte clase precum:
Dictionary <TKey, TValue>.KeyCollection,
Dictionary <TKey, TValue>.ValueCollection,
KeyNotFoundException
sau structuri:
KeyValuePair<TKey, TValue > .

Exemplu: Dictionary<TKey, TValue>

```
using System;
using System.Collections.Generic;

public class Produs  {
    public int pretProdus;
    public string numeProdus;

    public Produs(int pretProdus, string numeProdus)  {
        this.pretProdus = pretProdus;
        this.numeProdus = numeProdus;
    }

    public override string ToString() {
        return  numeProdus +" la pretul de " + pretProdus + " lei" ;
    }
}
```

```
class Exemplu  {
    public static void Main(){
        // Cream un dictionar (Dictionary) de Produse avand drept cuvinte cheie stringuri
        Dictionary<string, Produs> listaProduse =  new Dictionary<string, Produs>();

        //Cream cateva produse si le adaugam in catalog.
        //Cuvintele cheie nu pot fi duplicate. Valorile insa se pot duplica.
        Produs pix = new Produs(5, "Pix");
        listaProduse.Add("#1000", new Produs(50, "Minge"));
        listaProduse.Add("#1001", pix);
        listaProduse.Add("#1002", pix);
        listaProduse.Add("#1003", new Produs(100, "Carte"));

        // Metoda Add lanseaza o exceptie daca in lista se afla deja cuvantul cheie.
        try {
            listaProduse.Add("#1003", new Produs(1000, "Telefon"));
        }
        catch (ArgumentException) {
            Console.WriteLine("Elementul avand cuvantul cheie = \"#1003\" este deja in lista.");
        }
```

```
// Un element, poate fi accesat prin intermediul unei indexari.
        Console.WriteLine("Corespunzator ID-ului = #1003, se afla produsul: {0}.",
          listaProduse["#1003"]);

 // Indexarea poate fi utilizata pentru a schimba valoarea asociata unui cuvant cheie.
        listaProduse["#1003"] = pix;
        Console.WriteLine("Corespunzator ID-ului = #1003, se afla produsul: {0}.",
          listaProduse["#1003"]);

// Daca cuvantul cheie nu se afla in lista, atunci indexatea adauga acel element in lista.
        listaProduse["#1004"] = new Produs(3, "Un kg mere");

 // Indexarea lanseaza o exceptie daca cheia nu este in lista.
        try {
            Console.WriteLine("Corespunzator ID-ului = #1005, se afla produsul: {0}.",
          listaProduse["#1005"]);
        }
        catch (KeyNotFoundException) {
            Console.WriteLine("Cheia = #1005 nu a fost gasita.");
        }
```

```
// Daca un program are de incercat o serie de chei care ar putea sa nu fie in lista, atunci
// se poate utiliza proprietatea TryGetValue ca mai jos.
        Produs valoare;
        if (listaProduse.TryGetValue("#1004", out valoare)) {
            Console.WriteLine("Corespunzator ID-ului = #1004, se afla produsul: {0}.",
           valoare);
        }
        else {
            Console.WriteLine("Cheia = \"#1004\" nu a fost gasita.");
        }

 // Se poate utiliza metoda ContainsKey() pentru a vedea daca exista sau nu o anumita cheie.
        if (!listaProduse.ContainsKey("#1005")) {
            listaProduse.Add("#1005", new Produs(2, "Paine"));
            Console.WriteLine("Corespunzator ID-ului = #1005, se afla produsul: {0}.",
           listaProduse["#1005"]);
        }

 // Bucla foreach returneaza obiecte de tipul KeyValuePair, o structura din acelasi
 // spatiu de nume System.Collection.Generic. Acelasi concept privind foreach a fost intalnit si in
//cazul colectiilor non-generice (vezi clasa DictionaryBase si structura DictionaryEntry).
        Console.WriteLine();
        foreach (KeyValuePair<string,Produs> kvp in listaProduse) {
           Console.WriteLine("Corespunzator ID-ului = {0}, se afla produsul: {1}.",
           kvp.Key, kvp.Value);
        }
```

```
// Pentru a obtine doar valorile elementelor din lista se poate utiliza proprietatea Values
//care returneaza un obiect de tipul clasei Dictionary<string, Produs>.ValueCollection
    Dictionary<string, Produs>.ValueCollection valorileColectiei=listaProduse.Values;

 // Elementele colectiei ValueCollection sunt puternic tipizate,
 // in cazul nostru acestea sunt de tipul Produs.
    Console.WriteLine("Elementele colectiei
            Dictionary<string, Produs>.ValueCollection sunt de tipul Produs:");
      foreach (Produs p in valorileColectiei) {
         Console.WriteLine("{0}", p);
      }

  // Pentru a obtine cheile elementelor din lista se poate utiliza proprietatea Keys care
  / returneaza un obiect de tipul clasei Dictionary<string, Produs>.KeyCollection
       Dictionary<string, Produs>.KeyCollection cheileColectiei =listaProduse.Keys;
```

```
// Elementele colectiei KeyCollection sunt puternic tipizate,
// in cazul nostru acestea sunt de tipul string.
        Console.WriteLine("Elementele colectiei
                    Dictionary<string, Produs>.KeyCollection sunt de tipul string:");
        foreach (string s in cheileColectiei) {
            Console.WriteLine("{0}", s);
        }


 // Se poate utiliza metoda Remove() pentru a sterge o pereche cheie/valoare din
 //colectie.
        Console.WriteLine("\nStergem(#1003)");
        listaProduse.Remove("#1003");
        if (!listaProduse.ContainsKey("#1003")) {
            Console.WriteLine("Cheia #1003 nu a fost gasita.");
        }
    }
}
```

Rezultat:

Elementul avand cuvantul cheie = "#1003" este deja in lista.
Corespunzator ID-ului = #1003, se afla produsul: Carte la pretul de 100 lei.
Corespunzator ID-ului = #1003, se afla produsul: Pix la pretul de 5 lei.
Cheia = #1005 nu a fost gasita.
Corespunzator ID-ului = #1004, se afla produsul: Un kg mere la pretul de 3 lei.
Corespunzator ID-ului = #1005, se afla produsul: Paine la pretul de 2 lei.

Corespunzator ID-ului = #1000, se afla produsul: Minge la pretul de 50 lei.
Corespunzator ID-ului = #1001, se afla produsul: Pix la pretul de 5 lei.
Corespunzator ID-ului = #1002, se afla produsul: Pix la pretul de 5 lei.
Corespunzator ID-ului = #1003, se afla produsul: Pix la pretul de 5 lei.
Corespunzator ID-ului = #1004, se afla produsul: Un kg mere la pretul de 3 lei.
Corespunzator ID-ului = #1005, se afla produsul: Paine la pretul de 2 lei.
Elementele colectiei Dictionary<string, Produs>.ValueCollection sunt de tipul Produs:
Minge la pretul de 50 lei
Pix la pretul de 5 lei
Pix la pretul de 5 lei
Pix la pretul de 5 lei
Un kg mere la pretul de 3 lei
Paine la pretul de 2 lei
Elementele colectiei Dictionary<string, Produs>.KeyCollection sunt de tipul string:
#1000
#1001
#1002
#1003
#1004
#1005
Stergem(#1003)
Cheia #1003 nu a fost gasita.

Colecții sortate

Clasa negenerica System.Collections.SortedList, clasa generică System.Collections.Generic.SortedList <TKey, TValue> și clasa generică System.Collections.Generic.SortedDictionary <TKey, TValue> sunt similare clasei Dictionary < TKey, TValue > , în sensul că toate implementează interfața IDictionary, dar își păstrează elementele sortate după o cheie.

Cele trei clase au mai multe caracteristici în comun:
-Toate cele trei clase implementează System.Collections.IDictionary. Cele doua clase generice implementează, de asemenea, interfața generică System.Collections.Generic.IDictionary <TKey, TValue> .

-Fiecare element este o pereche cheie valoare (cu scopul de a putea fi enumerate). Cand se utilizeaza bucla foreach, SortedList returnează obiecte DictionaryEntry, în timp ce cele două tipuri generice returnează obiecte KeyValuePair <TKey, TValue> .

-Elementele sunt sortate conform implementării interfeței System.Collections.IComparer, pentru clasa negenerică SortedList, și respectiv implementării interfeței System.Collections.Generic.IComparer <T>, pentru cele două clase generice.

-Fiecare clasă oferă proprietăți (proprietățile Keys și Values) care returnează colecții care conțin numai cheile sau doar valorile.

Însă sunt și o serie de aspecte care diferențiază clasa SortedDictionary <TKey, TValue> față de clasele SortedList și SortedList <TKey, TValue> . Spre exemplu:

a) proprietățile Keys și Values din SortedList și SortedList <TKey, TValue> returnează chei și valori care sunt indexate. Așadar se poate utiliza o indexare. În schimb, pentru SortedDictionary <TKey, TValue> nu putem proceda la fel. (Vezi exemplul urmator).

b) SortedList și SortedList <TKey, TValue> utilizează mai puțină memorie decât SortedDictionary <TKey, TValue>.

Exemplu: SortedList<TKey, TValue> si SortedDictionary<TKey, TValue>

```
using System;
using System.Collections.Generic;

public class Produs{
    public int pretProdus;
    public string numeProdus;

    public Produs(int pretProdus, string numeProdus)    {
        this.pretProdus = pretProdus;
        this.numeProdus = numeProdus;
    }

    public override string ToString() {
        return  numeProdus + " la pretul de " + pretProdus+ " lei" ;
    }
}
```

```
class Exemplu{
    public static void Main(){
        SortedList<int, Produs> sortedListProduse=new SortedList<int,
Produs>();

        // Adaugam produse in lista SortedList<int, Produs>.
        sortedListProduse.Add(703, new Produs(2, "Pix"));
        sortedListProduse.Add(345,new Produs(50, "Minge"));
        sortedListProduse.Add(450,new Produs(2, "Caiet"));
        sortedListProduse.Add(606,new Produs(1000, "Telefon"));


        // Scriem elementele din lista SortedList<int, Produs>.
        Console.WriteLine("Elementele colectiei SortedList<int, Produs> sunt:");
        foreach (KeyValuePair<int,Produs> kvp in sortedListProduse)
        {
            Console.WriteLine("Cheia={0}, Valoarea={1}",kvp.Key, kvp.Value);
        }
        Console.WriteLine();
```

```
SortedDictionary<int, Produs> sortedDictionaryProduse = new SortedDictionary<int,
        Produs>();

    // Adaugam produse in lista sortedDictionaryProduse<int, Produs>.
    sortedDictionaryProduse.Add(703, new Produs(2, "Pix"));
    sortedDictionaryProduse.Add(345, new Produs(50, "Minge"));
    sortedDictionaryProduse.Add(450, new Produs(2, "Caiet"));
    sortedDictionaryProduse.Add(606, new Produs(1000, "Telefon"));


    // Scriem elementele din lista sortedDictionaryProduse<int, Produs>.
    Console.WriteLine("Elementele colectiei sortedDictionaryProduse<int, Produs>
        sunt:");
    foreach (KeyValuePair<int, Produs> kvp in sortedDictionaryProduse)
    {
       Console.WriteLine("Cheia={0}, Valoarea={1}", kvp.Key, kvp.Value);
    }
    Console.WriteLine();
```

```
//Intrucat propretatea Values (sau Keys) din SortedList<TKey, TValue>  intoarce un
//IList<T>, iar aceasta clasa contine
//indexarea this[int], putem accesa elementele listei IList<T> printr-un index.
        IList<Produs> listaProduse = sortedListProduse.Values;
        Console.WriteLine("Elementul cu indexul 2 in listaProduse este: {0}",
              listaProduse[2]);

//In schimb, pentru SortedDictionary<TKey, TValue> rezultatul intors de proprietatea
//Values este SortedDictionary<TKey, TValue>.ValueCollection, iar aceasta clasa nu
//este indexata.
//NU putem proceda ca mai sus. Daca stergeti comentariul de mai jos se obtine eroare.
        SortedDictionary<int, Produs>.ValueCollection valoarecolectieDictionary =
                                                  sortedDictionaryProduse.Values;
       // Console.WriteLine(valoarecolectieDictionary[2]);

    }
}
```

Rezultat:
Elementele colectiei SortedList<int, Produs> sunt:
Cheia=345, Valoarea=Minge la pretul de 50 lei
Cheia=450, Valoarea=Caiet la pretul de 2 lei
Cheia=606, Valoarea=Telefon la pretul de 1000 lei
Cheia=703, Valoarea=Pix la pretul de 2 lei

Elementele colectiei sortedDictionaryProduse<int, Produs> sunt:
Cheia=345, Valoarea=Minge la pretul de 50 lei
Cheia=450, Valoarea=Caiet la pretul de 2 lei
Cheia=606, Valoarea=Telefon la pretul de 1000 lei
Cheia=703, Valoarea=Pix la pretul de 2 lei

Elementul cu indexul 2 in listaProduse este: Telefon la pretul de 1000 lei

Cozi și stive

System.Collections.Queue, System.Collections.Generic.Queue <T> și System.Collections.Concurrent.ConcurrentQueue <T> sunt clase colecții first-in, first- out care implementeaza interfata ICollection. Clasele generice implementează și interfața ICollection < T >.

System.Collections.Stack, System.Collections.Generic.Stack <T> și System.Collections.Concurrent.ConcurrentStack <T> sunt clase colecții last-in, first- out care implementeaza interfata ICollection. Clasele generice implementează și interfața ICollection < T >.

Cozile și stive sunt utile atunci când aveți nevoie de depozitare temporară a unor informații, adică atunci când doriți să renunțați la un element după preluarea valorii sale.
Utilizați o coadă dacă aveți nevoie să accesați informațiile în aceeași ordine în care acestea sunt stocate în colecție.
Utilizați o stivă dacă aveți nevoie să accesați informațiile în ordine inversă.

Trei operații principale pot fi efectuate asupra unei cozi și elementelor sale:

- Enqueue	Adaugă un element la sfârșitul cozii.
- Dequeue	Elimină cel mai vechi element de la începutul cozii. Metoda TryDequeue returneaza false în cazul în care valoarea nu poate fi eliminată.
-Peek	Returnează cel mai vechi element care este la începutul cozii, dar nu-l scoate din coadă.

Trei operații principale pot fi efectuate asupra unei stive și elementelor sale:

-Push	Introduce un element în partea de sus a stivei.
-Pop	Inlătură un element din partea de sus a stivei.
-Peek	Returnează un element care este în partea de sus a stivei, dar nu-l scoate din stivă.