Тема 14. Колекції

План

14.1. Абстрактні структури даних
14.2. Колекції. Простір імен System.Collections
14.2.1. Клас Stack (Стек)
14.2.2. Клас Queue (Черга)
14.2.3. Клас ArrayList (Динамічний масив)
14.3. Класи-прототипи (класи з родовими параметрами)

14.1. Абстрактні структури даних

Будь-яка програма призначена для обробки даних, від способу організації яких залежить її алгоритм. Для різних завдань необхідні різні способи зберігання і обробки даних, тому вибір структур даних повинен передувати створенню алгоритмів і ґрунтуватися на вимогах до функціональності і швидкодії програми. Найчастіше в програмах використовуються *масив, список, стек, черга, бінарне дерево, хеш-таблиця, граф і множина*.

Масив — це скінченна сукупність однотипних величин. Масив займає безперервну область пам'яті і надає прямий доступ до своїх елементів за індексом. Пам'ять під масив виділяється до початку роботи з ним і потім не змінюється. Нагадаємо, що в C# є *одновимірні, прямокутні і зубчасті масиви*.

У **списку** кожен елемент зв'язаний з наступним і, можливо, з попереднім. У першому випадку список називається *однозв'язним*, в другому — *двозв'язним*. Якщо останній елемент зв'язати покажчиком з першим, вийде **кільцевий** список. Кількість елементів в списку може змінюватися в процесі роботи програми.

Черга — окремий випадок однонаправленого списку, додавання елементів в який виконується в один кінець, а вибірка — з іншого кінця. Інші операції з чергою не визначені. При вибірці елемент виключається з черги. Говорять, що черга реалізує принцип обслуговування FIFO (First In — First Out, першим прийшов, — першим пішов).

Бінарне дерево — динамічна структура даних, що складається з вузлів, кожен з яких містить, окрім даних, не більше двох посилань на різні бінарні піддерева. На кожен вузол є рівно одне посилання. Початковий вузол називається коренем дерева.

Хеш-таблиця, асоціативний масив, або **словник** — це масив, доступ до елементів якого здійснюється не за номером, а за деяким ключем. Можна сказати, що це таблиця, що складається з пар "ключ-значення" Хеш-таблиця ефективно реалізує операцію пошуку значення по ключу. При цьому ключ перетворюється в число (хеш-код), яке використовується для швидкого знаходження потрібного значення в хеш-таблиці.

Граф — це сукупність вузлів і ребер, що сполучають різні вузли. Багато реальних практичних задач можна описати в термінах графів, що робить їх структурою даних, яка часто використовується при написанні програм.

Множина — це невпорядкована сукупність елементів. Для множини визначені операції перевірки належності елементу множині, включення і виключення елементу, а також об'єднання, пересічення і віднімання множин.

Класи цих типів повинні реалізовувати відповідні інтерфейси, тобто *реалізувати допустимі операції* для роботи з даними. Ці класи називаються *колекціями*, або *контейнерами*. Для кожного типу колекції визначені свої методи роботи з елементами,

не залежні від конкретних даних в елементі, саме тому методи колекції можна використовувати для обробки даних різних типів. Вибір типу колекції визначається ти, що потрібно робити з даними в програмі і які вимоги пред'являються до її швидкодії.

14.2. Колекції. Простір імен System.Collections

У C# під колекцією розуміється деяка група об'єктів. Всі колекції розроблені на основі певних інтерфейсів, тому стандартизують спосіб обробки групи об'єктів.

Середовище .NET Framework підтримує три основні типи колекцій: загального призначення, спеціалізовані і орієнтовані на побітову організацію даних.

У просторі імен System.Collections визначені набори стандартних колекцій і інтерфейсів, які успадковують ці колекціяї. У таблиці 14.1 наведені найважливіші інтерфейси. Інтерфейси IComparer, IEnumerable, IEnumerator ми розглядали в лекції 14.

Простір імен System.Collections.Specialized містить спеціалізовані колекції, наприклад, колекцію рядків StringCollection і хеш-таблицю із ключами-рядками StringDictionary.

Таблиця 14.1. Інтерфейси простору імен System.Collections

Інтерфейс	Призначення
ICollection	Визначає загальні характеристики (наприклад, розмір) для набору елементів
IComparer	Дозволяє порівнювати два об'єкти
IDictionary	Дозволяє представляти вміст об'єкту у вигляді пар "ім'я-значення" (хеш-таблиці)
IDictionaryEnumerator	Використовується для нумерації вмісту об'єкту, що підтримує інтерфейс Idictionary
IEnumerable	Повертає інтерфейс IEnumerator для вказаного об'єкту
IEnumerator	Зазвичай використовується для підтримки оператора foreach відносно об'єктів
IHashCodeProvider	Повертає хэш-код для реалізації типу із застосуванням вибраного користувачем алгоритму хешування
Ilist	Підтримує методи додавання, видалення і індексування елементів в списку об'єктів

У таблиці 14.2 перелічені основні колекції, визначені в просторі System.Collections.

Таблиця 14.2. Класи колекцій загального призначення (з простору імен System.Collections)

Клас	Призначення	Найважливіші з реалізованих інтерфейсів
ArrayList	Динамічний масив	IList, ICollection, IEnumerable, ICloneable
BitArray	Компактний масив для зберігання бітових значень	ICollection, IEnumerable, ICloneable
Hashtable	Хеш-таблиця	IDictionary, ICollection, IEnumerable, ICloneable
Queue	Черга	ICollection, ICloneable, Ienumerable
SortedList	Колекція, відсортована по ключах. Доступ до елементів — по ключу або по індексу	IDictionary, ICollection, IEnumerable, ICloneable
Stack	Стек	ICollection, IEnumerable

14.2.1. Клас Stack

Стек — окремий випадок однонаправленого списку, додавання елементів в який і вибірка з якого виконуються з одного кінця, який називається вершиною стека (головою – head). Інші операції із стеком не визначені. При вибірці елемент виключається із стека. Говорять, що стек реалізує принцип обслуговування LIFO (Last In — First Out, останнім прийшов, — першим пішов).

У C# реалізацію стеку представляє клас **Stack**, який реалізує інтерфейси ICollection, IEnumerable і ICloneable. Stack - це динамічна колекція, розмір якої змінюється.

У класі Stack визначені наступні конструктори:

public Stack(); //створює порожній стек, початкова кількість елементів якого дорівнює 0

public Stack(int capacity); //створює порожній стек, початкова кількість елементів якого дорівнює capacity

public Stack(ICollection c); //створює стек, який містить елементи колекції, заданої параметром c, початкова кількість елементів якого дорівнює 0.

Окрім методів, визначених в інтерфейсах, що реалізуються класом Stack, в цьому класі визначені власні методи

Таблиця 14.3. Методи класу Stack

Метод	Опис

public virtual bool Contains(object v)	Повертає значення true, якщо об'єкт **v** міститься у визиваючому стеку, інакше повертає значення false.
public virtual void Clear()	Встановлює властивість Count рівній нулю, тим самим очищаючи стек.
public virtual object Peek()	Повертає елемент, розташований у вершині стека, але не витягуючи його із стека
public virtual object Pop()	Повертає елемент, розташований у вершині стека, і витягує його із стека
public virtual void Push(object v)	Поміщає об'єкт v у стек
public virtual object[] ToArray()	Повертає масив, який містить копії елементів викликаючого стека

Розглянемо декілька прикладів використання стека.

Приклад 14.1. Для заданого значення n запишемо в стек всі числа від 1 до n, а потім будемо вибирати їх із стека:

```
using System;
using System.Collections;
using System.Linq;
using System.Text;

namespace Example14_1
{
 class Program
 {
  static void Main(string[] args)
  {
   Console.Write("n= ");
   int n = int.Parse(Console.ReadLine());

   Stack intStack = new Stack();
   for (int i = 1; i <= n; i++)
      intStack.Push(i);

   Console.WriteLine("Розмірність стека " + intStack.Count);
   Console.WriteLine("Верхній елемент стека = " + intStack.Peek());
   Console.WriteLine("Розмірність стека " + intStack.Count);

   Console.Write("Вміст стека = ");
   while (intStack.Count != 0)
      Console.Write("{0} ", intStack.Pop());

   Console.WriteLine("\nНова розмірність стека " + intStack.Count);
   Console.ReadKey();
  }
 }

}
```

Приклад 14.2. У текстовому файлі міститься математичний вираз. Необхідно перевірити баланс круглих дужок у виразі.

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
using System.IO;

namespace Example14_2
{
 class Program
 {
  static void Main(string[] args)
  {
   StreamReader fileIn=new StreamReader("t.txt");
   string line=fileIn.ReadToEnd();
   fileIn.Close();

   Stack brackets=new Stack();
   bool prap=true;

   //перевіряємо баланс дужок
   for ( int i=0; i<line.Length;i++)
   {
   //якщо поточний символ дужка, що відкривається, то поміщаємо її в стек
    if (line[i]== '(') brackets.Push(i);
    else if (line[i]== ')') //якщо поточний символ дужка, що закривається, то
    {

    //якщо стек порожній, то для закритої дужки не вистачає парної відкритої
    if (brackets.Count == 0)
    {
      prap = false;
      Console.WriteLine("Можливо в позиції " + i + "зайва ) дужка");
    }
    else brackets.Pop(); //інакше витягуємо парну дужку
    }
   }

   //якщо після перегляду рядка стек виявився порожнім, то дужки збалансовані
   if (brackets.Count == 0)
    {
      if (prap) Console.WriteLine("дужки збалансовані"); }
      else //инакше баланс дужок порушений
      {
        Console.Write("Можливо зайва ( дужка в позиції:");
        while (brackets.Count != 0)
         {
             Console.Write("{0} ", (int)brackets.Pop());
         }
        Console.WriteLine();
     }
  }
 }
}
```

в файлі t.txt знаходиться вираз: **(1+2)-4*(a-3)/(2-7+6)**

14.2.2. Клас Queue (Черга)

Черга - це окремий випадок однонаправленого списку, додавання елементів в який виконується в один кінець (хвіст), а вибірка робиться з іншого кінця (голови). Інші операції з чергою не визначені. При вибірці елемент виключається з черги. Говорять, що черга реалізує принцип обслуговування FIFO (fist in - fist out, першим прийшов, - першим вийшов).

У C# реалізація принципу FIFO здійснюється за допомогою класу **Queue**, який, як і стек, реалізує інтерфейси ICollection, IEnumerable і ICloneable.

Queue - це динамічна колекція, розмір якої змінюється. При необхідності збільшення місткості черги відбувається з коефіцієнтом зростання за умовчанням рівним 2.0.

У класі Queue визначені наступні конструктори:

public Queue(); //створює порожню чергу, початкова місткість якої дорівнює 32

public Queue (int capacity); // створює порожню чергу, початкова місткість якої рівна capacity

public Queue (int capacity, float n); //створює порожню чергу, початкова місткість якої рівна capacity, і коефіцієнт зростання встановлюється параметром n

public Queue (ICollection c); //створює чергу, яка містить елементи колекції, заданої параметром c, і аналогічною місткістю

Окрім методів, визначених в інтерфейсах, що реалізуються класом Queue, в цьому класі визначені власні методи:

Таблиця 14.4. Методи класу Queue

Метод	Опис
public virtual bool Contains (object v)	Повертає значення true, якщо об'єкт v міститься в черзі, інакше повертає значення false
public virtual void clear ()	Встановлює властивість Count рівною нулю, тим самим очищаючи чергу
public virtual object Dequeue ()	Повертає об'єкт з початку черги, видаляючи його з черги
public virtual object Peek ()	Повертає об'єкт з початку черги, не видаляючи його з черги
public virtual void Enqueue(object v)	Додає об'єкт v в кінець черги
public virtual object [ ] ToArray ()	Повертає масив, який містить копії елементів черги
public virtual void TrimToSizeO	Встановлює властивість Capacity рівною значенню властивості Count

Розглянемо декілька прикладів використання черги.

Приклад 14.3. Для заданого значення n запишемо в чергу всі числа від 1 до n, а потім будемо вибирати їх з черги:

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
using System.IO;

namespace Example14_3
{
 class Program
 {
  static void Main(string[] args)
  {
   Console.Write("n= ");
```

```
        int n = int.Parse(Console.ReadLine());

        Queue intQ = new Queue();
        for (int i = 1; i <= n; i++)
         intQ.Enqueue(i);

        Console.WriteLine("Розмірність черги " + intQ.Count);
        Console.WriteLine("Верхній елемент черги = " + intQ.Peek());
        Console.WriteLine("Розмірність черги " + intQ.Count);

        Console.Write("Вміст черги = ");
        while (intQ.Count != 0)
         Console.Write("{0} ", intQ.Dequeue());

        Console.WriteLine("\nНова розмірність черги " + intQ.Count);
      }
     }
    }
```

Приклад 14.4. У текстовому файлі записана інформація про людей (прізвище, ім'я, по батькові, вік, вага через `flag`). Вивести на екран спочатку інформацію про людей молодших 40 років, а потім інформацію про всіх інших.

```
using System;
using System.Collections;
using System.IO;
using System.Linq;
using System.Text;

namespace Example14_4
{
    public struct Person //структура для зберігання даних про одну людину
    {
        public string LastName;
        public string Name;
        public string MiddleName;
        public int Age;
        public float Weight;
    }
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader fileIn = new StreamReader("t.txt", Encoding.GetEncoding(1251));
            string line;

            Queue people = new Queue();
            Person person;

            Console.WriteLine("ВІК МЕНШЕ 40 РОКІВ");
            while ((line = fileIn.ReadLine()) != null) //читаємо до кінця файлу
            {
                string[] temp = line.Split(' '); //розбиваємо рядок на складові елементи
                                                 //заповнюємо структуру

                person.LastName = temp[0];
                person.Name = temp[1];
                person.MiddleName = temp[2];
                person.Age = int.Parse(temp[3]);
                person.Weight = float.Parse(temp[4]);

                // якщо вік менше 40 років, то виводимо дані на екран, інакше поміщаємо їх в
                //черга для тимчасового зберігання
                if (person.age < 40)
                    Console.WriteLine(person.LastName +
                        "\t" + person.Name +
                        "\t" + person.MiddleName +
```

```
                    "\t" + person.Age +
                    "\t" + person.Weight);
                else people.Enqueue(person);
            }
            fileIn.Close();

            Console.WriteLine("ВІК 40 РОКІВ І СТАРШЕ");
            while (people.Count != 0) // вибираємо з черги дані
            {
                person = (Person)people.Dequeue();
                Console.WriteLine(person.LastName +
                    "\t" + person.Name +
                    "\t" + person.MiddleName +
                    "\t" + person.Age +
                    "\t" + person.Weight);
            }
            Console.ReadKey();
        }
    }
}
```

14.2.3. Клас ArrayList

Основним недоліком звичайних масивів є те, що об'єм пам'яті, необхідний для зберігання їх елементів, має бути виділений до початку роботи з масивом. Об'єкт класу ArrayList є масивом змінної довжини, елементами якого є посилання на об'єкти. Будь-який об'єкт класу ArrayList створюється з деяким початковим розміром.

Клас ArrayList реалізує інтерфейси ICollection, IList, IEnumerable і ICloneable. У класі ArrayList визначені наступні конструктори:

public ArrayList() //створює порожній масив з максимальною ємкістю рівною 16 елементам, при поточній розмірності 0

public ArrayList(int capacity) // створює масив із заданою ємкістю capacity, при поточній розмірності 0

public ArrayList(ICollection c) // створює масив, який ініціалізується елементами колекції c

Крім методів, визначених в інтерфейсах, які реалізує клас ArrayList, в ньому визначені і власні методи:

Таблиця 14.5. Методи класу ArrayList

Метод	Опис
public virtual void AddRange (ICollection c)	Додає елементи з колекції **c** в кінець визиваючої колекції
public virtual int BinarySearch (object v)	У визиваючій відсортованій колекції виконує пошук значення, заданого параметром **v**. Повертає індекс знайденого елементу. Якщо потрібне значення не знайдене, повертає від'ємне значення.
public virtual int BinarySearch (object v, IComparer comp)	У визиваючій відсортованій колекції виконує пошук значення, яке задане параметром **v**, на основі методу порівняння об'єктів, заданого параметром

	comp. Повертає індекс знайденого елементу. Якщо потрібне значення не знайденен, повертає від'ємне значення.
public virtual int BinarySearch (int startIdx, int count, object v, IComparer comp)	В визиваючій відсортованій колекції виконує пошук значення, заданого параметром **v**, на основі методу порівняння об'єктів, заданого параметром comp. Пошук починається з елементу, індекс якого дорівнює значенню **startIdx**, і включає **count** елементів. Метод повертає індекс знайденого елементу. Якщо потрібне значення не знайдене, повертає від'ємне значення.
public virtual void CopyTo(Array ar, int startIdx)	Копіює вміст визиваючої колекції, починаючи з елементу, індекс якого дорівнює значенню **startIdx**, в масив, заданий параметром **ar**. Масив має бути одновимірним і сумісним за типом з елементами колекції.
public virtual void CopyTo(int srcIdx, Array ar, int destIdx, int count)	Копіює count елементів визиваючої колекції, починаючи з елементу, індекс якого дорівнює значенню **srcIdx**, в масив, заданий параметром **ar**, починаючи з елементу, індекс якого дорівнює значенню **destIdx**. Масив має бути одновимірним і сумісним за типом з елементами колекції
public virtual ArrayList GetRange(int idx, int count)	Повертає частину визиваючої колекції типа ArrayList. Діапазон колекції, яка повертається, починається з індексу idx і включає count елементів. Об'єкт, що повертається, посилається на ті ж елементи, що і визиваючий об'єкт
public static ArrayList FixedSize(ArrayList ar)	Перетворює колекцію **ar** на ArrayList-масив з фіксованим розміром і повертає результат
public virtual void InsertRange(int startIdx, ICollection c)	Вставляє елементи колекції, заданої параметром **c**, в колекцію, починаючи з індексу, заданого параметром **startIdx**
public virtual int LastIndexOf(object v)	Повертає індекс останнього входження об'єкту **v** в колекції. Якщо шуканий об'єкт не знайдений, повертає від'ємне значення

public static ArrayList Readonly(ArrayList ar)	Перетворює колекцію **ar** на ArrayList-масив, призначений лише для читання
public virtual void RemoveRange(int idx, int count)	Видаляє **count** елементів із колекції, починаючи з елементу, індекс якого дорівнює значенню **idx**
public virtual void Reverse()	Розташовує елементи колекції в зворотному порядку
public virtual void Reverse(int startldx, int count)	Розташовує в зворотному порядку **count** елементів колекції, починаючи з індексу **startldx**
public virtual void SetRange(int startldx, ICollection c)	Замінює елементи колекції, починаючи з індексу **startldx**, елементами колекції, заданої параметром **c**
public virtual void Sort()	Сортує колекцію в порядку збільшення елементів
public virtual void Sort(IComparer comp)	Сортує колекцію на основі методу порівняння об'єктів, заданого параметром **comp**. Якщо параметр **comp** має нульове значення, для кожного об'єкту використовується стандартний метод порівняння
public virtual void Sort (int startidx, int endidx, icomparer comp)	Сортує частину колекції на основі методу порівняння об'єктів, заданого параметром **comp**. Сортування починається з індексу **startidx** і закінчується індексом **endidx**. Якщо параметр **comp** має нульове значення, для кожного об'єкту використовується стандартний метод порівняння
public virtual object [ ] ToArray ()	Повертає масив, який містить копії елементів визиваючого об'єкту
public virtual Array ToArray (Type type)	Повертає масив, який містить копії елементів об'єкту. Тип елементів в цьому масиві задається параметром **type**
public virtual void TrimToSize()	Встановлює властивість **Capacity** рівним значенню властивості **Count**

За замовчуванням при створенні об'єкту типу ArrayList будується масив з **16** елементів типу object. Можна задати бажану кількість елементів в масиві, передавши його в конструктор або встановивши як значення властивості Capacity, наприклад:

```
ArrayList arr1 = new ArrayList();   // створюється масив з 16 елементів
```

ArrayList arr2 = new ArrayList(1000); // створюється масив з 1000 елементів
ArrayList arr3 = new ArrayList();
arr3.Capacity = 1000; // кількість елементів задається

Клас ArrayList реалізований через клас Array, тобто містить закрите поле цього класу. Оскільки всі типи в C# є нащадками класу object, масив може містити елементи довільного типа. Навіть якщо в масиві зберігаються звичайні цілі числа, тобто елементи значимого типа, внутрішній клас є масивом посилань на екземпляри типу object, які є упакованими типами-значеннями. Відповідно, при занесенні в масив виконується упаковка, а при виборі — розпаковування елементу.

Якщо при додаванні елементу в масив виявляється, що фактична кількість елементів масиву перевищує його ємкість, вона автоматично подвоюється, тобто відбувається повторне виділення пам'яті і переписування туди всіх існуючих елементів. Приклад занесення елементів в екземпляр класу ArrayList:

arr1.Add(123); arr1.Add(-2); arr1.Add("Вася");

Доступ до елементу виконується за індексом, проте при цьому необхідно явним чином привести отримане посилання до цільового типу, наприклад:

int a = (int) arr1[0];
int b = (int) arr1[1];
string s = (string) arr1[2];

Спроба приведення до типу, який не відповідає типу що зберігається в елементі, викликає генерацію виключення InvalidCastException.

Розглянемо приклад використання класу ArrayList.

```
using System;
using System.Collections;

namespace Example14_5
{
 class Program
 {
  static void ArrayPrint(string s, ArrayList a)
  {
   Console.WriteLine(s);
   foreach (int i in a)
    Console.Write(i + " ");
   Console.WriteLine();
  }

  static void Main(string[] args)
  {
   ArrayList myArray = new ArrayList();
   Console.WriteLine("Початкова ємкість масиву: " + myArray.Capacity);
   Console.WriteLine("Початкова кількість елементів: " + myArray.Count);

   Console.WriteLine("\nДобавили 5 цифр");
   for (int i = 0; i < 5; i++) myArray.Add(i);
   Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
   Console.WriteLine("Поточна кількість елементів: " + myArray.Count);
   ArrayPrint("Вміст масиву", myArray);

   Console.WriteLine("\nОптимізуємо ємність масиву");
   myArray.Capacity = myArray.Count;
   Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
   Console.WriteLine("Поточна кількість елементів: " + myArray.Count);
   ArrayPrint("Вміст масиву", myArray);

   Console.WriteLine("\nДодаємо елементи в масив");
   myArray.Add(10);
```

```
            myArray.Insert(1, 0);
            myArray.AddRange(myArray);
            Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " + myArray.Count);
            ArrayPrint("Вміст масиву", myArray);

            Console.WriteLine("\nВидаляємо елементи з масиву");
            myArray.Remove(0);
            myArray.RemoveAt(10);
            Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " + myArray.Count);
            ArrayPrint("Вміст масиву", myArray);

            Console.WriteLine("\nВидаляємо весь масив");
            myArray.Clear();
            Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " + myArray.Count);
            ArrayPrint("Вміст масиву", myArray);
            Console.ReadKey();
        }
    }
}
```

14.3. Класи-прототипи (generics)

Класи-прототипи (generics)— це класи, що мають у якості параметрів типи даних. Найчастіше їх застосовують для зберігання даних, тобто як контейнерні класи, або колекції. У другу версію бібліотеки .NET додані параметризовані колекції для представлення основних структур даних — *стека, черги, списку, словника* тощо. Ці колекції, розташовані в просторі імен System.Collections.Generic, дублюють аналогічні колекції простору імен System.Collections.

Класи-прототипи називають також *родовими* або *шаблонними*, оскільки вони є зразками, за якими під час виконання програми будуються конкретні класи.

У таблиці 14.6 наведено відповідність між звичайними і параметризованими колекціями бібліотеки .NET (параметри, що визначають типи даних, які зберігаються в колекції, вказані в кутових дужках).

Таблиця 14.6. Параметризовані колекції

Клас-прототип (версія 2.0)	Звичайний клас
Comparer<T>	Comparer
Dictionary<K,T>	HashTable
LinkedList<T>	—
List<T>	ArrayList
Queue<T>	Queue
SortedDictionary<K,T>	SortedList
Stack<T>	Stack<T>

Як приклад розглянемо використання універсального "двійника" класу ArrayList — класу List<T> — для зберігання колекції об'єктів класу Person, а також для зберігання цілих чисел.

Приклад 14.6

```
namespace Example14_6
{
    public class Person
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public string Role { get; set; }

        public Person(string name, int age)
        {
            Name = name;
            Age = age;
        }
        public void Passport()
        {
            Console.WriteLine("Name = {0} Age = {1}", Name, Age);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            List<Person> persons = new List<Person>();
            persons.Add(new Person("Ковтун", 18));
            persons.Add(new Person("Петрововський", 20));
            persons.Add(new Person("Мороз", 3));

            foreach (Person person in persons)
                person.Passport();

            List<int> listInt = new List<int>();
            listInt.Add(5);
            listInt.Add(1);
            listInt.Add(3);

            listInt.Sort();
            int a = listInt[2];
            Console.WriteLine(a);

            foreach (int number in listInt)
                Console.Write(number + " ");

            Console.ReadLine();
        }
    }
}
```

У цьому прикладі створюється дві колекції. Перша (`persons`) містить елементи класу Person. Колекція `listInt` складається з цілих чисел, причому для роботи з ними не потрібні явні перетворення типу при отриманні елементу з колекції.

Висновки

В цій лекції ми розглянули основні класи-колекції C# і приклади роботи з ними. Колекції спрощують реалізацію багатьох завдань програмування, пропонуючи вже готові рішення для побудови структур даних. Всі колекції розроблені на основі певних інтерфейсів, тому стандартизують спосіб обробки групи об'єктів. Середовище .NET

Framework підтримує три основні типи колекцій: загального призначення, спеціалізовані і орієнтовані на побітову організацію даних.

Крім того, ми стисло розглянули параметризовані колекції, або класи-прототипи. Ці класи є зразками, за якими під час виконання програми будуються конкретні класи.

Питання і завдання для самостійної роботи студента

1. Вивчіть по довідковій системі склад простору імен System.Collections.
2. Вивчіть по довідковій системі властивості і методи класу ArrayList.
3. Опишіть основні види абстрактних структур даних.
4. Чи допускає структура "лінійний список" прямий доступ до своїх елементів?
5. Чим стек відрізняється від черги?
6. У яких завданнях використовується стек? Наведіть приклади.
7. У чому перевага масиву перед іншими структурами даних?
8. Які методи містить клас ArrayList?
9. Чим клас ArrayList відрізняється від звичайних масивів? Коли краще застосовувати клас ArrayList?