# Практическая работа № 11 . Абстрактные типы данных. Коллекции.

**Цель**.

* Получить практические навыки работы с коллекциями.
* Получить практические навыки работы со стандартными коллекциями пространства имен System.Collection.
* Получить практические навыки работы со стандартными параметризированными коллекциями пространства имен System.Collection.Generic.

## Часть 1. Абстрактные типы данных. Коллекции.

### Теоретические сведения.+

**Коллекция** в программировании — программный объект, содержащий в себе набор значений одного или различных типов, и позволяющий обращаться к этим значениям.

Назначение коллекции — служить хранилищем объектов и обеспечивать доступ к ним. Обычно коллекции используются для хранения групп однотипных объектов, подлежащих стереотипной обработке. Для обращения к конкретному элементу коллекции могут использоваться различные методы, в зависимости от того, какой абстрактный тип данных она реализует.

Различают:

* + **логическое** (абстрактный тип данных)
  + и **физическое** (реализация абстрактного типа данных) представление данных.

**Абстрактный тип данных** — это тип данных, который предоставляет для работы с элементами этого типа определённый набор функций, а также возможность создавать элементы этого типа при помощи специальных функций. Вся внутренняя структура такого типа спрятана от разработчика программного обеспечения — в этом и заключается суть абстракции. Абстрактный тип данных определяет набор функций, независимых от конкретной реализации типа, для оперирования его значениями. Конкретные реализации АТД называются структурами данных.

**Основные АТД.**

1. **Множество.**

Множество ­ это неупорядоченная совокупность элементов. Для множеств определены операции:

1. проверки принадлежности элемента множеству,
2. включения элемента,
3. исключения элемента,
4. объединения множеств,
5. пересечения множеств,
6. вычитания множеств.
7. **Словарь.**

Ассоциативный массив, или словарь ­ это массив, доступ к элементам которого осуществляется не по номеру, а по некоторому ключу, т.е. это таблица, состоящая из пар ключ-значение.

В качестве ключа могут использоваться значения различных типов, единственное ограничение – тип ключа должен допускать сравнение на равенство.

Может вводиться запрет на дублирование ключей в коллекции. Если такого ограничения нет, то при обращении по дублирующемуся ключу может выдаваться либо n-е найденное значение, либо все значения с данным ключом.

Операции, определенные над словарем:

1. Добавление пары ключ значение.
2. Удаление пары ключ значение по ключу.
3. Получение значения по ключу.

**3. Очередь**

Очередь – набор данных, реализующий принцип хранения «FIFO» («первым пришёл – первым вышел»). В очереди постоянно доступен только один элемент — тот, который был добавлен самым первым из имеющихся. При добавлении нового элемента он попадает в очередь. Текущий элемент можно удалить — тогда текущим станет элемент, добавленный первым из оставшихся.

Операции, определенные над очередью:

1. Помещение элемента в очередь.
2. Извлечение элемента из очереди.
3. **Стек**

Стек – набор данных, реализующий принцип хранения «LIFO» («последним пришёл – первым вышел»). В стеке постоянно доступен только один элемент – тот, который был добавлен последним. Новый элемент может быть добавлен в стек, он станет текущим. Текущий элемент всегда можно удалить («взять») из стека, после этого становится доступен элемент, который был добавлен непосредственно перед ним.

Операции определенные над стеком

1. Помещение элемента в стек.
2. Удаление элемента из стека.
3. **Список**

Список – набор данных, к которым возможен последовательный доступ. В любой момент доступен первый и последний элемент списка. От любого элемента списка можно получить доступ к следующему и к предыдущему по порядку, таким образом, можно последовательно дойти до любого желаемого. Новый элемент может добавляться в начало или в конец списка.

Операции определенные над списком:

1. Добавление элемента в начало списка.
2. Добавление элемента в конец списка.
3. Получение элемента из начала списка.
4. Получение элемента из конца списка.
5. Получение следующего элемента списка.

**Физическое представление данных**

Структура данных – это программная единица, позволяющая хранить и обрабатывать какие либо связанные данные. Используется для реализации каких-либо абстрактных типов данных. Структура данных определяет то, как данные будут размещены в памяти и соответственно время, необходимое для выполнения тех или иных операций над хранимыми данными.

1. **Массив**

Массив – это структура данных имеющая постоянную длину, порядок расположения элементов в массиве совпадает с порядком расположения элементов в памяти компьютера. Операции, связанные с изменением размера массива, как правило, требуют перераспределения памяти под новый массив и переноса элементов из старого массива в новый.

Все массивы в С# имеют общий базовый класс Array, определенный в пространстве имен System. В нем есть несколько полезных методов, упрощающих работу с массивами, например, методы получения размерности, сортировки и поиска.

Все массивы в С# построены на основе базового класса Array, который содержит полезные для программиста свойства и методы, часть из которых перечислены в табл. 1.

Таблица . Методы класса Array

|  |  |  |
| --- | --- | --- |
| Имя | Вид | Описание |
| Length | Свойство | Количество элементов массива (по всем размерностям) |
| Rank | Свойство | Количество размерностей |
| BinarySearch | Статический метод | Бинарный поиск |
| Clear | Статический метод | Присваивание значений элементам массива по умолчанию |
| Copy | Статический метод | Копирование заданного диапазона элементов из одного массива в другой |
| CopyTo | Метод | Копирование всех элементов текущего одномерного массива в другой массив |
| GetValue | Метод | Получение значения элемента массива |
| IndexOf | Статический метод | Поиск первого вхождения элемента в массив |
| LastIndexOf | Статический метод | Поиск последнего вхождения элемента в массив |
| Reverse | Статический метод | Переворот массива |
| SetValue | Метод | Установка значения элемента массива |
| Sort | Статический метод | Сортировка |

1. **Связный список**

Связный список – это структура данных, состоящая из узлов, каждый из которых содержит как собственные данные, так и одну или две ссылки на следующие и/или предыдущие узлы списка. Расположение элементов списков в памяти компьютера не совпадает с расположением элементов в списке. Обращение к произвольному элементу списка требует прохода по списку. Операции добавления и удаления элементов из списка не требуют перераспределения памяти под всю структуру данных.

class LPoint

{

public int data;

public LPoint next;

public LPoint(int number)

{

data = number;

next = null;

}

public override string ToString()

{

return data.ToString();

}

}

1. **Двоичное дерево**

Двоичное дерево – это структура данных, состоящая из узлов, каждый из которых содержит два указателя на узлы потомки. Узел, не имеющий указателя на родителя называется корнем дерева, узел, не имеющий потомков называется листом дерева. Расположение элементов дерева в памяти компьютера не совпадает с расположением элементов в дереве. К дереву применяются операция обращения к элементу дерева, добавления элемента в дерева и удаления элемента дерева.

**Пример**

class TPoint

{

public int data;

public TPoint left, right;

public TPoint(int number)

{

data = number;

left = null;

right=null;

}

public override string ToString()

{

return data.ToString();

}

}

1. **Хеш-таблица**

Хеш-таблица – это структура данных представляющая собой комбинацию массива и списков. Каждый элемент массива представляет собой указатель на начало списка. При помещении элемента в структуру сначала вычисляется хеш-функция (функция осуществляющая преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины) и в зависимости от полученного результата элемент помещается в один из списков, полученное значение является индексом массива. Число хранимых элементов, делённое на размер массива (число возможных значений хэш-функции) называется коэффициентом заполнения хэш-таблицы и является важным параметром, от которого зависит среднее время выполнения операций. Также хеш-функция должна равномерно обеспечивать равномерное распределение получаемых значений.

class LPoint

{

public string key;

public int value;

public LPoint next;

public LPoint(string s)

{

key = s;

value = key.GetHashCode();

next = null;

}

public override string ToString()

{

return key+":"+value.ToString();

}

public override int GetHashCode()

{

return key.GetHashCode();

}

}

class HTable

{

public LPoint[] table;

public int Size;

public HTable()

{

Size = 10;

table = new LPoint[10];

}

. . . .

}

## 2. Постановка задачи

1. Реализовать коллекцию, указанную в варианте.
2. Написать демонстрационную программу. При работе с коллекцией использовать объекты из иерархии классов, разработанной в работе №10.

Методы и свойства для коллекции **MyArray.**

|  |  |  |
| --- | --- | --- |
| **Элемент** | **Вид** | **Назначение** |
| Capacity | Свойство | Количество элементов, которые могут  храниться в массиве |
| Count | Свойство | Фактическое количество элементов массива |
| Item | Свойство | Получить или установить значение элемента по заданному индексу |
| Add(object value) | Метод | Добавление элемента в конец массива |
| BinarySearch(object value, IComparer compare) | Метод | Двоичный поиск в отсортированном массиве или его части |
| Clear() | Метод | Удаление всех элементов из массива |
| Clone() | Метод | Поверхностное копирование элементов одного массива в другой массив |
| IndexOf(object value) | Метод | Поиск первого вхождения элемента в массив (возвращает индекс найденного элемента или -1, если элемент не найден) |
| Insert(int index, object value) | Метод | Вставка элемента в заданную позицию (по заданному индексу) |
| LastIndexOf(object value) | Метод | Поиск последнего вхождения элемента в одномерный массив |
| Remove(object value) | Метод | Удаление первого вхождения заданного элемента в массив |
| RemoveAt(int index) | Метод | Удаление элемента из массива по заданному индексу |
| Reverse() | Метод | Изменение порядка следования элементов на обратный |
| Sort (IComparer comparer) | Метод | Упорядочивание элементов массива |

Методы и свойства для коллекции **HashTable.**

|  |  |  |
| --- | --- | --- |
| Элемент | Вид | Назначение |
|  |  |  |
| Capacity | Свойство | Количество элементов, которые могут  храниться в коллекции |
| Count | Свойство | Фактическое количество элементов коллекции |
| Keys | Свойство | Получить коллекцию ключей |
| Values | Свойство | Получить коллекцию значений |
| ContainsKey(object key) | Метод | Возвращает true , если в вызывающей  коллекции содержится ключ, заданный параметром*.* В противном случае возвращает значение false. |
| ContainsValue(object value) | Метод | Возвращает true , если в вызывающей  коллекции содержится значение, заданное параметром*.* В противном случае возвращает значение false. |
| Add(object key, object value) | Метод | Добавление элемента в конец коллекции |
| Clear() | Метод | Удаление всех элементов из коллекции |
| Clone() | Метод | Поверхностное копирование элементов одной коллекции в другую |
| Remove(object value) | Метод | Удаление первого вхождения заданного элемента в коллекцию |

Методы и свойства для коллекции **MySortedList.**

|  |  |  |
| --- | --- | --- |
| **Элемент** | **Вид** | **Назначение** |
| Capacity | Свойство | Количество элементов, которые могут  храниться в коллекции |
| Count | Свойство | Фактическое количество элементов коллекции |
| Keys | Свойство | Получить коллекцию ключей |
| Values | Свойство | Получить коллекцию значений |
| ContainsKey (object key) | Метод | Возвращает значение true , если в коллекции содержится ключ, заданный параметром*.* В противном случае возвращает значение false |
| ContainsValue(object value) | Метод | Возвращает значение true , если в коллекции содержится значение, заданное параметром*.* В противном случае возвращает значение false |
| GetByIndex (int index) | Метод | Возвращает значение, индекс которого задан параметром |
| GetKey(int index) | Метод | Возвращает ключ, индекс которого задан параметром |
| IndexOfKey(object key) | Метод | Возвращает индекс ключа, заданного параметром *к.* Возвращает значение - 1 , если в списке нет заданного ключа. |
| IndexOfValue(object value) | Метод | Возвращает индекс первого вхождения значения, заданного параметром *v.* Возвращает -1, если в списке нет заданного ключа |
| SetByIndex(int index*,* object value*)* | Метод | Устанавливает значение по индексу, заданному параметром index***,*** равным значению, переданному в параметре value. |
| Add(object key, object value) | Метод | Добавление элемента в конец коллекции |
| Clear() | Метод | Удаление всех элементов из коллекции |
| Clone() | Метод | Поверхностное копирование элементов одной коллекции в другую |
| Remove(object value) | Метод | Удаление первого вхождения заданного элемента в массив |
| RemoveAt(int index) | Метод | Удаление элемента |

Методы для коллекци **MyStack.**

|  |  |  |
| --- | --- | --- |
| **Элемент** | **Вид** | **Назначение** |
| Capacity | Свойство | Количество элементов, которые могут  храниться в коллекции |
| Count | Свойство | Фактическое количество элементов коллекции |
| Contains(object v) | Метод | Возвращает значение true , если объект *v* содержится в вызывающем стеке. В противном случае возвращает значение false |
| Clear() | Метод | Устанавливает свойство count равным нулю, тем самым эффективно очищая стек |
| Peek() | Метод | Возвращает элемент, расположенный в вершине стека, но не удаляет его |
| Pop() | Метод | Возвращает элемент, расположенный в вершине стека, и удаляет его |
| Push(object *v)* | Метод | Помещает объект *v* в стек |
| ToArray() | Метод | Возвращает массив, который содержит копии элементов вызывающего стека |
| Clone() | Метод | Поверхностное копирование элементов одной коллекции в другую |
| CopyTo(Array array, int arrayIndex) | Метод | Копирует элементы коллекции в массив |

Методы для коллекци **MyQueue.**

|  |  |  |
| --- | --- | --- |
| **Элемент** | **Вид** | **Назначение** |
| Capacity | Свойство | Количество элементов, которые могут  храниться в коллекции |
| Count | Свойство | Фактическое количество элементов коллекции |
| Contains (object *v)* | Метод | Возвращает значение true , если объект *v* содержится в вызывающей очереди. В противном случае возвращает значение false |
| Clear() | Метод | Устанавливает свойство Count равным нулю, тем самым эффективно очищая очередь |
| Dequeue () | Метод | Возвращает объект из начала вызывающей очереди, Возвращаемый объект из очереди удаляется |
| Enqueue(object *v)* | Метод | Добавляет объект *v* в конец очереди |
| Peek () | Метод | Возвращает объект из начала вызывающей очереди, но не удаляет его |
| ToArray() | Метод | Возвращает массив, который содержит копии элементов вызывающего стека |
| Clone() | Метод | Поверхностное копирование элементов одной коллекции в другую |
| CopyTo(Array array, int arrayIndex) | Метод | Копирует элементы коллекции в массив |

**Для всех коллекций реализовать конструкторы:**

* public MyCollection() - предназначен для создания пустой коллекции с заранее определенной начальной емкостью.
* public MyCollection (int capacity*)* - создает пустую коллекцию с начальной емкостью, заданной параметром capacity.
* public MyCollection (MyCollection c) - служит для создания очереди, которая инициализируется элементами и емкостью коллекции, заданной параметром с.

## Варианты

|  |  |  |
| --- | --- | --- |
|  | MyArray | Массив, динамически  изменяющий свой размер |
|  | MyHashtable | Хеш-таблица (пары элементов «ключ» - «значение») |
|  | MyQueue | Очередь (на базе списка) |
|  | MySortedList | Коллекция, отсортированная  по ключам (пары элементов «ключ» - «значение»). Доступ  к элементам — по ключу  или по индексу. |
|  | MyStack | Стек (на базе списка) |
|  | MyArray | Массив, динамически  изменяющий свой размер |
|  | MyHashtable | Хеш-таблица (пары элементов «ключ» - «значение») |
|  | MyMyQueue | Очередь (на базе списка) |
|  | MySortedList | Коллекция, отсортированная  по ключам (пары элементов «ключ» - «значение»). Доступ  к элементам — по ключу  или по индексу. |
|  | MyStack | Стек (на базе списка) |
|  | MyArray | Массив, динамически  изменяющий свой размер |
|  | MyHashtable | Хеш-таблица (пары элементов «ключ» - «значение») |
|  | MyQueue | Очередь (на базе списка) |
|  | MySortedList | Коллекция, отсортированная  по ключам (пары элементов «ключ» - «значение»). Доступ  к элементам — по ключу  или по индексу. |
|  | MyStack | Стек (на базе списка) |

## Часть 2. Пространство имен System.Collections.

## Теоретические сведения

Пространство имен System.Collections содержит множество интерфейсов и классов, которые определяют и реализуют коллекции различных типов. Коллекции упрощают программирование, предлагая уже готовые решения для построения структур данных, разработка которых "с нуля" отличается большой трудоемкостью. Речь идет о встроенных коллекциях, которые поддерживают, например, функционирование стеков, очередей и хеш-таблиц. Коллекции пользуются большой популярностью у всех С#-программистов.

В пространстве имен System.Collections определены наборы стандартных коллекций и интерфейсов, которые реализованы в этих коллекциях.

**Интерфейсы**

|  |  |
| --- | --- |
| Интерфейс | Назначение |
| ICollection | Определяет общие характеристики (например, размер) для набора элементов |
| IComparer | Позволяет сравнивать два объекта |
| IDictionary | Позволяет представлять содержимое объекта в виде пар «имя-­значение» |
| IDictionaryEnumenator | Используется для нумерации содержимого объекта, поддерживающего интерфейс IDictionary. |
| IEnumerable | Возвращает интерфейс IEnumerator для указанного объекта |
| IEnumerator | Обычно используется для поддержки оператора foreach в отношении объектов |
| IHashCodeProvider | Возвращает хеш-код для реализации типа с применением выбранного пользователем алгоритма хеширования |
| IList | Поддерживает методы добавления, удаления и индексирования элементов в списке объектов |

Основополагающим для всех коллекций является реализация **перечислителя** (нумератора), который поддерживается интерфейсами **IEnumerator и IEnumerable**. Перечислитель обеспечивает стандартизованный способ поэлементного доступа к содержимому коллекции. Поскольку каждая коллекция должна реализовать интерфейс IEnumerable, к элементам любого класса коллекции можно получить доступ с помощью методов, определенных в интерфейсе IEnumerator. Следовательно, после внесения небольших изменений код, который позволяет циклически опрашивать коллекцию одного типа, можно успешно использовать для циклического опроса коллекции другого типа. Содержимое коллекции любого типа можно опросить с помощью нумератора, используемого в цикле foreach.

**Интерфейс ICollection** можно назвать фундаментом, на котором построены все коллекции. В нем объявлены основные методы и свойства, без которых не может обойтись ни одна коллекция. Он наследует интерфейс IEnumerable.

ICollection включает в себя свойство Count, которое содержит количество элементов, хранимых в коллекции в данный момент. Если свойство Count равно нулю, значит, коллекция пуста.

Поскольку интерфейс ICollection наследует интерфейс IEnumerable, он также включает его единственный метод GetEnumerator ( ) :

IEnumerator GetEnumerator(), который возвращает нумератор коллекции.

**Интерфейс IList** наследует интерфейс ICollection и определяет поведение коллекции, доступ к элементам которой разрешен посредством индекса с отсчетом от нуля. Помимо методов, определенных в интерфейсе ICollection, интерфейс IList определяет и собственные методы.

**Методы интерфейса IList**

|  |  |
| --- | --- |
| Метод | Описание |
| int Add(object obj) | Добавляет объект objв вызывающую коллекцию. Возвращает индекс, по которому этот объект сохранен |
| void Clear() | Удаляет все элементы из вызывающей коллекции |
| bool Contains(object obj) | Возвращает значение true , если вызывающая коллекция содержит объект, переданный в параметре obj*,* и значение false в противном случае |
| int IndexOf(object obj) | Возвращает индекс объекта obj,если он (объект) содержится в вызывающей коллекции. Если объект obj не обнаружен, метод возвращает -1 |
| void Insert(int idx, object obj) | Вставляет в вызывающую коллекцию объект objпо индексу, заданному параметром idx.Элементы, находившиеся до этого по индексу idx и далее, смещаются вперед, чтобы освободить место для вставляемого объекта obj. |
| void Remove(object obj) | Удаляет из вызывающей коллекции объект, расположенный по индексу, заданному параметром idx. Элементы, находившиеся до этого за удаленным элементом, смещаются назад, чтобы ликвидировать образовавшуюся "брешь" |
| void RemoveAt(int idx) | Удаляет первое вхождение объекта objиз вызывающей коллекции. Элементы, находившиеся до этого за удаленным элементом, смещаются назад, чтобы ликвидировать образовавшуюся "брешь" |

В классе IList определены свойства:

* bool IsFixedSize { get; }
* bool IsReadOnly { get; }

Если коллекция имеет фиксированный размер, свойство IsFixedSize принимает значение true. Это означает, что в такую коллекцию нельзя вставлять элементы и удалять их из нее. Если коллекция предназначена только для чтения, свойство IsReadOnly имеет значение true. Это говорит о том, что содержимое коллекции изменению не подлежит.

**Интерфейс IDictionary** определяет поведение коллекции, которая устанавливает соответствие между уникальными ключами и значениями. Ключ - это объект, который используется для получения соответствующего ему значения. Следовательно, коллекция, которая реализует интерфейс IDictionary, служит для хранения пар ключ/значение. Сохраненную однажды пару можно затем извлечь по заданному ключу. Интерфейс IDictionary наследует интерфейс ICollection.

**Методы интерфейса IDictionary**

|  |  |
| --- | --- |
| Метод | Описание |
| void Add( object k, object v) | Добавляет в вызывающую коллекцию пару ключ/значение, заданную параметрами к и v. Ключ к не должен быть нулевым |
| void Clear() | Удаляет все пары ключ/значение из вызывающей коллекции |
| bool Contains (object k) | Возвращает значение true , если вызывающая коллекция содержит объект к в качестве ключа. В противном случае возвращает значение false |
| iDictionaryEnumerator GetEnumerator() | Возвращает нумератор для вызывающей коллекции |
| void Remove (object k) | Удаляет элемент, ключ которого равен значению k |

В интерфейсе IDictionary определены следующие свойства:

* bool isFixedSize {get ;} Равно значению true, если словарь имеет фиксированный размер.
* bool isReadOnly get; } Равно значению true , если словарь предназначен только для чтения.
* iCollectionKeys { get; } Получает коллекцию ключей.
* iCollection Values { get; } Получает коллекцию значений.

В пространстве имен System.Collections опреден тип структуры с именем DictionaryEntry. Коллекции, в которых хранятся пары ключ/значение, используют для их хранения объект типа DictionaryEntry. В этой структуре определены следующие два свойства:

* public object Key { get; set; }
* public object Value { get; set; }

Эти свойства используются для получения доступа к ключу или к соответствующему ему значению. Объект типа DictionaryEntry можно создать с помощью следующего конструктора:

public DictionaryEntry(object кey, object value)

Здесь параметр key принимает ключ, а параметр value — значение.

**Классы коллекций общего назначения:**

|  |  |  |
| --- | --- | --- |
| Класс | Назначение | Интерфейсы |
| ArrayList | Массив, динамически  изменяющий свой размер | IList, ICollection, IEnumerable, ICloneable |
| Hashtable | Хеш-таблица | IDictionary, ICollection, IEnumerable,  ICloneable |
| Queue | Очередь | ICollection, ICloneable, IEnumerable |
| SortedList | Коллекция, отсортированная  по ключам. Доступ  к элементам — по ключу  или по индексу | IDictionary, ICollection, IEnumerable,  ICloneable |
| Stack | Стек | ICollection, IEnumerable |

Класс ArrayList предназначен для поддержки динамических массивов, которые при необходимости могут увеличиваться или сокращаться. В С# стандартные массивы имеют фиксированную длину, которая не может измениться во время выполнения программы. Это означает, что программист должен знать заранее, сколько элементов будет храниться в массиве. Но иногда до выполнения программы нельзя точно сказать, массив какого размера понадобится. В таких случаях и используется класс ArrayList. Объект класса ArrayList представляет собой массив переменной длины, элементами которого являются объектные ссылки. Любой объект класса ArrayList создается с некоторым начальным размером. При превышении этого размера коллекция автоматически его увеличивает. В случае удаления объектов массив можно сократить.

Класс ArrayList реализует интерфейсы ICollection, IList, IEnumerable и ICloneable. В классе ArrayList определены следующие конструкторы:

* public ArrayList() - предназначен для создания пустого ArrayList-массива с начальной емкостью, равной 8 элементам.
* public ArrayList(ICollection с) - предназначен для создания массива, который инициализируется элементами и емкостью коллекции, заданной параметром с.
* public ArrayList(int capacity) - создает массив с заданным начальным размером.
* ArrayList arrl = new ArrayList(); // создается массив из 8 элементов
* ArrayLi st arr2 = new ArrayList(1000); // создается массив из 1000 элементов
* ArrayList arr3 = new ArrayList();
* arr3.Capacity = 1000; // количество элементов задается

**Основные элементы класса ArrayList**

|  |  |  |
| --- | --- | --- |
| Элемент | Вид | Назначение |
| Capacity | Свойство | Количество элементов, которые могут  храниться в массиве |
| Count | Свойство | Фактическое количество элементов массива |
| Item | Свойство | Получить или установить значение элемента по заданному индексу |
| Add | Метод | Добавление элемента в конец массива |
| AddRange | Метод | Добавление серии элементов в конец массива |
| BinarySearch | Метод | Двоичный поиск в отсортированном массиве или его части |
| Clear | Метод | Удаление всех элементов из массива |
| Clone | Метод | Поверхностное копирование элементов одного массива  в другой массив |
| CopyTo | Метод | Копирование всех или части элементов массива в одномерный массив |
| GetRange | Метод | Получение значений подмножества элементов массива в виде объекта типа ArrayList |
| Index | Метод | Поиск первого вхождения элемента в массив (возвращает индекс найденного элемента или -1, если элемент не найден) |
| Insert | Метод | Вставка элемента в заданную позицию (по заданному индексу) |
| InserRange | Метод | Вставка группы элементов, начиная с заданной позиции |
| LastIndexOf | Метод | Поиск последнего вхождения элемента в одномерный массив |
| Remove | Метод | Удаление первого вхождения заданного элемента в массив |
| RemoveAt | Метод | Удаление элемента из массива по заданному индексу |
| RemoveRange | Метод | Удаление группы элементов из массива |
| Reverse | Метод | Изменение порядка следования элементов на обратный |
| SetRange | Метод | Установка значений элементов массива в заданном диапазоне |
| Sort | Метод | Упорядочивание элементов массива или его части |
| TrimToSize | Метод | Установка емкости массива равной фактическому количеству  элементов |

class Program

{

static void Print(ArrayList al)

{

Console.WriteLine("Начальная емкость коллекции"+al.Capacity);

Console.WriteLine("Начальное количество элементов в коллекции" + al.Count);

Console.WriteLine();

}

static void PrintCollection(ArrayList al)

{

for (int i = 0; i < al.Count; i++)

Console.Write(al[i] + " ");

Console.WriteLine();

}

static void Main(string[] args)

{

ArrayList al = new ArrayList();//коллекция из 8 объектов

Print(al);

al.Add(1);

al.Add(2);

al.Add(3);

al.Add(4);

al.Add(5);

al.Add(6);

Print(al);

al.Remove(1);

al.Remove(5);

Print(al);

foreach (int x in al)

Console.Write(x + " ");

Console.WriteLine();

PrintCollection(al);

//добавление элементов

for (int i = 10; i < 20; i++)

al.Add(i);

Print(al);

//печать с помощью foreach

foreach (int x in al)

Console.Write(x + " ");

Console.WriteLine();

//изменение элементов

Console.WriteLine("Изменяем первые три элемента");

al[0] = 10;

al[1] = 20;

al[2] = 30;

PrintCollection(al);

//сортировка

al.Sort();

PrintCollection(al);

//бинарный поиск

int nom=al.BinarySearch(10);

Console.WriteLine("Номер элемента со значением 10 = " + nom);

}

}

**Класс Hashtable** предназначен для создания коллекции, в которой для хранения объектов используется хеш-таблица. Возможно, многим известно, что в хеш-таблице для хранения информации используется механизм, именуемый хешированием (hashing). Суть хеширования состоит в том, что для определения уникального значения, которое называется хеш-кодом, используется информационное содержимое соответствующего ему ключа. Хеш-код затем используется в качестве индекса, по которому в таблице отыскиваются данные, соответствующие этому ключу. Преобразование ключа в хеш-код выполняется автоматически, т.е. сам хеш-код вы даже не увидите. Но преимущество хеширования - в том, что оно позволяет сохранять постоянным время выполнения таких операций, как поиск, считывание и запись данных, даже для больших объемов информации. Класс Hashtable реализует интерфейсы IDictionary, ICollection, IEnumerable, ISerializable, IDeserializationCallback и ICloneable.

В классе Hashtable определено множество конструкторов, включая следующие:

* public Hashtable() - позволяет создать стандартный объект класса Hashtable.
* public Hashtable(IDictionary с) - для инициализации Hashtable-объекта используются элементы заданной коллекции с.
* public Hashtable(int capacity) - инициализирует емкость создаваемой хеш-таблицы значением capacity.
* public Hashtable(int capacity, float fillRatio) - инициализирует емкость значением capacity), а коэффициент заполнения значением fillRatio. Значение коэффициента заполнения (коэффициента нагрузки), которое должно попадать в диапазон 0,1-1,0, определяет степень заполнения хеш-таблицы, после чего ее размер увеличивается. В частности, размер таблицы увеличится, когда количество элементов станет больше емкости таблицы, умноженной на ее коэффициент заполнения.

**Основные элементы класса Hashtable**

|  |  |  |
| --- | --- | --- |
| Элемент | Вид | Назначение |
| Keys | Свойство | Получить коллекцию ключей |
| Values | Свойство | Получить коллекцию значений |
| ContainsKey | Метод | Возвращает true , если в вызывающей  коллекции содержится ключ, заданный параметром*.* В противном случае возвращает значение false. |
| ContainsValue | Метод | Возвращает true , если в вызывающей  коллекции содержится значение, заданное параметром*.* В противном случае возвращает значение false. |

static void Main(string[] args)

{

Hashtable ht = new Hashtable();

ht.Add("cat", "кошка");

ht.Add(" dog", "собака");

ht.Add("house", "дом");

ht.Add("table", "стол");

ht.Add("program", "программа");

ht.Add("student", "студент");

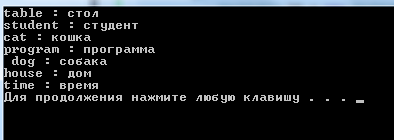
ht["time"] = "время";

ICollection c = ht.Keys;

foreach (string s in c)

Console.WriteLine(s +" : "+ ht[s]);

}



Как видно по приведенным результатам, пары ключ/значение хранятся не в упорядоченном виде. Обратите внимание на то, как было получено и отображено содержимое хеш-таблицы ht. Во-первых, коллекция ключей считывается с помощью свойства Keys. Каждый ключ затем используется в качестве индекса хеш-таблицы ht, который позволяет найти и отобразить значение, соответствующее каждому ключу.

**Класс SortedList** предназначен для создания коллекции, которая хранит пары ключ/значение в упорядоченном виде, а именно отсортированы по ключу. Класс SortedList реализует интерфейсы IDictionary, ICollection, IEnumerable и ICloneable.

В классе SortedList определено несколько конструкторов, включая следующие:

* public SortedList() - позволяет создать пустую коллекцию с начальной емкостью, равной 8 элементам.
* public SortedList(IDictionary c) - создает SortedList-коллекцию, которая инициализируется элементами и емкостью коллекции, заданной параметром с.
* public SortedList(int capacity) - создает пустой SortedList-список, который инициализируется емкостью, заданной параметром capacity.
* public SortedList(IComparer comp) - позволяет задать метод сравнения, который должен использоваться для сравнения объектов списка. С помощью этой формы создается пустая коллекция с начальной емкостью, равной 8 элементам.

Емкость SortedList-коллекции увеличивается автоматически, если в этом возникает необходимость, при добавлении элементов. Если окажется, что текущая емкость может быть превышена, она удваивается. Преимущество задания емкости при создании SortedList-списка состоит в минимизации затрат системных ресурсов, связанных с изменением размера коллекции. Конечно, задавать начальную емкость имеет смысл только в том случае, если вы знаете, какое количество элементов должно храниться в коллекции.

В классе SortedList помимо методов, определенных в реализованных им интерфейсах, также определены собственные методы.

|  |  |  |
| --- | --- | --- |
| Элемент | Вид | Назначение |
| Keys | Свойство | Получить коллекцию ключей |
| Values | Свойство | Получить коллекцию значений |
| ContainsKey | Метод | Возвращает значение true , если в коллекции содержится ключ, заданный параметром. В противном случае возвращает значение false |
| ContainsValue | Метод | Возвращает значение true , если в коллекции содержится значение, заданное параметром. В противном случае возвращает значение false |
| GetBylndex | Метод | Возвращает значение, индекс которого задан параметром |
| GetKey | Метод | Возвращает ключ, индекс которого задан параметром |
| GetKeyList() |  | Возвращает iList-коллекцию ключей, хранимых в вызывающей SortedList-коллекции |
| GetValueList() |  | Возвращает iList-коллекцию значений, хранимых в вызывающей SortedList-коллекции |
| IndexOfKe |  | Возвращает индекс ключа, заданного параметром к. Возвращает значение - 1 , если в списке нет заданного ключа |
| IndexOfValue |  | Возвращает индекс первого вхождения значения, заданного параметром v. Возвращает -1, если в списке нет заданного ключа |
| SetBylndex(int idx, object v) |  | Устанавливает значение по индексу, заданному параметром idx, равным значению, переданному в параметре v |
| TrimToSize() |  | Устанавливает свойство capacity равным значению свойства Count |

Подобно Наshtable -коллекции, SortedList-список хранит пары ключ/значение в форме структуры типа DictionaryEntry, но с помощью методов и свойств, определенных в классе SortedList, программисты обычно получают отдельный доступ к ключам и значениям.

Использование отсортированного списка типа SortedList демонстрируется в программе, которая представляет собой переработанную и расширенную версию программы из предыдущего раздела, демонстрировавшей Hashtable-коллекцию. Изучая результаты выполнения этой программы, обратите внимание на то, что SortedList-коллекция отсортирована по ключу.

static void Main(string[] args)

{

SortedList sl = new SortedList();

sl.Add("cat", "кошка");

sl.Add(" dog", "собака");

sl.Add("house", "дом");

sl.Add("table", "стол");

sl.Add("program", "программа");

sl.Add("student", "студент");

sl["time"] = "время";

ICollection c = sl.Keys;

foreach (string s in c)

Console.WriteLine(s + " : " + sl[s]);

Console.WriteLine("Список, полученый с помощью целочисленных индексаторов");

for (int i = 0; i < sl.Count; i++)

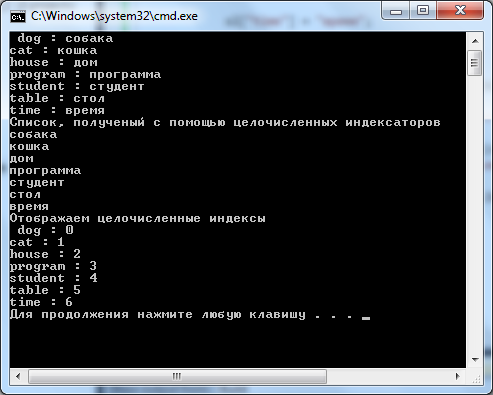
Console.WriteLine(sl.GetByIndex(i)+ " ");

Console.WriteLine("Отображаем целочисленные индексы");

foreach (string s in c)

Console.WriteLine(s + " : " + sl.IndexOfKey(s));

}



**Класс Stack** представляет собой список, добавление и удаление элементов к которому осуществляется по принципу "последним пришел — первым обслужен" (last-in, first-out, LIFO).

Класс коллекции, предназначенный для поддержки стека, называется Stack. Он реализует интерфейсы ICollection, IEnumerable и ICloneable. Стек — это динамическая коллекция, которая при необходимости увеличивается, чтобы принять для хранения новые элементы, причем каждый раз, когда стек должен расшириться, его емкость удваивается.

В классе Stack определены следующие конструкторы:

* public Stack() - предназначен для создания пустого стека с начальной емкостью, равной 10 элементам.
* public Stack(int capacity)- создает пустой стек с начальной емкостью, заданной параметром capacity
* public Stack(ICollection c) - служит для построения стека, который инициализируется элементами и емкостью коллекции, заданной параметром с.

В классе Stack помимо методов, определенных в реализованных им интерфейсах, также определены собственные методы.

|  |  |  |
| --- | --- | --- |
| Элемент | Вид | Назначение |
| Contains(object v) | Метод | Возвращает значение true , если объект v содержится в вызывающем стеке. В противном случае возвращает значение false |
| Clear() | Метод | Устанавливает свойство count равным нулю, тем самым эффективно очищая стек |
| Peek() | Метод | Возвращает элемент, расположенный в вершине стека, но не удаляет его |
| Pop() | Метод | Возвращает элемент, расположенный в вершине стека, и удаляет его |
| Push(object v) | Метод | Помещает объект v в стек |
| ToArray() | Метод | Возвращает массив, который содержит копии элементов вызывающего стека |

class Program

{

static void ShowPush(Stack st, int elem)

{

st.Push(elem);

Console.WriteLine("Помещаем элемент {0} в стек",elem);

Console.WriteLine("Стек:");

foreach(int x in st)

Console.Write(x+" ");

Console.WriteLine();

}

static void ShowPop(Stack st)

{

if (st.Count == 0) Console.WriteLine("Стек пуст");

else

{

int elem = (int)st.Pop();

Console.WriteLine("Удаляем элемент {0} из стекa", elem);

Console.WriteLine("Стек:");

foreach (int x in st)

Console.Write(x + " ");

Console.WriteLine();

}

}

static void Main(string[] args)

{

Stack st = new Stack();

Console.WriteLine("В стеке {0} элементов", st.Count);

Console.WriteLine();

ShowPush(st, 22);

ShowPush(st, 65);

ShowPush(st, 91);

ShowPop(st);

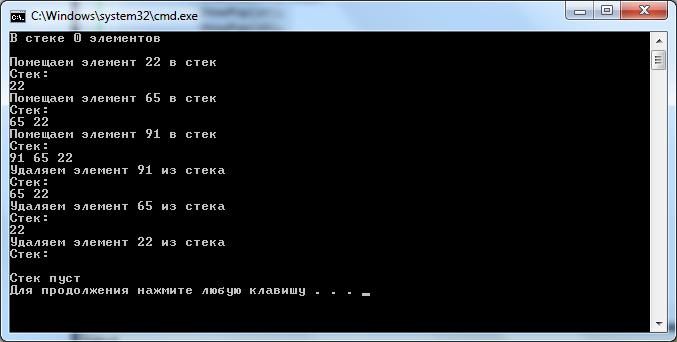
ShowPop(st);

ShowPop(st);

ShowPop(st);

}

}



Класс коллекции, предназначенный для поддержки очереди, называется **Queue.** Он реализует интерфейсы iCollection, IEnumerable и ICloneable. Очередь — это динамическая коллекция, которая при необходимости увеличивается, чтобы принять для хранения новые элементы, причем каждый раз, когда такая необходимость возникает, текущий размер очереди умножается на коэффициент роста, который по умолчанию равен значению 2,0.

В классе Queue определены следующие конструкторы:

* public Queue() - предназначен для создания пустой очереди с начальной ем-
* костью, равной 32 элементам, и коэффициентом роста 2,0.
* public Queue (int capacity) - создает пустую очередь с начальной емкостью, заданной параметром capacity, и коэффициентом роста 2,0.
* public Queue (int capacity, float growFact)- позволяет задать коэффициент роста посредством параметра growFact.
* public Queue (ICollection c) - служит для создания очереди, которая инициализируется элементами и емкостью коллекции, заданной параметром с.

Помимо методов, определенных в интерфейсах, которые реализует класс Queue, также определены собственные методы.

|  |  |  |
| --- | --- | --- |
| Элемент | Вид | Назначение |
| Contains (object v) | Метод | Возвращает значение true , если объект v содержится в вызывающей очереди. В противном случае возвращает значение false |
| Clear() | Метод | Устанавливает свойство Count равным нулю, тем самым эффективно очищая очередь |
| Dequeue () | Метод | Возвращает объект из начала вызывающей очереди, Возвращаемый объект из очереди удаляется |
| Enqueue(object v) | Метод | Добавляет объект v в конец очереди |
| Peek () | Метод | Возвращает объект из начала вызывающей очереди, но не удаляет его |
| тоАггау () | Метод | Возвращает массив, который содержит копии элементов из вызывающей очереди |
| TrimToSize() | Метод | Устанавливает свойство Capacity равным значению свойства Count |

class Program

{

static void ShowEnq(Queue q, int elem)

{

q.Enqueue(elem);

Console.WriteLine("Помещаем элемент {0} в очередь", elem);

Console.WriteLine("Очередь:");

foreach (int x in q)

Console.Write(x + " ");

Console.WriteLine();

}

static void ShowDeq(Queue q)

{

if (q.Count == 0) Console.WriteLine("Очередь пустая");

else

{

int elem=(int) q.Dequeue();

Console.WriteLine("Удаляем элемент {0} из очереди", elem);

Console.WriteLine("Очередь:");

foreach (int x in q)

Console.Write(x + " ");

Console.WriteLine();

}

}

static void Main(string[] args)

{

Queue q = new Queue();

Console.WriteLine("В очереди {0} элементов", q.Count);

Console.WriteLine();

ShowEnq(q, 22);

ShowEnq(q, 65);

ShowEnq(q, 91);

ShowDeq(q);

ShowDeq(q);

ShowDeq(q);

ShowDeq(q);

}

}

**Доступ к коллекциям с помощью нумератора**

Часто при работе с коллекциями возникает необходимость в циклическом опросе ее элементов. Например, нужно отобразить все элементы коллекции. Один из способов достижения этого — использование цикла foreach. Еще один способ — использование нумератора.

**Нумератор** представляет собой объект, который реализует интерфейс IEnumerator. В интерфейсе IEnumerator определено единственное свойство:

object Current { get; }

Свойство Current позволяет получить элемент, соответствующий текущему значению нумератора. Поскольку свойство Current предназначено только для чтения, нумератор можно использовать только для считывания значения объекта в коллекции, а не для его модификации.

В интерфейсе IEnumerator определены два метода.

* bool MoveNext() - при каждом обращении к методу MoveNext () текущая позиция нумератора перемещается к следующему элементу коллекции. Метод возвращает значение true , если к следующему элементу можно получить доступ, или значение false, если достигнут конец коллекции. До выполнения первого обращения к методу MoveNext () значение свойства Current неопределено.
* void Reset() - устанавливает нумератор в начало коллекции. После вызова метода Reset() нумерация элементов начнется с начала коллекции, и для доступа к первому ее элементу необходимо вызвать метод MoveNext ().

Класс коллекции, который реализует интерфейс IDictionary, предназначен для хранения пар ключ/значение. Для опроса элементов в такой коллекции используется интерфейс IDictionaryEnumerator, а не IEnumerator. Класс DictionaryEnumerator является производным от класса IEnumerator и дополнительно определяет три свойства.

* DictionaryEntry Entry { get; }- позволяет получить следующую пару ключ/значение в форме структуры типа DictionaryEntry.
* object Key { get; } - позволяет получить прямой доступ к ключу.
* object Value { get; }- позволяет получить прямой доступ к значению.

Интерфейс IDictionaryEnumerator используется подобно обычному нумератору за исключением того, что текущие значения элементов здесь можно получить с помощью свойств Entry, Key или Value, а не свойства Current. Таким образом, реализовав IDictionaryEnumerator-нумератор, вы должны вызвать метод MoveNext(), чтобы получить первый элемент. Остальные элементы коллекции опрашиваются путем последующих вызовов метода MoveNext (). Когда все элементы коллекции будут исчерпаны, метод MoveNext () возвратит значение false.

static void Main(string[] args)

{

ArrayList al = new ArrayList(1);

for (int i = 0; i < 10; i++)

al.Add(i);

IEnumerator ienum = al.GetEnumerator();

while (ienum.MoveNext())

Console.Write(ienum.Current + " ");

Console.WriteLine();

//еще раз

ienum.Reset();

while (ienum.MoveNext())

Console.Write(ienum.Current + " ");

Console.WriteLine();

Hashtable ht = new Hashtable();

ht.Add("Иванов И.И.", "222-22-22");

ht.Add("Сидоров С.С.", "233-33-33");

ht.Add("Петров П.П.", "244-44-44");

IDictionaryEnumerator etr = ht.GetEnumerator();

Console.WriteLine("Отображаем информацию с помощью свойства Entry.");

while (etr.MoveNext())

Console.WriteLine(etr.Entry.Key + ": " + etr.Entry.Value);

Console.WriteLine();

Console.WriteLine("Отображаем информацию с помощью " +"свойств Key и Value.");

etr.Reset();

while (etr.MoveNext())

Console.WriteLine(etr.Key + ": " +etr.Value);

}

## 2. Постановка задачи

1. Создать коллекцию, в которую добавить объекты созданной иерархии классов.
2. Используя меню, реализовать в программе добавление, удаление объектов, поиск объекта по ключу, сортировку.
3. Разработать и реализовать три запроса (количество элементов определенного вида, печать элементов определенного вида и т.п.).
4. Реализовать нумератор и организовать с его помощью перебор коллекции.

## Часть 3. Обобщенные коллекции. Пространство имен System.Collections.Generic.

## Теоретические сведения

Многие алгоритмы не зависят от типов данных, с которыми они работают. Простейшими примерами могут служить сортировка и поиск. Возможность отделить алгоритмы от типов данных предоставляют *классы-прототипы* (generics) — классы, имеющие в качестве параметров типы данных. Чаще всего эти классы применяются для хранения данных, то есть в качестве контейнерных классов, или коллекций.

Начиная с версии .NET 2.0, язык программирования С# был расширен поддержкой средства, которое называется ***обобщением (generic)***. Вместе с ним библиотеки базовых классов пополнились совершенно новым пространством имен, связанным с коллекциями — System.Collections.Generic.

Общая форма объявления обобщенного класса:

*class имя\_класса<список\_параметров\_типа> {...}*

Ссылка на обобщенный класс:

*имя\_класса<список\_аргументов\_типа> имя\_переменной =*

*new имя\_класса<список\_параметров\_типа> (список\_аргументов\_конструктора);*

Пример 1.

namespace ConsoleApplication1

{

namespace ConsoleApplication1

{

// Создадим обобщенный класс имеющий поле типа T

class MyObj<T>

{

T obj;

public MyObj(T obj)

{

this.obj = obj;

}

public void objectType()

{

Console.WriteLine("Тип объекта: " + typeof(T));

}

}

// Обобщенный класс с несколькими полями

class MyObjects<T, V, E>

{

T obj1;

V obj2;

E obj3;

public MyObjects(T obj1, V obj2, E obj3)

{

this.obj1 = obj1;

this.obj2 = obj2;

this.obj3 = obj3;

}

public void objectsType()

{

Console.WriteLine("\nТип объекта 1: " + typeof(T) +

"\nТип объекта 2: " + typeof(V) +

"\nТип объекта 3: " + typeof(E));

}

}

class Program

{

static void Main()

{

// Создадим экземпляр обобщенного класса типа int

MyObj<int> obj1 = new MyObj<int>(25);

obj1.objectType();

MyObjects<string, double, int> obj2 = new MyObjects<string, double, int>("Alex", 26, 1200);

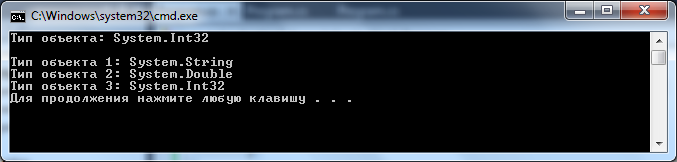
obj2.objectsType();

}

}

}

}



Если в программе используются обобщения, то очень полезно, когда переменные обобщенных типов легко можно отличить от необобщенных.

Рекомендации по именованию обобщенных типов:

* Имена обобщенных типов должны начинаться с буквы Т.
* Если обобщенный тип может быть заменен любым классом, поскольку нет никаких специальных требований, и используется только один обобщенный тип, Т — вполне подходящее имя для обобщенного типа:

public class List<T> { }

public class LinkedList<T> { }

* Если к обобщенному типу предъявляются специальные требования (например, что тип должен реализовывать интерфейс или наследоваться от определенного класса), либо же используется два или более обобщенных типа в качестве параметров, то следует применять осмысленные имена типов:

public class SortedList<TKey, TValue> { }

Во вторую версию библиотеки .NET добавлены параметризованные коллекции для представления основных структур данных, применяющихся при создании программ, — стека, очереди, списка, словаря и т. д. Эти коллекции, расположенные в пространстве имен **System.Collections.Generic**, дублируют аналогичные коллекции пространства имен **System.Collections.**

**Параметризованные коллекции библиотеки .NET**

|  |  |
| --- | --- |
| Класс- прототип | Обычный класс |
| Comparer <T> | Comparer |
| Dictionary <K,T> | HashTable |
| LinkedList <T> | - |
| List <T> | ArrayList |
| Queue<T> | Queue |
| SortedDictionary <K,T> | SortedList |
| Stack <T> | Stack |

**Параметризованные интерфейсы библиотеки .NET.**

|  |  |
| --- | --- |
| Параметризованный интерфейс | Обычный интерфейс |
| ICollection<T> | ICollection |
| IComparable<T> | IComparable |
| IDictionary<Т> | IDictionary |
| IEnumerable<T> | I Enumerable |
| IEnumerator<T> | I Enumerator |
| IList<T> | IList |

Параметром класса-прототипа является тип данных, с которым он работает (T – это тип, который является параметром коллекции, т. е. вместо него можно подставить любой другой тип данных).

Пример.

class Person : IComparable

{. . . . }

class Student : Person

{. . . . }

class Program

{

static void Main(string[] args)

{

List<Person> StudentGroup = new List<Person>();//создали коллекцию

StudentGroup.Add(new Student("Иванов",18,1,4));

StudentGroup.Add(new Student("Петров",19,2,5));

StudentGroup.Add(new Student("Сидоров",21,5,4.5));

StudentGroup.Add(new Person("Семенов",23));

//StudentGroup.Add(1);

Console.WriteLine("Студенческая группа");

foreach (Person x in StudentGroup)

x.Show();

List<int> Vector = new List<int>();

Vector.Add(1);

Vector.Add(5);

Vector.Add(4);

Console.WriteLine("\nВектор");

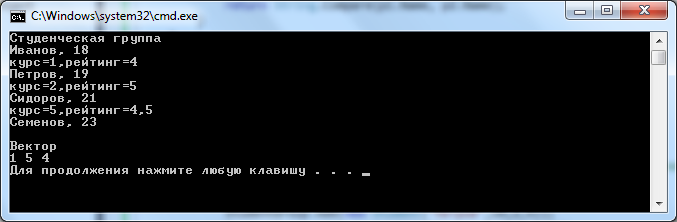
foreach (int y in Vector)

Console.Write(y + " ");

Console.WriteLine();

}

}



Коллекция StudentGroup содержит объекты пользовательских классов иерархии классов Person->Student.

В коллекции, для которой объявлен тип элементов Person, благодаря полиморфизму можно хранить элементы любого производного класса, но не элементы других типов.

Казалось бы, по сравнению с обычными коллекциями это ограничение, а не универсальность, однако на практике коллекции, в которых действительно требуется хранить значения различных, не связанных межу собой типов, почти не используются.

Достоинством же такого ограничения является то, что компилятор может выполнить контроль типов во время компиляции, а не выполнения программы, что повышает ее надежность и упрощает поиск ошибок.

## 2. Постановка задачи

1. Создать **параметризированную** коллекцию, в которую добавить объекты созданной иерархии классов.
2. Используя меню, реализовать в программе добавление, удаление объектов, поиск объекта по ключу, сортировку.
3. Разработать и реализовать три запроса (количество элементов определенного вида, печать элементов определенного вида и т.п.).
4. Реализовать нумератор и организовать с его помощью перебор коллекции.