# Практическая работа № 11. Стандарные коллекции .Net

**Цели:**

Получить практические навыки работы со стандартными универсальными коллекциями пространства имен System.Collection.

* Получить практические навыки работы со стандартными параметризированными коллекциями пространства имен System.Collection.Generic.
* Определить наиболее эффективные коллекции для решения задачи поиска.

## Теоретические сведения.

### Коллекции

**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.

**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.

cat

dog

table

door

кошка

собака

стол

дверь

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.

### Универсальные коллекции

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 |
| 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.
* 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.

Например:

* ArrayList arrl = new ArrayList(); // создается пустой массив
* ArrayLi st arr2 = new ArrayList(1000); // создается массив из 1000 элементов
* ArrayList arr3 = new ArrayList();//создается пустой массив
* 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 |

**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 пустой 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, которое должно попадать в диапазон 0,1-1,0, определяет степень заполнения хеш-таблицы. Например, если указать коэффициент заполнения 0,7, то таблица будет заполнена на 70%. После заполнения таблицы на 70% при добавлении в нее элемента, ее размер увеличивается в 2 раза.

**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. |

**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.
* 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 при их добавлении в коллекцию. С помощью этой формы создается пустая коллекция.

Емкость SortedList-коллекции увеличивается автоматически, если в этом возникает необходимость, при добавлении элементов. Если окажется, что текущая емкость может быть превышена, она удваивается. 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.

**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.
* 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 |

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 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 |

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

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, расположенны в пространстве имен **System.Collections.Generic** и, в основном, 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 |

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).

В коллекциях, которые мы рассматривали ранее, хранятся ссылки на объекты типа object. Когда значение структурного типа приводится к типу object (или к типу того интерфейса, который реализован системой) выполняется операция упаковки – явного преобразования из типа значений в тип ссылок. Эта операция выполняется автоматически и не требует вмешательства программиста. Обратной операцией является распаковка – значение объекта присваивается переменной.

У таких коллекций есть два недостатка:

* в одной и той же коллекции можно хранить элементы любого типа, следовательно, ошибки при помещении в коллекцию невозможно проконтролировать на этапе компиляции, а при извлечении элемента требуется его явное преобразование;
* при хранении в коллекции элементов значимых типов выполняется большой объем действий по упаковке и распаковке элементов, что в значительной степени снижает эффективность работы.

**Пример 1.**

class Person : IComparable

{. . . . }

class Student : Person

{. . . . }

class Program

{

static void Main(string[] args)

{

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

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

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

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

Group.Add(new Person("Семенов",23));

Console.WriteLine("Студенческая группа");

foreach (Person x in Group)

x.Show();

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

Vector.Add(1);

Vector.Add(5);

Vector.Add(4);

Console.WriteLine("\nВектор");

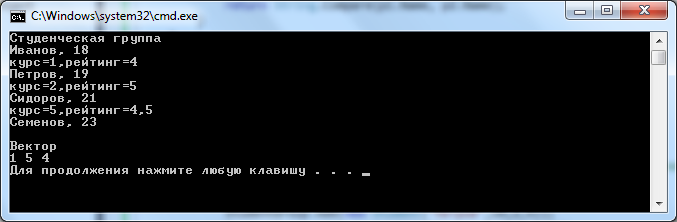
foreach (int y in Vector)

Console.Write(y + " ");

Console.WriteLine();

}

}



Коллекция Group 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.

Коллекция Vector состоит из целых чисел, причем для работы с ними не требуются ни операции упаковки и распаковки, ни явные преобразования типа при получении элемента из коллекции, как это было в обычных коллекциях.

Классы-прототипы называют также родовыми или шаблонными, поскольку они представляют собой образцы, по которым во время выполнения программы строятся конкретные классы. Использование стандартных параметризованных коллекций для хранения и обработки данных является хорошим стилем программирования, поскольку позволяет сократить сроки разработки программ и повысить их надежность. Рекомендуется тщательно изучить по документации свойства и методы этих классов и выбирать наиболее подходящие в зависимости от решаемой задачи.

**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 |

## 2. Постановка задачи

### 2.1. Задание 1.

* 1. Создать универсальную коллекцию, добавить в нее объекты из созданной иерархии классов (см. лаб. раб. 10).
  2. Реализовать в программе добавление и удаление объектов коллекции.
  3. Разработать и реализовать три запроса (количество элементов определенного вида, печать элементов определенного вида и т.п.), можно взять запросы из лаб. раб. 10.
  4. Выполнить перебор элементов коллекции с помощью метода foreach.
  5. Выполнить клонирование коллекции.
  6. Выполнить сортировку коллекции (если коллекция не отсортирована) и поиск заданного элемента в коллекции.

### 2.2. Задание 2.

* 1. Создать обобщенную коллекцию, добавить в нее объекты созданной иерархии классов (см. лаб. раб. 10).
  2. Реализовать в программе добавление и удаление объектов коллекции.
  3. Разработать и реализовать три запроса (количество элементов определенного вида, печать элементов определенного вида и т.п.), можно взять запросы из лаб. раб. 10..
  4. Выполнить перебор элементов коллекции с помощью метода foreach.
  5. Выполнить клонирование коллекции.
  6. Выполнить сортировку коллекции (если коллекция не отсортирована) и поиск заданного элемента в коллекции.

### 2.3. Задание 3.

1. Для выполнения задания нужно выбрать 2 класса из иерархии, созданной в 10 лаб. работе (Базовый -> Производный)
2. В производном классе определить свойство или метод, которое возвращает ссылку на объект базового класса (это свойство должно возвращать объект базового класса). Например, для иерархии классов Person->Student в классе производном классе Student можно определить свойство

public Person GetBase

{

get=> new Person(name, age);//возвращает объект базового класса

}

1. Определить класс TestCollections, который содержит поля следующих типов:

* Коллекция1 < TValue > ;
* Коллекция1 <string> ;
* Коллекция2 <TKey, TValue> или Коллекция2 < TKey >;
* Коллекция2 <string, TValue> или Коллекция2 <string>;

где тип ключа TKey и тип значения TValue связаны отношением Базовый - > Производный (см. задание 1), Коллекция1 и Коллекция2 – коллекции из пространства имен System.Collections.Generic, тип коллекции указан в варианте.

Например:

List<Student> col1=new List<Student>();

List<string> col2=new List<string>();

Dictionary<Person, Student> col3=new Dictionary<Person, Student>();

Dictionary<string, Student> col4=new Dictionary<string, Student>();

1. Написать конструктор класса TestCollections, в котором создаются коллекции с заданным числом элементов (1000 элементов).
2. Предусмотреть автоматическую генерацию элементов коллекции таким образом, что каждый объект (Student) содержит подобъект базового класса (Person). Все четыре коллекции должны содержать одинаковое число элементов. Сами элементы также должны быть одинаковые, т.е. сгенерировали элемент производного класса и на его основе заполнили элементами все 4 коллекции. В коллекциях, где должны быть строки, для заполнения используется метод ToString().
3. Для четырех разных элементов – первого, центрального, последнего и элемента, не входящего в коллекцию – надо измерить время поиска элемента в коллекциях. Для коллекций, состоящих из пар используются методы ContainsKey() и ContainsValue(), для остальных коллекций используется метод Contains().
4. **Обратите внимание на то, что искать нужно сами элементы, а не ссылки на них!**
5. Для получения более правильных измерений лучше измерить время поиска несколько раз и найти среднее значение.
6. Объяснить полученные результаты.

## 3. Варианты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № варианта | Задание 1 | Задание 2 | | Задание 3 (Коллекция1) | Задание 3 (Коллекция2) |
|  | Oueue | Dictionary<K,T> | | List <T> | SortedDictionary <K,T> |
|  | ArrayList | Stack<T> | | LinkedList <T> | Dictionary <K,T> |
|  | Stack | SortedDictionary<K,T> | | SortedSet<T> | HashSet<T> |
|  | SortedList | List<T> | | Queue<T> | SortedSet<T> |
|  | Hashtable | Oueue<T> | | Stack <T> | Dictionary <K,T> |
|  | ArrayList | Dictionary<K,T> | | LinkedList <T> | SortedDictionary <K,T> |
|  | Stack | List<T> | | Dictionary <K,T> | SortedSet<T> |
|  | Hashtable | SortedDictionary<K,T> | | LinkedList <T> | HashSet<T> |
|  | Oueue | Stack<T> | | Queue<T> | Dictionary <K,T> |
|  | SortedList | Dictionary<K,T> | | Stack <T> | SortedDictionary <K,T> |
|  | ArrayList | Oueue<T> | | SortedDictionary <K,T> | HashSet<T> |
|  | SortedList | Stack<T> | | LinkedList <T> | SortedSet<T> |
|  | Stack | SortedDictionary<K,T> | | HashSet<T> | Dictionary <K,T> |
|  | Oueue | List<T> | | Queue<T> | SortedDictionary <K,T> |
|  | Hashtable | SortedSet<T> | | Stack <T> | HashSet<T> |
|  | SortedList | HashSet<T> | | List <T> | SortedSet<T> |
|  | ArrayList | SortedDictionary <K,T> | | LinkedList <T> | SortedDictionary <K,T> |
|  | Stack | Dictionary <K,T> | | SortedDictionary <K,T> | Dictionary <K,T> |
|  | SortedList | SortedDictionary <K,T> | | Queue<T> | HashSet<T> |
|  | Hashtable | Dictionary <K,T> | | Stack <T> | SortedSet<T> |
|  | ArrayList | LinkedList <T> | | Dictionary <K,T> | SortedDictionary <K,T> |
|  | Stack | Dictionary <K,T> | | LinkedList <T> | Dictionary <K,T> |
|  | Hashtable | HashSet<T> | List <T> | | HashSet<T> |
|  | Oueue | SortedSet<T> | Queue<T> | | SortedSet<T> |
|  | SortedList | Dictionary <K,T> | Stack <T> | | SortedDictionary <K,T> |
|  | Hashtable | SortedDictionary <K,T> | List <T> | | Dictionary <K,T> |
|  | ArrayList | SortedDictionary <K,T> | LinkedList <T> | | HashSet<T> |
|  | SortedList | Dictionary <K,T> | List <T> | | SortedSet<T> |
|  | Oueue | SortedDictionary <K,T> | Queue<T> | | SortedDictionary <K,T> |
|  | ArrayList | Dictionary <K,T> | Stack <T> | | Dictionary <K,T> |

## 4. Содержание отчета

**Для заданий 11.1 и 11.2**

1. Диаграмма классов (в том числе иерархия классов для объектов, помещаемых в коллекцию).
2. Описание методов, используемых для работы с коллекцией.
3. Код программы или ссылка на гитхаб.

**Для задания 11.3**

1. Диаграмма классов (в том числе иерархия классов для объектов, помещаемых в коллекцию).
2. Код программы или ссылка на гитхаб.
3. Полученные результаты в виде таблицы.
4. Анализ полученных результатов (объяснить, какая из рассмотренных коллекций оказалась самой быстродействующей, и почему).

## Методические указания

1. Для измерения времени поиска рекомендуется использовать класс Stopwatch из библиотеки System.Diagnostics. Время лучше измерять в тиках, незначащие нули не показывать.
2. При поиске элемента обратите внимание на то, что искать нужно сами элементы (содержимое), а не ссылки на них.

Например:

//запомнили в first ссылку на первый элемент

Person first = collection1.First;

//запомнили в first содержимое первого элемента

Person first = new Person(ts.collection1.First.Name, ts.collection1.First.Age) ;

или

Person first= collection1.First.Clone();

1. Обязательно проверьте, был ли найден элемент, т.е. какое значение вернул метод Contains().
2. Поверьте, все ли необходимые методы реализованы в классах, объекты которых записываются в коллекции.
3. Проверьте, можно ли сгенерировать с помощью метода RandomInit() (лаб. работа №10) 1000 различных значений.

## Вопросы для защиты лабораторной работы

1. Что такое абстрактный тип данных? Привести примеры.
2. Чем абстрактный тип данных отличается от структуры данных?
3. Чем обобщенные коллекции отличаются от необобщенных?
4. В чем заключаются преимущества обобщенных коллекций?
5. Какие методы нужно перегрузить в своем классе для правильной работы коллекции Dictionary<TKey, TValue> / HashSet<T>?
6. Какие методы нужно перегрузить в своем классе для правильной работы коллекции SortedList<TKey, TValue> / SortedSet<T>?
7. Какие методы нужно перегрузить в своем классе для правильной работы коллекции List<T>, Stack<T>, Queue<T>, LinkedList<T>?
8. В какой коллекции быстрее будет выполнен поиск: Dictionary<TKey, TValue> или List<T> и почему?
9. В какой коллекции быстрее будет выполнен поиск SortedList<TKey, TValue> или List<T> и почему?
10. В какой коллекции быстрее будет выполнен поиск SortedList<TKey, TValue> или Dictionary<TKey, TValue> и почему?

## Критерии оценивания работы

|  |  |  |
| --- | --- | --- |
| № | Критерий | Баллы |
| 1 | Решены задачи 1-ой части работы:  продемонстрирована работа со стандартными универсальными коллекциями | 1 |
| 2. | Решены задачи 2-ой части работы:  продемонстрирована работа со стандартными обобщенными коллекциями | 1 |
| 3. | Выполнен поиск элементов и измерено время поиска в соответствии с заданием 3 части работы | 2 |
| 4 | Приводится объяснение результатов | 1-2 |
| 5 | Даны полные ответы на теоретические вопросы (п.5) | 1 |
| 6 | Написанный код соответствует стиль-гайду | 1 |
| 7 | Дополнительные баллы | не предусмотрены |