№ 10 Коллекции

Задание

1. Создать *необобщенную* коллекцию **ArrayList.**
   a. Заполните ее 5-ю случайными целыми числами
   b. Добавьте к ней строку
   c. Добавьте объект типа Student
   d. Удалите заданный элемент
   e. Выведите количество элементов и коллекцию на консоль.
   f. Выполните поиск в коллекции значения
2. Создать **обобщенную коллекцию** в соответствии с вариантом задания и заполнить ее данными, тип которых определяется вариантом задания (колонка – *первый тип)*.
   a. Вывести коллекцию на консоль
   b. Удалите из коллекции n последовательных элементов
   c. Добавьте другие элементы (используйте все возможные методы добавления для вашего типа коллекции).
   d. Создайте *вторую коллекцию* (см. таблицу) и заполните ее данными из первой коллекции.
   e. Выведите вторую коллекцию на консоль. В случае не совпадения количества параметров (например, *LinkedList<T>* и *Dictionary<Tkey, TValue>*), при нехватке - генерируйте ключи, в случае избыточности – оставляйте *TValue*.
   f. Найдите во второй коллекции заданное значение.

№	Первая коллекция	Первый тип	Вторая коллекция
1	Stack<T>	char	List<T>
2	Queue<T>	int	Dictionary<Tkey, TValue>
3	HashSet<T>	long	LinkedList<T>
4	List<T>	float	Stack<T>
5	Dictionary<Tkey, TValue>	double	Queue<T>
6	LinkedList<T>	char	HashSet<T>
7	SortedDictionary<TKey, TValue>	string	List<T>
8	SortedList<TKey, TValue>	long	Stack<T>
9	SortedSet<T>	float	Queue<T>
10	Dictionary<Tkey, TValue>	int	HashSet<T>
11	SortedList<TKey, TValue>	char	List<T>
12	Stack<T>	double	LinkedList<T>
13	Queue<T>	int	SortedDictionary<TKey, TValue>
14	HashSet<T>	long	SortedList<TKey, TValue>
15	List<T>	long	SortedSet<T>
16	Dictionary<Tkey, TValue>	string	Stack<T>
17	LinkedList<T>	double	SortedList<TKey, TValue>

18	SortedDictionary<TKey, TValue>	char	LinkedList<T>
19	SortedList<TKey, TValue>	int	Stack<T>
20	Stack<T>	int	SortedList<TKey, TValue>
21	SortedSet<T>	int	Dictionary<Tkey, TValue>
22	Dictionary<Tkey, TValue>	long	List<T>
23	SortedList<TKey, TValue>	float	Stack<T>
24	List<T>	char	Dictionary<Tkey, TValue>
25	Dictionary<Tkey, TValue>	bool	List<T>
26	LinkedList<T>	bool	Dictionary<Tkey, TValue>

3. Повторите задание п.2 для **пользовательского типа данных** (в качестве типа T возьмите любой свой класс из лабораторной *№5 (Наследование*….). Не забывайте о необходимости реализации интерфейсов (IComparable, ICompare,….). При выводе коллекции используйте цикл foreach.
4. Создайте объект *наблюдаемой коллекции* **ObservableCollection<T>**. Создайте произвольный метод и зарегистрируйте его на событие CollectionChange. Напишите демонстрацию с добавлением и удалением элементов. В качестве типа *T* используйте свой класс из лабораторной *№5 Наследование*….

Вопросы

1. Перечислите стандартные коллекции NET Framework.
2. Поясните принцип работы коллекции:

    a) Stack<T>

    b) Queue<T>

    c) HashSet<T>

    d) List<T>

    e) Dictionary<Tkey, TValue>

    f) LinkedList<T>

    g) SortedDictionary<TKey, TValue>

    h) SortedList<TKey, TValue>

    i) SortedSet<T>

3. Охарактеризуйте необобщенные, специальные, с поразрядной организацией, обобщенные и параллельные коллекции.
4. Какие интерфейсы используются в коллекциях C#?
5. Для чего используется интерфейс IComparable?

6. Что содержит интерфейс IEnumerator или обобщенный интерфейс IEnumerator<T>? Где и как его можно использовать?
7. Что такое наблюдаемая коллекция? Где и каким образом ее можно использовать?

Обзор коллекций

В C# **коллекция** представляет собой совокупность объектов. В среде .NET Framework имеется немало интерфейсов и классов, в которых определяются и реализуются различные типы коллекций.

Главное преимущество коллекций заключается в том, что они стандартизируют обработку групп объектов в программе. Все коллекции разработаны на основе набора четко определенных интерфейсов. Некоторые встроенные реализации таких интерфейсов, в том числе ArrayList, Hashtable, Stack и Queue, могут применяться в исходном виде и без каких-либо изменений. Имеется также возможность реализовать собственную коллекцию, хотя потребность в этом возникает редко.

В среде .NET Framework поддерживаются пять типов коллекций: необобщенные, специальные, с поразрядной организацией, обобщенные и параллельные.

Необобщенные коллекции

Реализуют ряд основных структур данных, включая динамический массив, стек, очередь, а также словари, в которых можно хранить пары "ключ-значение". В отношении необобщенных коллекций важно иметь в виду следующее: они оперируют данными типа object. Таким образом, необобщенные коллекции могут служить для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных. Очевидно, что такие коллекции не типизированы, поскольку в них хранятся ссылки на данные типа object. Классы и интерфейсы необобщенных коллекций находятся в пространстве имен **System.Collections**.

Специальные коллекции

Оперируют данными конкретного типа или же делают это каким-то особым образом. Например, имеются специальные коллекции для символьных строк, а также специальные коллекции, в которых используется однонаправленный список. Специальные коллекции объявляются в пространстве имен **System.Collections.Specialized**.

Поразрядная коллекция

В прикладном интерфейсе Collections API определена одна коллекция с поразрядной организацией — это BitArray. Коллекция типа BitArray поддерживает поразрядные операции, т.е. операции над отдельными двоичными разрядами, например И, ИЛИ, исключающее ИЛИ, а следовательно, она существенно отличается своими возможностями от остальных типов коллекций. Коллекция типа BitArray объявляется в пространстве имен System.Collections.

Обобщенные коллекции

Обеспечивают обобщенную реализацию нескольких стандартных структур данных, включая связные списки, стеки, очереди и словари. Такие коллекции являются типизированными в силу их обобщенного характера. Это означает, что в обобщенной коллекции могут храниться только такие элементы данных, которые

совместимы по типу с данной коллекцией. Благодаря этому исключается случайное несовпадение типов. Обобщенные коллекции объявляются в пространстве имен**System.Collections.Generic**.

Параллельные коллекции

Поддерживают многопоточный доступ к коллекции. Это обобщенные коллекции, определенные в пространстве имен **System.Collections.Concurrent**.

В пространстве имен System.Collections.ObjectModel находится также ряд классов, поддерживающих создание пользователями собственных обобщенных коллекций.

Основополагающим для всех коллекций является понятие *перечислителя*, который поддерживается в необобщенных интерфейсах IEnumerator и IEnumerable, а также в обобщенных интерфейсах IEnumerator<T> и IEnumerable<T>. Перечислитель обеспечивает стандартный способ поочередного доступа к элементам коллекции. Следовательно, он перечисляет содержимое коллекции. В каждой коллекции должна быть реализована обобщенная или необобщенная форма интерфейса IEnumerable, поэтому элементы любого класса коллекции должны быть доступны посредством методов, определенных в интерфейсе IEnumerator или IEnumerator<T>. Это означает, что, внеся минимальные изменения в код циклического обращения к коллекции одного типа, его можно использовать для аналогичного обращения к коллекции другого типа. Любопытно, что для поочередного обращения к содержимому коллекции в цикле foreach используется перечислитель.

С перечислителем непосредственно связано другое средство, называемое *итератором*. Это средство упрощает процесс создания классов коллекций, например специальных, поочередное обращение к которым организуется в цикле foreach.

Классы коллекций по своей сути подобны классам стандартной библиотеки шаблонов (Standard Template Library — STL), определенной в C++. То, что в программировании на C++ называется контейнером, в программировании на C# называется коллекцией.

Интерфейсы обобщенных коллекций

В пространстве имен System.Collections.Generic определен целый ряд интерфейсов обобщенных коллекций, имеющих соответствующие аналоги среди интерфейсов необобщенных коллекций:

ICollection<T>

Определяет основополагающие свойства обобщенных коллекций

IComparer<T>

Определяет обобщенный метод Compare() для сравнения объектов, хранящихся в коллекции

IDictionary<Tkey, TValue>

Определяет обобщенную коллекцию, состоящую из пар "ключ-значение"

IEnumerable<T>

Определяет обобщенный метод GetEnumerator(), предоставляющий перечислитель для любого класса коллекции

Enumerator<T>

Предоставляет методы, позволяющие получать содержимое коллекции по очереди

IEqualityComparer<T>

Сравнивает два объекта на предмет равенства

IList<T>

Определяет обобщенную коллекцию, доступ к которой можно получить с помощью индексатора

В пространстве имен System.Collections.Generic определена структура **KeyValuePair<TKey, TValue>** Она служит для хранения ключа и его значения и применяется в классах обобщенных коллекций, в которых хранятся пары "ключ-значение", как, например, в классе Dictionary<TKey, TValue> В этой структуре определяются два следующих свойства:

```
public TKey Key { get; };
public TValue Value { get; };
```

В этих свойствах хранятся ключ и значение соответствующего элемента коллекции.

Классы обобщенных коллекций

Классы обобщенных коллекций по большей части соответствуют своим необобщенным аналогам, хотя в некоторых случаях они носят другие имена. Отличаются они также своей организацией и функциональными возможностями. Классы обобщенных коллекций определяются в пространстве имен System.Collections.Generic:

Dictionary<Tkey, TValue>

Сохраняет пары "ключ-значение". Обеспечивает такие же функциональные возможности, как и необобщенный класс Hashtable

HashSet<T>

Сохраняет ряд уникальных значений, используя хештаблицу

LinkedList<T>

Сохраняет элементы в двунаправленном списке

List<T>

Создает динамический массив. Обеспечивает такие же функциональные возможности, как и необобщенный класс ArrayList

Queue<T>

Создает очередь. Обеспечивает такие же функциональные возможности, как и необобщенный класс Queue

SortedDictionary<TKey, TValue>

Создает отсортированный список из пар "ключ-значение"

SortedList<TKey, TValue>

Создает отсортированный список из пар "ключ-значение". Обеспечивает такие же функциональные возможности, как и необобщенный класс SortedList

SortedSet<T>

Создает отсортированное множество

Stack<T>

Создает стек. Обеспечивает такие же функциональные возможности, как и необобщенный класс Stack