**№ 10 Коллекции**

**Курносенко Софья**

1. Создайте класс по варианту, определите в нем свойства и методы, реализуйте указанный интерфейс и другие при необходимости, соберите объекты класса в коллекцию (можно сделать специальных класс с вложенной коллекцией и методами ею управляющими), продемонстрируйте работу с ней (добавление/удаление/поиск/вывод:

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Тип | Интерфейс | Коллекция |
| 1 | Автомобиль | IList<T> | Dictionary < TKey, TValue> |
| 2 | Книга | IDictionary< TKey, TValue> | List<T> |
| 3 | Товар | IOrderedDictionary | ConcurrentBag<T> |
| 4 | Работник | IEnumerable<T> | Hashtable |
| 5 | Студент | IEnumerable<T> | Queue<T> |
| 6 | Компьютер | ISet<T> | HashSet<T> |
| 7 | Программное обеспечение | IList<T> | SortedList < TKey, TValue> |
| 8 | Мебель | IList<T> | ArryList |
| 9 | Изображение | ISet<T> | LinkedList<T> |
| 10 | Игра | IEnumerable<T> | BlockingCollection<T> |
| 11 | Геометрическая фигура | IEnumerator | Stack |
| 12 | Интернет-ресурс | IList<T> | ConcurrentDictionery< TKey, TValue> |
| 13 | Услуги | IOrderedDictionary | Queue<T> |
| 14 | Концерт | любой | Dictionary < TKey, TValue> |
| 15 | Растение | IList<T> | HashSet<T> |

2. Создайте **универсальную коллекцию** в соответствии с вариантом задания и заполнить ее данными встроенного типа .Net (int, char,…).

a. Выведите 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* (из таблицы выберите другой тип коллекции) 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.

3. 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 таблицы.

**► Коллекции**

**Введение в коллекции**

Хотя в языке C# есть массивы, которые хранят в себе наборы однотипных объектов, но работать с ними не всегда удобно. Например, **массив хранит фиксированное количество объектов**, однако что если мы заранее не знаем, сколько нам потребуется объектов. И в этом случае намного удобнее применять коллекции. Коллекции предоставляют более гибкий способ работы с группами объектов. В отличие от массивов, **коллекция**, с которой вы работаете, **может расти или уменьшаться *динамически* при необходимости**. Еще один плюс коллекций состоит в том, что некоторые из них реализуют стандартные структуры данных, например, стек, очередь, словарь, которые могут пригодиться для решения различных специальных задач.

Большая часть классов коллекций содержится в пространствах имен System.Collections (простые необобщенные классы коллекций), System.Collections.Generic (обобщенные или типизированные классы коллекций) и System.Collections.Specialized (специальные классы коллекций). Также для обеспечения параллельного выполнения задач и многопоточного доступа применяются классы коллекций из пространства имен System.Collections.Concurrent.

Основой для создания всех коллекций является реализация интерфейсов **IEnumerator** и **IEnumerable** (и их обобщенных двойников **IEnumerator<T>** и **IEnumerable<T>**). Интерфейс IEnumerator представляет перечислитель, с помощью которого становится возможен последовательный перебор коллекции, например, в цикле **foreach**. А интерфейс IEnumerable через свой метод GetEnumerator предоставляет перечислитель всем классам, реализующим данный интерфейс. Поэтому интерфейс IEnumerable (IEnumerable<T>) является базовым для всех коллекций.

**Краткий обзор 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 крайне 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**.

## ► Необобщенные коллекции

Необобщенные или простые коллекции определены в пространстве имен System.Collections. Их особенность состоит в том, что их функциональность, функциональные возможности описываются в интерфейсах, которые также находятся в этом пространстве имен.

Рассмотрим основные интерфейсы:

* **IEnumerable**: определяет метод GetEnumerator. Данный метод возвращает перечислитель - то есть некоторый объект, реализующий интерфейс IEnumerator.
* **IEnumerator**: реализация данного интерфейса позволяет перебирать элементы коллекции с помощью цикла foreach
* **ICollection**: является основой для всех необобщенных коллекций, определяет основные методы и свойства для всех необобщенных коллекций (например, метод CopyTo и свойство Count). Данный интерфейс унаследован от интерфейса IEnumerable, благодаря чему базовый интерфейс также реализуется всеми классами необобщенных коллекций
* **IList**: предназначен для создания списков, элементы которых доступны по своим позициям. Также определяет ряд методов для манипуляции элементами: Add (добавление элементов), Remove/RemoveAt (удаление элемента) и ряд других.
* **IComparer**: определяет метод int Compare(object x, object y) для сравнения двух объектов
* **IDictionary**: определяет поведение коллекции, при котором она должна хранить объекты в виде пар ключ-значение: для каждого объекта определяется уникальный ключ, и этому ключу соответствует определенное значение
* **IDictionaryEnumerator**: определяет методы и свойства для перечислителя словаря
* **IEqualityComparer**: определяет два метода Equals и GetHashCode, с помощью которых два объекта сравниваются на предмет равенства
* **IStructuralComparer**: определяет метод Compare для структурного сравнения двух объектов: при таком сравнении сравниваются не ссылки на объекты, а непосредственное содержимое объектов
* **IStructuralEquatable**: позволяет провести структурное равенство двух объектов. Как и в случае с интерфейсом IStructuralComparer сравнивается содержимое двух объектов

Эти интерфейсы реализуются следующими классами коллекций в пространстве имен System.Collections:

* **ArrayList**: класс простой коллекции объектов. Реализует интерфейсы IList, ICollection, IEnumerable
* **BitArray**: класс коллекции, содержащей массив битовых значений. Реализует интерфейсы ICollection, IEnumerable
* **Hashtable**: класс коллекции, представляющей хэш-таблицу и храняющий набор пар "ключ-значение"
* **Queue**: класс очереди объектов, работающей по алгоритму FIFO("первый вошел -первый вышел"). Реализует интерфейсы ICollection, IEnumerable
* **SortedList**: класс коллекции, хранящей наборы пар "ключ-значение", отсортированных по ключу. Реализует интерфейсы ICollection, IDictionary, IEnumerable
* **Stack**: класс стека

## ► Обобщенные коллекции

Библиотека классов .NET предоставляет ряд универсальных классов коллекций в пространствах имен [System.Collections.Generic](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic) и [System.Collections.ObjectModel](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.objectmodel).

**Универсальная коллекция** обеспечивает строгую типизацию, так что в нее нельзя добавить другие типы данных.

Многие универсальные типы коллекций являются прямыми аналогами неуниверсальных типов. [List<T>](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic.list-1) — это универсальная версия [ArrayList](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.arraylist).

В теме [Обобщенные типы](https://metanit.com/sharp/tutorial/3.12.php) мы уже говорили про обобщения: как создавать и использовать обобщенные классы, роль универсального параметра T. Обобщенные коллекции - это те же самые обобщенные классы. И опять же их использование перед необобщенными коллекциями имеет те же преимущества: повышение производительности (не надо тратить время на упаковку и распаковку объекта) и повышенная типобезопасность.

Классы обобщенных коллекций находятся в пространстве имен **System.Collections.Generic**. Функционал коллекций также по большей части описывается в обобщенных интерфейсах.

Только интерфейсы обобщенных коллекций отличаются от необобщеных двойников не только наличием универсального параметра T, но и самой функциональностью. Рассмотрим основные интерфейсы обобщенных коллекций:

* **IEnumerable<T>**: определяет метод GetEnumerator, с помощью которого можно получать элементы любой коллекции

. Реализация данного интерфейса позволяет перебирать элементы коллекции с помощью цикла foreach

* **IEnumerator<T>**: определяет методы, с помощью которых потом можно получить содержимое коллекции по очереди
* **ICollection<T>**: представляет ряд общих свойств и методов для всех обобщенных коллекций (например, методы CopyTo, Add, Remove, Contains, свойство Count)
* **IList<T>**: предоставляет функционал для создания последовательных списков
* **IComparer<T>**: определяет метод Compare для сравнения двух однотипных объектов
* **IDictionary<TKey, TValue>**: определяет поведение коллекции, при котором она должна хранить объекты в виде пар ключ-значение: для каждого объекта определяется уникальный ключ типа, указанного в параметре TKey, и этому ключу соответствует определенное значение, имеющее тип, указанный в параметре TValue
* **IEqualityComparer<T>**: определяет методы, с помощью которых два однотипных объекта сравниваются на предмет равенства

Эти интерфейсы реализуются следующими классами коллекций в пространстве имен System.Collections.Generic:

* **List<T>**: класс, представляющий последовательный список. Реализует интерфейсы IList<T>, ICollection<T>, IEnumerable<T>
* **Dictionary<TKey, TValue>**: класс коллекции, хранящей наборы пар "ключ-значение". Реализует интерфейсы ICollection<T>, IEnumerable<T>, IDictionary<TKey, TValue>
* **LinkedList<T>**: класс двухсвязанного списка. Реализует интерфейсы ICollection<T> и IEnumerable<T>
* **Queue<T>**: класс очереди объектов, работающей по алгоритму FIFO("первый вошел – первый вышел"). Реализует интерфейсы ICollection, IEnumerable<T>
* **SortedSet<T>**: класс отсортированной коллекции однотипных объектов. Реализует интерфейсы ICollection<T>, ISet<T>, IEnumerable<T>
* **SortedList<TKey, TValue>**: класс коллекции, хранящей наборы пар "ключ-значение", отсортированных по ключу. Реализует интерфейсы ICollection<T>, IEnumerable<T>, IDictionary<TKey, TValue>
* **SortedDictionary<TKey, TValue>**: класс коллекции, хранящей наборы пар "ключ-значение", отсортированных по ключу. В общем похож на класс SortedList<TKey, TValue>, основные отличия состоят лишь в использовании памяти и в скорости вставки и удаления
* **Stack<T>**: класс стека однотипных объектов. Реализует интерфейсы ICollection, IReadOnlyCollection<T> и IEnumerable<T>

Большинство обобщенных классов коллекций дублируют необобщенные классы коллекций. Но если вам не надо хранить объекты разных типов, то предпочтительнее использовать обобщенные коллекции.

## ► Список List<T>

Класс List<T> из пространства имен System.Collections.Generic представляет простейший список однотипных объектов.

Среди его методов можно выделить следующие:

* **void Add(T item)**: добавление нового элемента в список
* **void AddRange(ICollection collection)**: добавление в список коллекции или массива
* **int BinarySearch(T item)**: бинарный поиск элемента в списке. Если элемент найден, то метод возвращает индекс этого элемента в коллекции. При этом список должен быть отсортирован.
* **int IndexOf(T item)**: возвращает индекс первого вхождения элемента в списке
* **void Insert(int index, T item)**: вставляет элемент item в списке на позицию index
* **bool Remove(T item)**: удаляет элемент item из списка, и если удаление прошло успешно, то возвращает true
* **void RemoveAt(int index)**: удаление элемента по указанному индексу index
* **void Sort()**: сортировка списка

Посмотрим реализацию списка на примере:

using System;

using System.Collections.Generic;

namespace Collections

{

class Program

{

static void Main(string[] args)

{

List<int> numbers = new List<int>() { 1, 2, 3, 45 };

numbers.Add(6); // добавление элемента

numbers.AddRange(new int[] { 7, 8, 9 });

numbers.Insert(0, 666); // вставляем на первое место в списке число 666

numbers.RemoveAt(1); // удаляем второй элемент

foreach (int i in numbers)

{

Console.WriteLine(i);

}

List<Person> people = new List<Person>(3);

people.Add(new Person() { Name = "Том" });

people.Add(new Person() { Name = "Билл" });

foreach (Person p in people)

{

Console.WriteLine(p.Name);

}

Console.ReadLine();

}

}

class Person

{

public string Name { get; set; }

}

}

## ► Класс ObservableCollection

Кроме стандартных классов коллекций типа списков, очередей, словарей, стеков .NET также предоставляет специальный класс **ObservableCollection**. Он по функциональности похож на список List за тем исключением, что позволяет известить внешние объекты о том, что коллекция была изменена. Класс ObservableCollection <T> определяет событие – **CollectionChanged**, которое запускается в ответ на добавление элемента, удаление, замену.

Рассмотрим на примере:

using System;

using System.Collections.ObjectModel;

using System.Collections.Specialized;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

ObservableCollection<User> users = new ObservableCollection<User>

{

new User { Name = "Bill"},

new User { Name = "Tom"},

new User { Name = "Alice"}

};

users.CollectionChanged += Users\_CollectionChanged;

users.Add(new User { Name = "Bob" });

users.RemoveAt(1);

users[0] = new User { Name = "Anders" };

foreach (User user in users)

{

Console.WriteLine(user.Name);

}

Console.Read();

}

private static void Users\_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)

{

switch (e.Action)

{

case NotifyCollectionChangedAction.Add: // если добавление

                    User newUser = e.NewItems[0] as User;

Console.WriteLine($"Добавлен новый объект: {newUser.Name}");

break;

case NotifyCollectionChangedAction.Remove: // если удаление

                    User oldUser = e.OldItems[0] as User;

Console.WriteLine($"Удален объект: {oldUser.Name}");

break;

case NotifyCollectionChangedAction.Replace: // если замена

                    User replacedUser = e.OldItems[0] as User;

User replacingUser = e.NewItems[0] as User;

Console.WriteLine($"Объект {replacedUser.Name} заменен объектом {replacingUser.Name}");

break;

}

}

}

class User

{

public string Name { get; set; }

}

}

Во-первых, класс ObservableCollection находится в пространстве имен System.Collections.ObjectModel, кроме того, также понадобится ряд объектов из пространства System.Collections.Specialized, поэтому в начале подключаем эти пространства имен.

Класс ObservableCollection определяет событие **CollectionChanged**, подписавшись на которое, мы можем обработать любые изменения коллекции.

В обработчике этого события Users\_CollectionChanged для получения всей информации о событии используется объект **NotifyCollectionChangedEventArgs** e. Его свойство **Action** позволяет узнать характер изменений. Оно хранит одно из значений из перечисления **NotifyCollectionChangedAction**.

Свойства NewItems и OldItems позволяют получить соответственно добавленные и удаленные объекты. Таким образом, мы получаем полный контроль над обработкой добавления, удаления и замены объектов в коллекции.

ObservableCollection <T> поддерживает следующие интерфейсы:

ICollection<T>, INotifyCollectionChanged, INotifyPropertyChanged.

**Поддержка интерфейса означает возможность:**

* Добавления элементов в коллекцию через метод Add();
* Удаления элементов из коллекции вызовом метода Remove();
* Копирования из коллекции и в коллекцию методом CopyTo();
* Вставка элемента с помощью метода Insert и изменение элементов через использование индексатора и метода SetItem.

Использование этих методов идентично для классов List<T> и ObservableCollection<T> и нет смысла их отдельно рассматривать.

**Схема работы с классом типична для .NET:**

* определяется обработчик события;
* оформляется подписка на событие;
* реализуется код, где генерируется событие.

Обработчик события должен принимать объект события типа NotifyCollectionChangedEventArgs, из его свойства Action одно из значений перечисления NotifyCollectionChangedAction, которое должно проинформировать о характере произошедшего события.