**Вопросы для лабораторной №10 КОЛЛЕКЦИИ**

**1. Перечислите стандартные коллекции NET Framework**

Необобщенные (System.Collections), обобщенные (System.Collections.Generic), специальные(System.Collections.Specialized), с поразрядной организацией (BitArray), параллельные, (System.Collections.Concurrent).

**2. Поясните принцип работы коллекции:**

**a) Stack<T>**  представляет коллекцию, которая использует алгоритм LIFO ("последний вошел - первый вышел"). При такой организации каждый следующий добавленный элемент помещается поверх предыдущего. Извлечение из коллекции происходит в обратном порядке - извлекается тот элемент, который находится выше всех в стеке.

**b) Queue<T>** представляет обычную очередь, работающую по алгоритму FIFO ("первый вошел - первый вышел").

**c) HashSet<T>** представляет хеш-таблицу. Он наследует свой функционал от класса AbstractSet, а также реализует интерфейс Set.

Хеш-таблица представляет такую структуру данных, в которой все объекты имеют уникальный ключ или хеш-код. Данный ключ позволяет уникально идентифицировать объект в таблице.

**d) List<T>** представляет простейший список однотипных объектов.

**e) Dictionary<Tkey, TValue>** Благодаря свойствам Key и Value, которые есть у данной структуры, мы можем получить ключ и значение элемента в словаре.

**f) LinkedList<T>** представляет двухсвязный список, в котором каждый элемент хранит ссылку одновременно на следующий и на предыдущий элемент.

**g) 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

**h) SortedList<TKey, TValue>** класс коллекции, хранящей наборы пар "ключ-значение", отсортированных по ключу. Реализует интерфейсы ICollection<T>, IEnumerable<T>, IDictionary<TKey, TValue>

**i) SortedSet<T>** класс отсортированной коллекции однотипных объектов. Реализует интерфейсы ICollection<T>, ISet<T>, IEnumerable<T>

**3. Охарактеризуйте необобщенные, специальные, с поразрядной организацией, обобщенные и параллельные 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.

***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 ко ***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. ллекций. Коллекция типа BitArray объявляется в пространстве имен System.Collections.

**4. Какие интерфейсы используются в коллекциях C#?**

|  |  |
| --- | --- |
| Интерфейс | Описание |
| IEnumerable<T> | Интерфейс IEnumerable необходим, когда с коллекцией используется оператор foreach. Этот интерфейс определяет метод GetEnumerator(), возвращающий перечислитель, который реализует IEnumerator. |
| ICollection<T> | ICollection<T> — это интерфейс, реализованный классами обобщенных коллекций. С его помощью можно получить количество элементов в коллекции (свойство Count) и скопировать коллекцию в массив (метод СоруТо()). Можно также добавлять и удалять элементы из коллекции (Add(), Remove(), Clear()). |
| IList<T> | Интерфейс IList<T> предназначен для создания списков, элементы которых доступны по своим позициям. Этот интерфейс определяет индексатор, а также способы вставки и удаления элементов в определенные позиции (методы Insert() и Remove()). IList<T> унаследован от ICollection<T>. |
| ISet<T> | Интерфейс ISet<T> появился в версии .NET 4. Этот интерфейс реализуется множествами. Он позволяет комбинировать различные множества в объединения, а также проверять, не пересекаются ли два множества. ISet<T> унаследован от ICollection<T>. |
| IDictionary<TKey, TValue> | Интерфейс IDictionary<TKey, TValue> реализуется обобщенными классами коллекций, элементы которых состоят из ключа и значения. С помощью этого интерфейса можно получать доступ ко всем ключам и значениям, извлекать элементы по индексатору типа ключа, а также добавлять и удалять элементы. |
| ILookup<TKey, TValue> | Подобно IDictionary<TKey, TValue> поддерживает ключи и значения. Однако в этом случае коллекция может содержать множественные значения для одного ключа. |
| IComparer<T> | Интерфейс IComparer<T> реализован компаратором и используется для сортировки элементов внутри коллекции с помощью метода Compare(). |
| IEqualityComparer<T> | Интерфейс IEqualityComparer<T> реализован компаратором, который может быть применен к ключам словаря. Через этот интерфейс объекты могут быть проверены на предмет эквивалентности друг другу. В .NET 4 этот интерфейс также реализован массивами и кортежами. |
| IProducerConsumerCollection<T> | Интерфейс IProducerConsumerCollection<T> был добавлен в версию .NET 4 для поддержки новых, безопасных в отношении потоков классов коллекций. |

**5. Для чего используется интерфейс IComparable?**

Для сортировки наборов сложных объектов применяется интерфейс IComparable. Он имеет всего один метод:

public interface IComparable

{

int CompareTo(object o);

}

Метод CompareTo предназначен для сравнения текущего объекта с объектом, который передается в качестве параметра object o. На выходе он возвращает целое число, которое может иметь одно из трех значений:

* Меньше нуля. Значит, текущий объект должен находиться перед объектом, который передается в качестве параметра
* Равен нулю. Значит, оба объекта равны
* Больше нуля. Значит, текущий объект должен находиться после объекта, передаваемого в качестве параметра

**6. Что содержит интерфейс IEnumerator или обобщенный интерфейс**

**IEnumerator<T>? Где и как его можно использовать?**

Интерфейс IEnumerable имеет метод, возвращающий ссылку на другой интерфейс - перечислитель:

|  |  |
| --- | --- |
| 1  2  3  4 | public interface IEnumerable  {      IEnumerator GetEnumerator();  } |

А интерфейс IEnumerator определяет функционал для перебора внутренних объектов в контейнере:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public interface IEnumerator  {      bool MoveNext(); // перемещение на одну позицию вперед в контейнере элементов      object Current {get;}  // текущий элемент в контейнере      void Reset(); // перемещение в начало контейнера  } |

**7. Что такое наблюдаемая коллекция? Где и каким образом ее можно использовать?**

В случае если нужна информация о том, когда элементы коллекции удаляются или добавляются, можно использовать класс **ObservableCollection<T>**. Этот класс был определен для WPF и предназначен для того, чтобы пользовательский интерфейс мог получать информацию об изменениях коллекции. По этой причине он включен в сборку WindowsBase, следовательно, на нее необходимо сослаться. Пространство имен этого класса — **System.Collections.ObjectModel**.

Класс ObservableCollection<T> унаследован от базового класса Collection<T>, который может применяться для создания специальных коллекций; он использует внутри себя List<T>. Методы базового класса SetItem() и RemoveItem() переопределены для инициации события CollectionChanged. Клиенты этого класса могут регистрироваться на это событие, используя интерфейс INotifyCollectionChanged.