1. **Перечислите стандартные коллекции NET Framework.**

* **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

1. **Поясните принцип работы коллекции:**

**a)Stack**

Класс 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.

В классе Stack можно выделить два основных метода, которые позволяют управлять элементами:

* **Push**: добавляет элемент в стек на первое место
* **Pop**: извлекает и возвращает первый элемент из стека
* **Peek**: просто возвращает первый элемент из стека без его удаления

**b)Queue**

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

У класса Queue<T> можно отметить следующие методы:

* **Dequeue**: извлекает и возвращает первый элемент очереди
* **Enqueue**: добавляет элемент в конец очереди
* **Peek**: просто возвращает первый элемент из начала очереди без его удаления

**c)HashSet**

Класс [HashSet<T>](https://msdn.microsoft.com/ru-ru/library/bb359438(v=vs.110).aspx) основан на модели математических множеств и позволяет проводить высокопроизводительные операции над множествами, подобные доступу к ключам в [Dictionary<TKey, TValue>](https://msdn.microsoft.com/ru-ru/library/xfhwa508(v=vs.110).aspx) или [Hashtable](https://msdn.microsoft.com/ru-ru/library/system.collections.hashtable(v=vs.110).aspx). Проще говоря, класс [HashSet<T>](https://msdn.microsoft.com/ru-ru/library/bb359438(v=vs.110).aspx) можно рассматривать как коллекцию [Dictionary<TKey, TValue>](https://msdn.microsoft.com/ru-ru/library/xfhwa508(v=vs.110).aspx), но без значений.

Коллекция [HashSet<T>](https://msdn.microsoft.com/ru-ru/library/bb359438(v=vs.110).aspx) не сортируется и не может содержать повторяющихся элементов. Если упорядочивание или дублирование элементов является более важным, чем производительность приложения, то следует рассмотреть использование класса [List<T>](https://msdn.microsoft.com/ru-ru/library/6sh2ey19(v=vs.110).aspx) вместе с методом [Sort](https://msdn.microsoft.com/ru-ru/library/3da4abas(v=vs.110).aspx).

Класс [HashSet<T>](https://msdn.microsoft.com/ru-ru/library/bb359438(v=vs.110).aspx) позволяет осуществлять многие математические операции над множествами, такие как сложение (объединение) и вычитание множеств. В следующей таблице перечислены возможные операции [HashSet<T>](https://msdn.microsoft.com/ru-ru/library/bb359438(v=vs.110).aspx) и их математические эквиваленты.

**d)List**

Класс List<T> 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 с 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

**e)Dictionary**

Еще один распространенный тип коллекции представляют словари. Словарь хранит объекты, которые представляют пару ключ-значение. Каждый такой объект является объектом структуры **KeyValuePair<TKey, TValue>**. Благодаря свойствам Key и Value, которые есть у данной структуры, мы можем получить ключ и значение элемента в словаре.

Класс словарей также, как и другие коллекции, предоставляет методы Add и Remove для добавления и удаления элементов. Только в случае словарей в метод Add передаются два параметра: ключ и значение. А метод Remove удаляет не по индексу, а по ключу.

**f) LinkedList**

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

Если в простом списке List<T> каждый элемент представляет объект типа T, то в LinkedList<T> каждый узел представляет объект класса LinkedListNode<T>. Этот класс имеет следующие свойства:

* **Value**: само значение узла, представленное типом T
* **Next**: ссылка на следующий элемент типа LinkedListNode<T> в списке. Если следующий элемент отсутствует, то имеет значение null
* **Previous**: ссылка на предыдущий элемент типа LinkedListNode<T> в списке. Если предыдущий элемент отсутствует, то имеет значение null

Используя методы класса LinkedList<T>, можно обращаться к различным элементам, как в конце, так и в начале списка:

* **AddAfter(LinkedListNode<T> node, LinkedListNode<T> newNode)**: вставляет узел newNode в список после узла node.
* **AddAfter(LinkedListNode<T> node, T value)**: вставляет в список новый узел со значением value после узла node.
* **AddBefore(LinkedListNode<T> node, LinkedListNode<T> newNode)**: вставляет в список узел newNode перед узлом node.
* **AddBefore(LinkedListNode<T> node, T value)**: вставляет в список новый узел со значением value перед узлом node.
* **AddFirst(LinkedListNode<T> node)**: вставляет новый узел в начало списка
* **AddFirst(T value)**: вставляет новый узел со значением value в начало списка
* **AddLast(LinkedListNode<T> node)**: вставляет новый узел в конец списка
* **AddLast(T value)**: вставляет новый узел со значением value в конец списка
* **RemoveFirst()**: удаляет первый узел из списка. После этого новым первым узлом становится узел, следующий за удаленным
* **RemoveLast()**: удаляет последний узел из списка

**g)SortedDictionary**

**h)SortedList**

Класс SortedDictionary<TKey, Tvalue> представляет дерево бинарного поиска, в котором все элементы отсортированы на основе ключа. Тип ключа должен реализовать интерфейс IComparable<TKey>. Если тип ключа не сортируемый, компаратор можно также создать, реализовав IComparer<TKey> и указав его в качестве аргумента конструктора сортированного словаря.

Классы SortedDictionary<TKey, Tvalue> и SortedList<TKey, TValue> имеют схожую функциональность. Но поскольку SortedList<TKey, TValue> реализован в виде списка, основанного на массиве, a SortedDictionary<TKey, Tvalue> реализован как словарь, эти классы обладают разными характеристиками:

* SortedList<TKey, TValue> использует меньше памяти, чем SortedDictionary<TKey, TValue>
* SortedDictionary<TKey, TValue> быстрее вставляет и удаляет элементы.
* При наполнении коллекции отсортированными данными SortedList<TKey,TValue> работает быстрее, если при этом не требуется изменение емкости.

В классе SortedDictionary<TKey, TValue> реализуются интерфейсы IDictionary, IDictionary<TKey, TValue>, ICollection, ICollection<KeyValuePair<TKey, TValue>>, IEnumerable и IEnumerable<KeyValuePair<TKey, TValue>>. В классе SortedDictionary<TKey, TValue>

**i) SortedSet**

Упорядоченное множество

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

***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**.

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

* **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**: позволяет получать элементы коллекции по порядку. 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

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

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

|  |  |
| --- | --- |
| 1  2  3  4 | 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

1. **Что содержит интерфейс IEnumerator или обобщенный интерфейс IEnumerator? Где и как его можно использовать?**

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.

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

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.

Во-первых, класс ObservableCollection находится в пространстве имен System.Collections.ObjectModel, кроме того, также понадобятся 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 объектNotifyCollectionChangedEventArgs e. Его свойство Action позволяет узнать характер изменений. Оно хранит одной 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.