1. На какие основные виды/типы делятся все коллекции .NET? Охарактеризуйте каждый из них.

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**.

2. Что такое generic-коллекции? Назовите примеры известных вам generic-коллекций.

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>**

3. В чем разница между ArrayList и Array?

Array – статический массив (т.е. на этапе компиляции известна размерность массива), ArrayList – динамический массив(является классом Collection переменной длины).

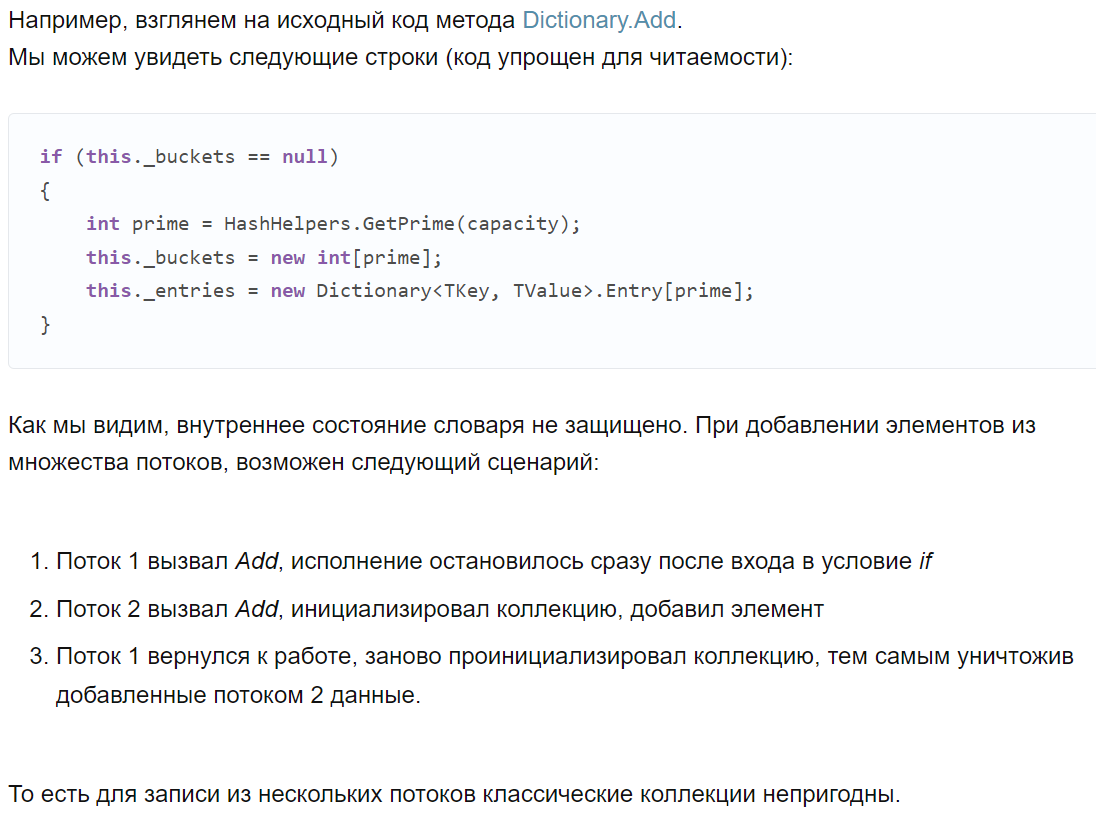
4. Охарактеризуйте коллекции, которые вы использовали в своем варианте.

**В 8 варианте:**

* ArrayList – динамический массив / ArrayList<T> - обобщенная коллекция динамического массива.
* Query – очередь (Last In Last Out) / Query<T> - обобщенная коллекция очереди.

5. Чем отличаются коллекции, расположенные в пространстве имен System.Collections.Concurrent?

Предоставляет несколько потокобезопасных классов коллекций, которые следует использовать вместо соответствующих типов в пространствах имен System.Collections и System.Collections.Generic, если несколько потоков параллельно обращаются к такой коллекции.

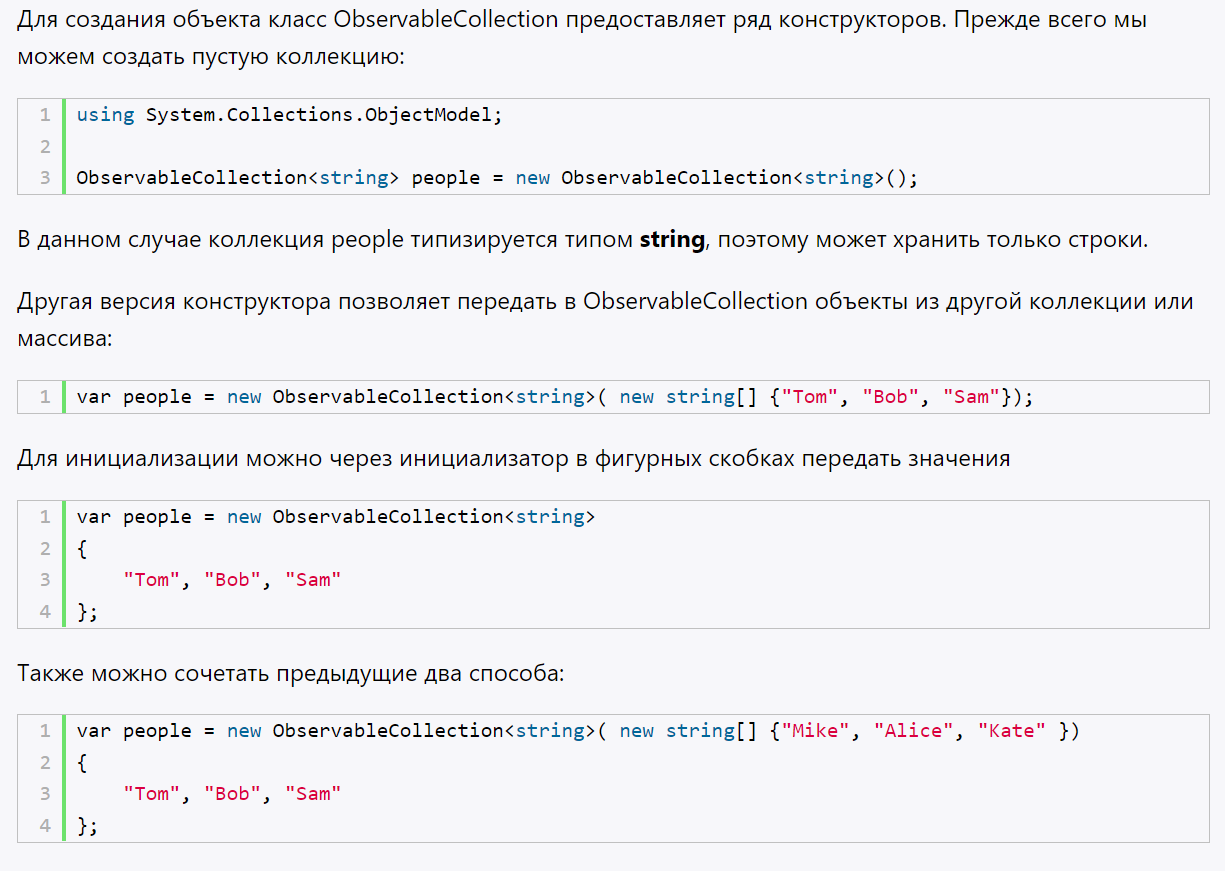


6. Какое пространство имен необходимо подключить в проект, чтобы иметь возможность использовать generic-коллекции?

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

7. Что такое наблюдаемая коллекция? Как ее можно использовать?

Кроме стандартных классов коллекций типа списков, очередей, словарей, стеков .NET также предоставляет специальный класс **ObservableCollection<T>** (с англ. – наблюдаемая коллекция). В отличие от ранее рассмотренных коллекций данный класс определен в пространстве имен **System.Collections.ObjectModel**. По функциональности коллекция ObservableCollection похожа на список List за тем исключением, что позволяет известить внешние объекты о том, что коллекция была изменена.



8. Охарактеризуйте интерфейсы IEnumerator, IEnumerable. В чем отличие назначений интерфейсов IEnumerator и IEnumerable.

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.

Интерфейс IEnumerable объявляет только один метод GetEnumerator, который возвращает другой тип интерфейса, называемый интерфейсом IEnumerator для этой конкретной коллекции. IEnumerator, с другой стороны, является базовым интерфейсом для всех не общих счетчиков, которые используются для чтения данных в коллекции.

An IEnumerator- это вещь, которая может перечислять: у нее есть Current свойство MoveNext и Reset методы and (которые в .NET-коде вы, вероятно, не будете вызывать явно, хотя могли бы).

An IEnumerable- это вещь, которую можно перечислить...что просто означает, что у него есть метод GetEnumerator, который возвращает IEnumerator.

9. Поясните принцип работы коллекций:

a. LinkedList <T>

Сохраняет элементы в двунаправленном списке

Двусвязный (двунаправленный) список — это разновидность связного списка, при которой переход по элементам возможен в обоих направлениях (как вперед, так и назад), в отличие от односвязного (однонаправленного) списка.

Вот термины, необходnuимые для понимания концепции двусвязных (двунаправленных) списков:

* **Ссылка**. В каждой ссылке связного списка могут храниться данные, называемые элементом.
* **Следующая ссылка**. В каждой ссылке связного списка содержится ссылка на следующую ссылку (Next).
* **Предыдущая ссылка**. В каждой ссылке связного списка содержится ссылка на предыдущую ссылку (Prev).
* **Связный список**содержит ссылку (связку) на первую ссылку (First) и на последнюю ссылку (Last).

b. HashSet <T>

Сохраняет ряд уникальных значений, используя хештаблицу.

c. Dictionary <Tkey, TValue>

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

d. ConcurrentBag <Tkey, TValue>

ConcurrentBag предоставляет поточно-ориентированную коллекцию неупорядоченного набора элементов. Таким образом, он будет часто использоваться в параллельном программировании для обеспечения производительности. ConcurrentDictionary работает в три раза быстрее, чем Dictionary.

e. Stack<t>, Queue<T>

Stack – динамическая структура данных, работающая по принципу LIFO(Last In First Out).

Очередь (queue) - это структура данных, которая работает по принципу FIFO (First In First Out - Первый пришел, первый вышел). При добавлении новый элемент помещается в конец очереди или ее хвост, а удаление идет с начала очереди или головы очереди.

Банальным примером очереди может служить обычная очередь в больнице, где новые пациенты встают в конец очереди, а прием пациентов осуществляется с начала очереди.

f. SortedList, SortedList.

Создает отсортированный список из пар "ключ-значение".

Коллекция SortedList представляет собой гибрид структур данных Hashtable и Array . Это словарь, содержащий пары ключ / значение, которые автоматически сортируются в соответствии с данными в ключе. Это означает, что, в отличие от Hashtable, содержимое SortedList имеет определенный порядок и может быть доступно с помощью индексного номера, а также путем поиска ключа.  
  
Вы можете подумать о том, что SortedList имеет два массива для хранения ключей и значений отдельно. По мере добавления элементов ключ вставляется в массив ключей в правильном месте для поддержания порядка сортировки. Значение вставляется в соответствующую позицию в массиве значений. Как и в случае с Hashtable, это сопряжение ключа и значения может быть доступно в качестве DictionaryEntry . Каждый словарь может иметь объекты для своего ключа и значения до тех пор, пока ключевая часть не является нулевой, является уникальной и поддерживает интерфейс IComparable, который позволяет сравнивать и сортировать элементы.  
  
SortedList предоставляет дополнительную функциональность по сравнению с Hashtable из-за его способности быть адресованным по номеру индекса и из-за его автоматической сортировки. Следует отметить, что этот дополнительный функционал дает пониженную производительность.