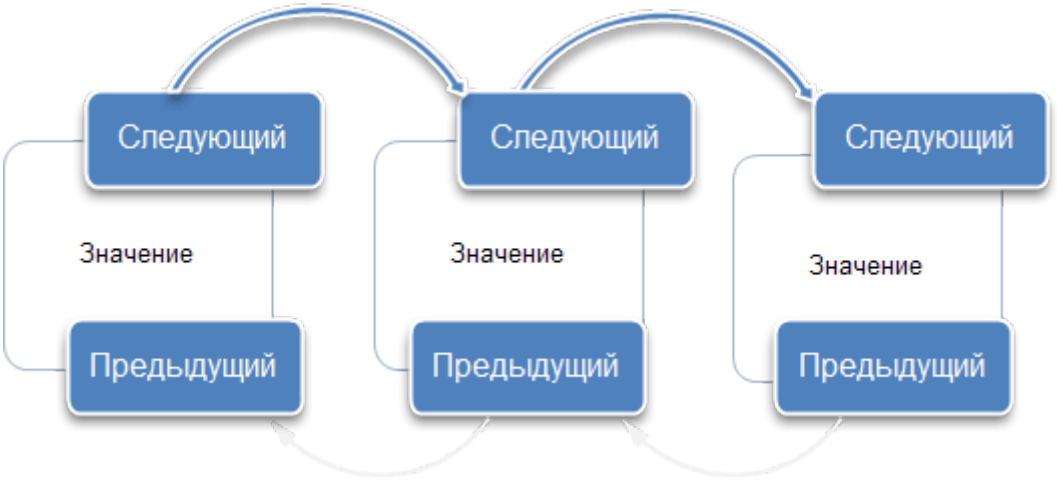
LinkedList<T> and LinkedListNode<T>

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
- RemoveLast(): удаляет последний узел из списка

В классе LinkedList<T> реализуются **интерфейсы** ICollection, ICollection<T>, IEnumerable, IEnumerable<T>, ISerializable и IDeserializationCallback. В двух последних интерфейсах поддерживается сериализация списка.

В классе LinkedList<T> определяются два **конструктора**:
public LinkedList()
public LinkedList(IEnumerable<T> collection)
В первом конструкторе создается пустой связный список, а во втором конструкторе — список, инициализируемый элементами из коллекции collection.

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

SortedList (vs SortedDictionary)

Этот класс сортирует элементы на основе значения ключа. Можно использовать не только любой тип значения, но также и любой тип ключа.

Объект SortedList может получить доступ к элементу по ключу, как элемент в любой IDictionary реализации, или по индексу, как элемент в любой IList реализации.

SortedList на внутреннем уровне поддерживает два массива для хранения элементов списка, то есть один массив для ключей, другой для их значения.

Класс SortedList<TKey, TValue> подобен классу SortedDictionary<TKey, TValue>, но у него другие рабочие характеристики. В частности, класс SortedList<TKey, TValue> использует меньше памяти, тогда как класс SortedDictionary<TKey, TValue> позволяет быстрее вставлять и удалять неупорядоченные элементы в коллекцию.

В классе SortedList<TKey, TValue> реализуются **интерфейсы** IDictionary, IDictionary<TKey, TValue>, ICollection, ICollection<KeyValuePair<TKey, TValue>>, IEnumerable и IEnumerable<KeyValuePair<TKey, TValue>. Размер коллекции типа SortedList<TKey, TValue> изменяется динамически, автоматически увеличиваясь по мере необходимости.

Конструкторы:

- *public SortedList()* - пустой список с выбираемой по умолчанию первоначальной емкостью
- *public SortedList(IDictionary<TKey, TValue> dictionary)* - отсортированный список с указанным количеством элементов dictionary
- *public SortedList(int capacity)* - с помощью параметра capacity задается емкость коллекции, создаваемой в виде отсортированного списка
- *public SortedList(IComparer<TK> comparer)* - с помощью параметра comparer способ сравнения объектов, содержащихся в списке

Методы, помимо тех, что уже объявлены в интерфейсах, которые в нем реализуются:

- *Add()* Добавляет в список пару "ключ-значение". Если ключ уже находится в списке, то его значение не изменяется, и генерируется исключение ArgumentException
- *ContainsKey()*
  Возвращает логическое значение true, если вызывающий список содержит объект key в качестве ключа; а иначе — логическое значение false
- *ContainsValue()*
  Возвращает логическое значение true, если вызывающий список содержит значение value; в противном случае — логическое значение false
- *GetEnumerator()*
  Возвращает перечислитель для вызывающего словаря
- *IndexOfKey(), IndexOfValue()*
  Возвращает индекс ключа или первого вхождения значения в вызывающем списке. Если искомый ключ или значение не обнаружены в списке, возвращается значение -1.
- *Remove()*
  Удаляет из списка пару "ключ-значение" по указанному ключу key. При удачном исходе операции возвращается логическое значение true, а если ключ key отсутствует в списке — логическое значение false.
- *TrimExcess()*
  Сокращает избыточную емкость вызывающей коллекции в виде отсортированного списка.

Свойства, помимо тех, что уже объявлены в интерфейсах, которые в нем реализуются:

- *Capacity*
  Получает или устанавливает емкость вызывающей коллекции в виде отсортированного списка
- *Comparer*
  Получает метод сравнения для вызывающего списка
- *Keys*
  Получает коллекцию ключей
- *Values*
  Получает коллекцию значений

Реализуется **индексатор**, определенный в интерфейсе IDictionary<TKey, TValue>:
public TValue this[TKey key] { get; set; }

Этот индексатор служит для получения и установки значения элемента коллекции, а также для добавления в коллекцию нового элемента. Но в данном случае в качестве индекса служит ключ элемента, а не сам индекс.