<https://learn.microsoft.com/ru-ru/dotnet/api/system.collections.ilist?view=net-8.0>

// public interface IList : ICollection

//{

// int Add(object value);

// void Clear();

// bool Contains(object value);

// int IndexOf(object value);

// void Insert(int index, object value);

// void Remove(object value);

// void RemoveAt(int index);

// bool IsFixedSize { get; }

// bool IsReadOnly { get; }

// object this[int index] { get; set; }

//}

1. **Основные виды/типы коллекций .NET:**
   * **Массивы (Arrays):** Представляют собой фиксированную структуру данных, хранящую элементы одного типа. Быстрые для доступа по индексу, но не гибкие в изменении размера.
   * **Коллекции в пространстве имен System.Collections:** Классические коллекции, не поддерживающие обобщения (например, ArrayList, Hashtable).
   * **Generic-коллекции (System.Collections.Generic):** Обобщенные коллекции, которые работают с параметрами типа, обеспечивая безопасность типов (например, List<T>, Dictionary<TKey, TValue>).
   * **Коллекции для многопоточности (System.Collections.Concurrent):** Поддерживают безопасную работу с данными в многопоточной среде (например, ConcurrentBag<T>, ConcurrentDictionary<TKey, TValue>).
   * **Наблюдаемые коллекции (System.Collections.ObjectModel):** Позволяют отслеживать изменения в коллекции (например, ObservableCollection<T>).
   * **Специализированные коллекции (System.Collections.Specialized):** Коллекции для специфических задач (например, StringCollection, NameValueCollection).
2. **Что такое generic-коллекции?** Generic-коллекции — это типизированные коллекции, которые позволяют избежать ошибок времени выполнения, связанных с приведением типов. Они повышают производительность, так как не требуют упаковки и распаковки значений.

**Примеры:**

* + List<T>
  + Dictionary<TKey, TValue>
  + Queue<T>
  + HashSet<T>
  + Stack<T>

1. **Разница между ArrayList и Array:**
   * **Array:** Это базовый массив с фиксированным размером. Все элементы имеют один и тот же тип.
   * **ArrayList:** Коллекция, которая может изменять свой размер динамически. Поддерживает элементы любого типа (object), что делает её менее производительной и небезопасной с точки зрения типов по сравнению с обобщёнными коллекциями.
2. **Коллекции, использованные в проекте:**
   * **List<T>:** Обобщённая коллекция для хранения элементов, поддерживающая произвольный доступ по индексу.
   * **Dictionary<TKey, TValue>:** Обобщённая коллекция пар "ключ-значение", обеспечивающая быструю выборку по ключу.
   * **Queue<T>:** Очередь (FIFO), удобна для обработки данных в порядке их поступления.
   * **ObservableCollection<T>:** Наблюдаемая коллекция, позволяющая отслеживать изменения для обновления UI.
3. **Особенности коллекций System.Collections.Concurrent:**
   * Эти коллекции специально разработаны для многопоточной среды.
   * Обеспечивают потокобезопасность без необходимости ручной синхронизации.
   * Примеры: ConcurrentBag<T>, ConcurrentQueue<T>, ConcurrentDictionary<TKey, TValue>.
4. **Пространство имен для generic-коллекций:** Для использования обобщённых коллекций необходимо подключить пространство имен:

using System.Collections.Generic;

1. **Наблюдаемая коллекция:**
   * Это коллекция, реализующая интерфейс INotifyCollectionChanged, который позволяет отслеживать изменения (добавление, удаление элементов).
   * Пример: ObservableCollection<T> из пространства имен System.Collections.ObjectModel.
   * Используется, например, в WPF или .NET MAUI для обновления интерфейса пользователя при изменении данных.
2. **Интерфейсы IEnumerator и IEnumerable:**
   * **IEnumerable:** Позволяет перебирать элементы коллекции с помощью foreach. Определяет метод GetEnumerator(), который возвращает IEnumerator.
   * **IEnumerator:** Отвечает за итерацию по коллекции. Предоставляет методы MoveNext() и Reset() и свойство Current.
   * **Различие:** IEnumerable предоставляет итератор, а IEnumerator реализует сам процесс итерации.
3. **Принцип работы коллекций:**
   * **LinkedList<T>:** Двусвязный список. Элементы связаны ссылками, что обеспечивает быструю вставку и удаление, но замедляет доступ по индексу.
   * **HashSet<T>:** Коллекция уникальных элементов. Использует хэш-таблицу для быстрого поиска.
   * **Dictionary<TKey, TValue>:** Ассоциативный массив, хранящий пары "ключ-значение". Быстрый доступ к значениям по ключу.
   * **ConcurrentBag<T>:** Коллекция для многопоточной среды, предоставляющая поддержку добавления/извлечения элементов в произвольном порядке.
   * **Stack<T>, Queue<T>:**
     + Stack<T> — реализация стека (LIFO).
     + Queue<T> — реализация очереди (FIFO).
   * **SortedList<T>:** Коллекция, автоматически сортирующая элементы по ключу. Реализована как массив пар "ключ-значение".