**Вопросы к лабораторной работе №9:**

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

- наличие разнотипных данных

- ссылки на данные типа object (не обеспечивают типовую безопасность)

- System.Collections

* + **Обобщенные;**

- обеспечивают обобщенную реализацию нескольких стандартных структур

данных, включая связные списки, стеки, очереди и словари

- типизированные в силу своего обобщенного характера (обеспечивают типовую безопасность)

- System.Collections.Generic

* + **Специальные;**

**-** оперируют данными конкретного типа или же делают это каким-то особым образом

- System.Collections.Specialized

* + **С поразрядной организацией (поразрядная коллекция);**

- BitArray

- System.Collections

* + **Параллельные;**

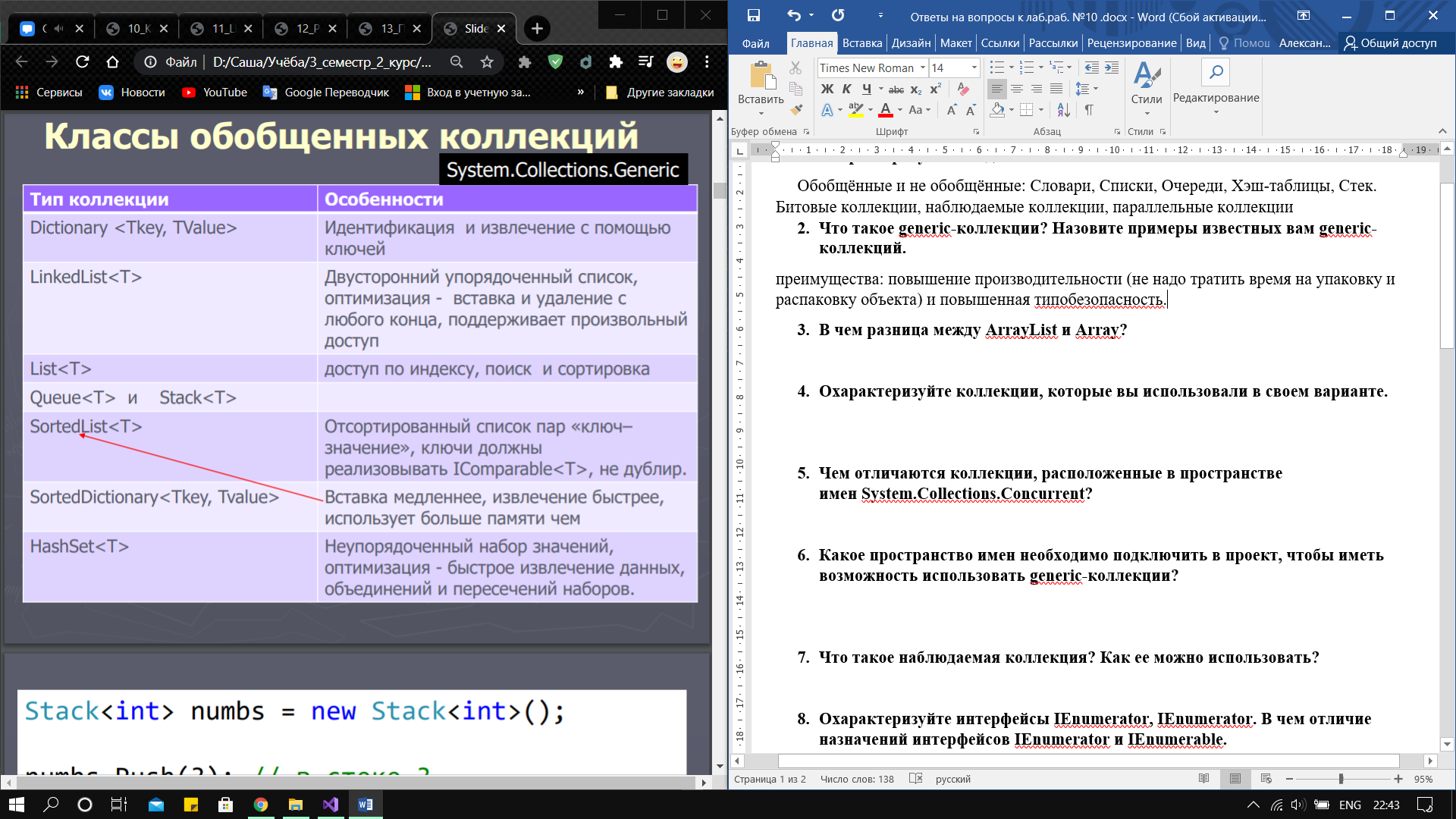
- многопоточный доступ к коллекции

- System.Collections.Concurrent

Обобщённые и необобщённые: Словари, Списки, Очереди, Хэш-таблицы, Стек.

1. **Что такое Generic-коллекции? Назовите примеры известных вам generic-коллекций.**

**Обобщённые коллекции** – это те же самые обобщённые классы (классы, которые изменяют своё поведение в зависимости от приписываемого им типа). Их использование перед необобщёнными коллекциями имеет те же преимущества: повышение производительности и повышенная типобезопасность.



1. **В чем разница между ArrayList и Array?**

Класс **ArrayList** определяется как массив переменной длины, который состоит из ссылок на объекты и может динамически увеличивать и уменьшать свой размер. Это список, каждый элемент связан друг с другом через указатели.

**Array** – это массив, представляющий из себя низкоуровневую структуру данных, которая по существу отображает область в памяти и имеет фиксированный размер.

Основное различие между Array и ArrayList состоит в том, что **Array является структурой данных фиксированной длины, а ArrayList является классом Collection переменной длины**.

1. **Охарактеризуйте коллекции, которые вы использовали в своем варианте.**

HashSet<T>

**HashSet <T> -** Сохраняет ряд уникальных значений, используя хеш-таблицу.

* содержит неповторяющюся коллекцию элементов.
* содержит элементы неупорядоченно (без сортировки).
* позволяет быстро определить, есть такой элемент или нет (быстро потому что, использует индекс, который вычисляется из хэш-кода элемента).
* имеет методы Add, Remove, Contains, но поскольку он использует хэш-реализацию, эти операции занимают 1 действие (методы Contains и Remove в [List<T>](https://dir.by/developer/csharp/list) занимает n-действий.)
* имеет методы:   
  UnionWith (объединение элементов с другим HashSet<T>)   
  IntersectWith (пересечение элементов с другим HashSet<T>)   
  ExceptWith (разность элементов с другим HashSet<T>)   
  SymmetricExceptWith (симетрическая разность элементов с другим HashSet<T>)
* относится к [типизированной коллекции...](https://dir.by/developer/csharp/collection_one_type) то есть все элементы одного типа

**Queue<T>**

**Очередь (queue)** — это коллекция, в которой элементы обрабатываются по схеме "первый вошел, первый вышел" (first in, first out — FIFO).

Члены класса Queue<T> представлены ниже:

* Count: Свойство Count возвращает количество элементов в очереди.
* Enqueue(): Метод Enqueue() добавляет элемент в конец очереди.
* Dequeue(): Метод Dequeue() читает и удаляет элемент из головы очереди. Если на момент вызова метода Dequeue() элементов в очереди больше нет, генерируется исключение InvalidOperationException.
* Peek(): Метод Peek() читает элемент из головы очереди, но не удаляет его.
* TrimExcess(): Метод TrimExcess() изменяет ёмкость очереди. Метод Dequeue() удаляет элемент из очереди, но не изменяет её ёмкости. TrimExcess() позволяет избавиться от пустых элементов в начале очереди.

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

**Параллельные коллекции** - коллекции классов, предназначенные для безопасной работы в многопоточной среде, которыми можно воспользоваться при создании многопоточных приложений.

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

System.Collections.Generic

1. **Что такое наблюдаемая коллекция? Как ее можно использовать?**

**ObservableCollection<T>**

ObservableCollection-это **коллекция**, которая позволяет коду вне **коллекции** знать о том, когда происходят изменения в **коллекции** (добавление, перемещение, удаление)

В случае если нужна информация о том, когда элементы коллекции удаляются или добавляются, можно использовать класс ObservableCollection<T>. Этот класс предназначен для того, чтобы пользовательский интерфейс мог получать информацию об изменениях коллекции. 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>

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

Коллекция, которая реализует IEnumerable, может использоваться с оператором foreach.

IEnumerator – вещь, которая может перечислять: имеет свойство Current и методы MoveNext и Reset.

IEnumerable – это вещь, которую можно перечислить. У неё есть метод GetEnumerator, который возвращает IEnumerator.

Интерфейс ***IEnumerable*** имеет метод GetEnumerator(), возвращающий ссылку на другой интерфейс IEnumerator – перечислитель. Коллекция, которая реализует данный интерфейс, может использоваться с оператором foreach.

А интерфейс ***IEnumerator*** определяет функционал для перебора внутренних объектов в контейнере. Этот интерфейс может перечислять: имеет свойство Current и методы MoveNext() и Reset().

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

}

1. **Поясните принцип работы коллекций:**
2. **LinkedList <T> -** Сохраняет элементы в двунаправленном списке.
3. **HashSet <T> -** Сохраняет ряд уникальных значений, используя хеш-таблицу.
4. **Dictionary <Tkey, TValue> -** Сохраняет пары "ключ-значение". Обеспечивает такие же функциональные возможности, как и необобщенный класс Hashtable.
5. **ConcurrentBag <Tkey, TValue> - Представляет потокобезопасную неупорядоченную коллекцию объектов.** Этот класс не определяет никакого порядка для добавления или извлечения элементов. Он реализует концепцию отображения потоков на используемые внутренние массивы. Для доступа к элементам применяются методы Add(), TryPeek() и TryTake().
6. **Stack<t>,** представляет коллекцию, которая использует алгоритм LIFO ("последний вошел - первый вышел").
7. **Queue<T> -** представляет обычную очередь, которая работает по алгоритму FIFO ("первый вошел - первый вышел").
8. **SortedList -** хранит наборы пар ключ-значение, отсортированные по ключу.