|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики»  *Факультет социально-экономических и компьютерных наук* |
|  |
| Ханжин Александр Евгеньевич  **ЛАБОРАТОРНАЯ РАБОТА №11**  студента образовательной программы «Разработка информационных систем для бизнеса» по направлению подготовки *09.03.04 Программная инженерия*   |  |  | | --- | --- | |  | Старший преподаватель кафедры ИТБ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Марквирер В.Д. |   Пермь, 2024 |

**Оглавление**

[Задача 1 3](#_Toc153017737)

[1.1 Постановка задачи 3](#_Toc153017738)

[1.2 Анализ 3](#_Toc153017739)

[1.3 Программа (листинг) 6](#_Toc153017740)

[1.4 Тестирование программы 7](#_Toc153017741)

# Задача 1

## Диаграмма классов

Диаграмма классов является одним из основных инструментов объектно-ориентированного моделирования и позволяет описать архитектуру программного обеспечения. Она помогает разработчикам понять, какие классы и объекты присутствуют в системе, а также как они взаимодействуют между собой. Диаграмма классов лабораторной работы представлена на рисунке 1.1.1



***Рис. 1.1.1. Диаграмма классов.***

## Описание методов первой части лабораторной работы

В первой части лабораторной работы я работал с коллекцией Queue. Сама коллекция представляет из себя очередь данных, работающую по системе FIFO (first in first out). Очередь является динамической коллекцией, которая при необходимости увеличивается, чтобы принять для хранения новые элементы, причем каждый раз, когда такая необходимость возникает, текущий размер очереди умножается на коэффициент роста, который по умолчанию равен значению 2.0.

Во время выполнения работы я изучил коллекцию Queue и постарался применить в своей работе как можно больше разных методов:

1. Enqueue(T item): Данный метод добавляет элемент в конец очереди. Он принимает аргумент типа T (тип элементов в очереди) и помещает его в конец очереди. Если очередь ранее была пуста, этот элемент становится первым.
2. Dequeue(): Метод Dequeue извлекает и возвращает элемент из начала очереди. После извлечения элемента, он удаляется из очереди. Если очередь пуста, будет сгенерировано исключение InvalidOperationException.
3. Peek(): Метод Peek возвращает элемент из начала очереди без его удаления. Этот метод полезен, когда нужно просмотреть следующий элемент в очереди, но не извлекать его.
4. Clear(): Метод Clear удаляет все элементы из очереди, оставляя ее пустой. После вызова этого метода, Count очереди будет равен нулю.
5. Contains(T item): Метод Contains проверяет наличие указанного элемента в очереди. Он принимает аргумент типа T и возвращает true, если элемент найден, и false в противном случае.
6. ToArray(): Метод ToArray преобразует содержимое очереди в массив элементов типа T. Этот метод полезен, если требуется работать с элементами очереди как с массивом. Этот метод мне пригодился, когда мне нужно было реализовать сортировку типа данных, не предполагающую ее.

## Описание методов второй части лабораторной работы

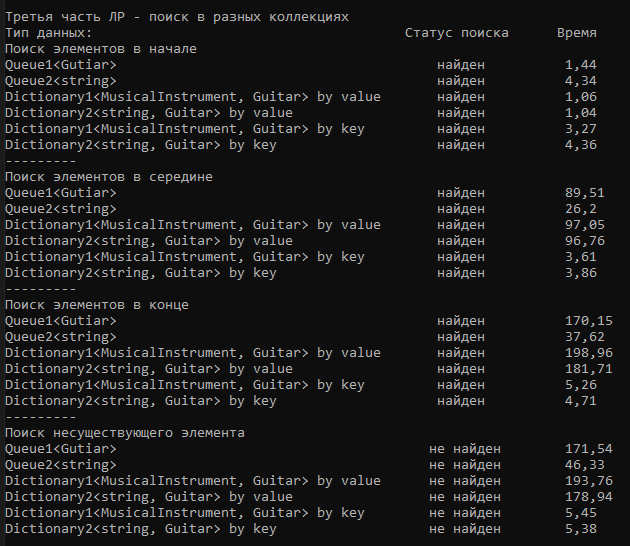
В второй части лабораторной работы я работал с коллекцией Stack. Стек (Stack) - это коллекция данных, которая работает по принципу LIFO (last in first out), что означает, что последний добавленный элемент будет первым извлеченным. Стек также является динамической коллекцией, которая увеличивается по мере необходимости для хранения новых элементов. При добавлении нового элемента в стек, его размер увеличивается, то есть его текущий размер умножается на коэффициент роста. Коэффициент роста по умолчанию для стека может быть различным в зависимости от реализации, но обычно он равен значению 2.0 или другому заранее определенному значению.

Во время выполнения работы я изучил коллекцию Stack и постарался применить в своей работе как можно больше разных методов:

1. Push(T item): Метод Push добавляет элемент типа T в вершину стека. Этот элемент становится последним добавленным и будет первым извлеченным при вызове метода Pop().
2. Pop(): Метод Pop извлекает и удаляет элемент из вершины стека. Этот элемент был добавлен последним с помощью метода Push() и теперь будет удален из стека.
3. Peek(): Метод Peek позволяет просмотреть элемент на вершине стека, не удаляя его. Этот метод полезен, если вы хотите узнать значение верхнего элемента без его удаления.
4. Clear(): Метод Clear очищает стек от всех элементов, удаляя их и возвращая стек к пустому состоянию. После вызова этого метода стек будет не содержать ни одного элемента.
5. Contains(T item): Метод Contains проверяет наличие указанного элемента в стеке. Если элемент присутствует в стеке, метод вернет true, в противном случае - false.
6. ToArray(): Метод ToArray преобразует содержимое стека в массив элементов типа T. Это позволяет легко получить доступ ко всем элементам стека в виде массива.

## Полученные результаты в третьей части лабораторной работы

В третьей части лабораторной работы я проводил тестирование поиска элементов в разных типах данных. Для этого был создан отдельный класс, в котором находятся 2 разные коллекции разными параметрами класса. Для подсчёта средних значений времени, поиск элементов производился много раз, и бралось полученное среднее значение.



***Рис. 1.4.1. Полученные результаты работы программы.***

Из полученных данных видно, что очередь и словарь, сортируемый по значениям, имеют сложность аналогичную сложность поиска, возрастающую от позиции элемента (приходится перебирать больше значений для нахождения нужного элемента). Это происходит так как поиск в очереди и словаре по значениям выполняется O(n). В словаре, сортируемом по ключам, абсолютно противоположная ситуация, скорость поиска не зависит от позиции элемента, так в словаре сложность поиска O(1). Сложность поиска несуществующего элемента в очереди и словаре с сортировкой по ключам равна сложности нахождения последнего элемента, так как приходится перебрать все значения, чтобы убедиться, что элемента нет в коллекции.

## Программа (листинг)

Полный листинг программы можно найти на вебсайте github по ссылке: https://github.com/Asklit/lab11.