Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа «Компьютерных технологий и информационных систем»

**ОТЧЕТ по лабораторной работе №6**

по дисциплине «Java»

**Выполнил:**

студент группы 5130902/20201 \_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. Сафонов

подпись

**Проверил:**

Доцент, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_ С. А. Нестеров

подпись

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

Санкт-Петербург, 2025

Оглавление

[1. 3.3 по слайдам 3](#_Toc192461659)

[1. Практика 3.3 5](#_Toc192461660)

[Практика 3.4 8](#_Toc192461661)

1. 3.3 Collections – Part 2

Сортировка по фамилии

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Сортировка по оценке

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

1. Практика 3.3

Изображение выглядит как текст, Шрифт, линия, число

Контент, сгенерированный ИИ, может содержать ошибки.

Deque

Node

Collection

HashMap

ArrayList

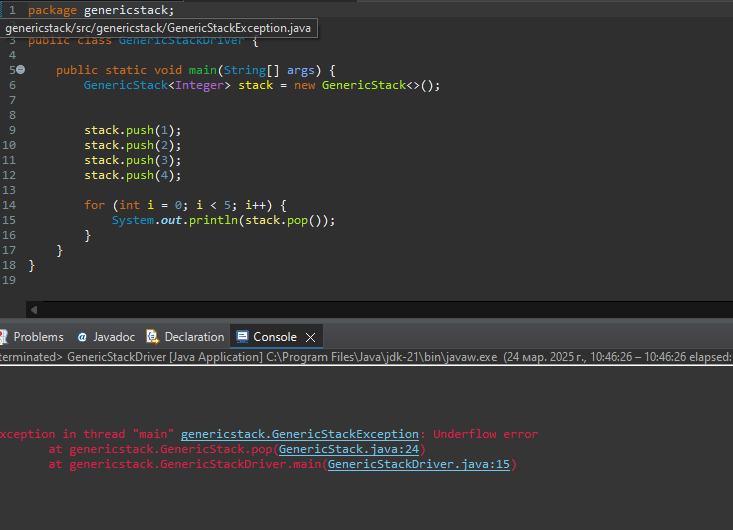
Queue

Отличия stack, queue:

Stack: LIFO (last in first out), пример – блинчики на тарелке. Первым будет съеден тот, что на верхушке, то есть положенный последним.

Queue: FIFO (first in first out) пример – очередь в библиотеку. Первый пришедший уйдет первым.





Добавление нода в начало связного списка – addFirst(). В конец – add()

Цель реализации Comparable – сортировка объектов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Практика 3.4

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.



Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

1. **Linear Search**
2. **Selection Sort**
3. **Bubble Sort**
4. **Binary Search**
5. **Merge Sort**
6. **ASCII Sort Order**

**3) Разница между линейным и бинарным поиском**:

* **Линейный поиск**: Это самый простой алгоритм поиска. Он последовательно проверяет каждый элемент в списке или массиве, пока не найдет целевое значение или не дойдет до конца коллекции. Время работы алгоритма — O(n), где n — количество элементов.
  + **Преимущества**: Легко реализуется, работает с неотсортированными данными.
  + **Недостатки**: Нееффективен для больших наборов данных, так как требует проверки каждого элемента.
* **Бинарный поиск**: это более эффективный алгоритм поиска, который работает с отсортированными данными. Он многократно делит диапазон поиска пополам и сравнивает целевое значение с элементом в середине текущего диапазона. Если целевое значение меньше, поиск продолжается в левой части, если больше — в правой. Время работы — O (log n), где n — количество элементов.
  + **Преимущества**: значительно быстрее линейного поиска для больших наборов данных (требует меньше сравнений).
  + **Недостатки**: требует, чтобы данные были отсортированы заранее.

**4) Порядок сортировки строк и чисел**:

При сортировке данных, содержащих как строки, так и числа, Java обычно сортирует их по лексикографическому порядку для строк и числовому порядку для чисел. Поведение сортировки может зависеть от конкретной реализации и структуры данных. Вот как это работает:

* **Для чисел**: Числовые значения сортируются по возрастанию их значения. Например, в порядке возрастания: 1, 5, 10, 20.
* **Для строк**: Строки сортируются лексикографически (по алфавиту). Это означает, что строки сравниваются посимвольно по их Unicode-значениям. Например, "apple", "banana" и "cherry" будут отсортированы по алфавиту.

**Смешанная сортировка (строки и числа)**: Если в коллекции содержатся как строки, так и числа, и используется стандартная сортировка, например, Collections.sort(), Java будет сортировать их в их естественном порядке, при этом числа обычно будут располагаться перед строками, так как их примитивные значения сравниваются напрямую, а строки — лексикографически. Если необходимо изменить это поведение, можно реализовать собственный компаратор для определения порядка сортировки смешанных типов данных.