Министерство высшего образования и науки РФ

Санкт–Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчет**

по дисциплине «Системный подход в разработке ПО»

**Basics of Input and Output**

**Выполнил:**

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

подпись

**Проверил:**

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

подпись

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

Санкт-Петербург

2025г.

Оглавление

[1. Integral 3](#_Toc196138339)

[2. 5-1 Basics of Input and Output 4](#_Toc196138340)

[3. Стивенс Р. Алгоритмы. Теория и практическое применение. - М:Издательство "Э", 2016. 7](#_Toc196138341)

[4. Васильев, А. Н. Программирование на Java для начинающих 16](#_Toc196138342)

# Integral

**Задание**

Находим пример из книги Васильев, А. Н. Программирование на Java для начинающих на вычисление интегралов с. 381-385 (глава 10 Лямбда-выражения). Там интеграл считается по формуле трапеций. К этому примеру дописываем два метода, считающие интеграл по формуле прямоугольников и методом Монте-Карло.

**Решение**

**Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

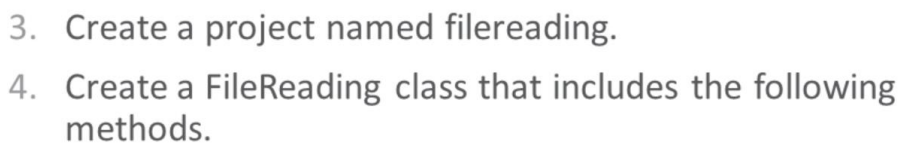
Автоматически созданное описание**

# 5-1 Basics of Input and Output

**Задание**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

****

**Изображение выглядит как текст, Шрифт, белый, алгебра

Автоматически созданное описание**

**Изображение выглядит как текст, Шрифт, белый, алгебра

Автоматически созданное описание**

**Изображение выглядит как текст, Шрифт, чек, белый

Автоматически созданное описание**

**Изображение выглядит как текст, Шрифт, белый, чек

Автоматически созданное описание**

**Изображение выглядит как текст, Шрифт, белый, чек

Автоматически созданное описание**

**Изображение выглядит как текст, Шрифт, белый, алгебра

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Автоматически созданное описание**

**Решение**

**Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание**

**Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание**

****

****

# Стивенс Р. Алгоритмы. Теория и практическое применение. - М:Издательство "Э", 2016.

**1. Введение в структуры данных**

Стеки и очереди — это базовые структуры данных, которые используются для временного хранения элементов в определённом порядке. Они являются фундаментом для многих алгоритмов и применяются в таких областях, как:

* Управление памятью (стек вызовов).
* Синтаксический анализ (парсинг выражений).
* Алгоритмы поиска (например, DFS и BFS).
* Симуляция реальных процессов (очереди задач).

**2. Стеки (LIFO)**

**Определение:**  
Стек — это структура данных, в которой элементы добавляются и удаляются по принципу **"последним пришёл — первым вышел"** (Last In, First Out, LIFO). Основные операции:

* **Push** — добавление элемента на вершину стека.
* **Pop** — удаление элемента с вершины стека.
* **Peek** — просмотр верхнего элемента без удаления.

**Примеры использования:**

* История браузера (кнопки "Назад" и "Вперёд").
* Отмена действий в текстовых редакторах (Ctrl+Z).
* Рекурсивные вызовы функций.

**Реализации:**

1. **На массиве:**
   * Плюсы: быстрый доступ, простота реализации.
   * Минусы: фиксированный размер, необходимость расширения.
2. **На связном списке:**
   * Плюсы: динамическое расширение, эффективное использование памяти.
   * Минусы: дополнительные затраты на хранение ссылок.

**Стек на связном списке:**

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

Автоматически созданное описание

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

Автоматически созданное описание

**3. Очереди (FIFO)**

**Определение:**  
Очередь — это структура данных, в которой элементы добавляются в конец и удаляются из начала по принципу **"первым пришёл — первым вышел"** (First In, First Out, FIFO). Основные операции:

* **Enqueue** — добавление элемента в конец очереди.
* **Dequeue** — удаление элемента из начала очереди.
* **Peek** — просмотр первого элемента без удаления.

**Примеры использования:**

* Очереди задач в операционных системах.
* Обработка запросов на сервере.
* Буферизация данных.

**Реализации:**

1. **На массиве (кольцевая очередь):**
   * Плюсы: эффективное использование памяти.
   * Минусы: ограниченный размер, сложность управления индексами.
2. **На связном списке:**
   * Плюсы: динамическое расширение.
   * Минусы: дополнительные затраты на хранение ссылок.

**Кольцевая очередь на массиве:**

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

Автоматически созданное описание

Изображение выглядит как текст, Мультимедийное программное обеспечение, программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание

**4. Специализированные структуры**

**Очереди с приоритетом:**

* Элементы обрабатываются в порядке их приоритета.
* Реализации:
  + На основе массива с сортировкой (вставка — O(N)*O*(*N*), удаление — O(1)*O*(1)).
  + На основе кучи (вставка и удаление — O(log⁡N)*O*(log*N*)).

**Деки (двусторонние очереди):**

* Поддерживают добавление и удаление с обоих концов.
* Примеры:
  + История посещений в браузере.
  + Алгоритмы скользящего окна.

**5. Алгоритмы на стеках и очередях**

**Реверсирование данных:**

* Используется стек для временного хранения элементов.
* Пример: реверсирование строки или массива.

Реверсирование массива через стек:

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

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

**Сортировка:**

* **Сортировка вставкой для стека:**  
  Временная сложность — O(N2)*O*(*N*2).

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

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

* **Сортировка выбором для стека:**  
  Также O(N2)*O*(*N*2), но с другим подходом к поиску элементов.

**Ханойская башня:**

* Классическая задача на рекурсию и стеки.
* Решение требует 2N−12*N*−1 шагов для N*N* дисков.

**6. Заключение**

Стеки и очереди — это мощные инструменты для управления данными в алгоритмах. Их правильное применение позволяет оптимизировать многие процессы, от системных задач до сложных вычислений. Понимание этих структур — ключ к эффективному программированию.

**Упражнения**

****

Когда один из стеков полный, NextIndex1>NextIndex2. В этот момент заполняются оба стека, NextIndex1 становится индексом верхнего элемента второго стека, а NextIndex2 — первого.



Stack reverseStack(Stack input) {

Stack temp = new Stack();

while (!input.isEmpty()) temp.Push(input.Pop());

return temp;

}



void StackInsertionSort(Stack items) {

Stack temp = new Stack();

int n = items.size();

for (int i = 0; i < n; i++) {

Data next = items.Pop();

// Перенос элементов во временный стек и вставка next

while (!items.isEmpty() && items.peek() > next)

temp.Push(items.Pop());

items.Push(next);

while (!temp.isEmpty()) items.Push(temp.Pop());

}

}

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

На самом деле алгоритму не обязательно возвращать все неотсортированные элементы в исходный стек, поскольку он все равно будет брать их оттуда и вставлять согласно порядку сортировки. Вместо этого вы можете вернуть в начальный стек отсортированные записи и затем уже вставлять туда остальные элементы. Это немного ускорит алгоритм, но время его работы так или иначе будет равно O(N2 ).



Тот факт, что алгоритм сортировки вставкой работает, говорит о том, что вагоны поезда можно отсортировать с использованием всего двух колей — накопительной и выходной. Накопительная колея может служить в качестве второго стека, выводя на выходную те вагоны, которые вы в данный момент сортируете (или наоборот). Это потребует больше шагов по сравнению с ситуацией, в которой накопительная колея не одна. Имеет смысл увеличить количество таких колей, поскольку перемещение железнодорожных вагонов происходит куда медленнее, чем перетягивание элементов из стека в стек.



* 1. void StackSelectionSort(Stack items) {

Stack temp = new Stack();

int n = items.size();

for (int i = 0; i < n; i++) {

Data max = items.Pop();

// Поиск максимума

for (int j = i + 1; j < n; j++) {

Data current = items.Pop();

if (current > max) {

temp.Push(max);

max = current;

} else temp.Push(current);

}

items.Push(max);

while (!temp.isEmpty()) items.Push(temp.Pop());

}

}



Внеся небольшие изменения, вы сможете применить алгоритм сортировки выбором для упорядочивания железнодорожных вагонов. Версия, описанная в главе 5, хранит самый крупный элемент в отдельной переменной. Во время сортировки вагонов нельзя просто отложить некоторые из них в сторону. Однако его можно отправить на выходящий путь, а остальные сохранить на накопительной колее. Найдя вагон с наибольшим номером, вы без труда вернете его обратно на накопительную колею, переместив на выходной путь новый вагон. Естественно, в случае с реальными поездами вам не обязательно ограничиваться передним вагоном. Прежде чем начинать какие-то перемещения, вы можете пройтись вдоль неупорядоченного состава и посмотреть, какой вагон имеет самый большой номер. Когда такой найдется, его можно будет выкатить на выходной путь, а остальные вагоны переместить на накопительную колею. Это избавит вас от необходимости выводить на выходной путь не те вагоны и сократит длинный процесс упорядочивания.

PriorityQueue<Integer> pq = new PriorityQueue<>();

pq.add(3); pq.add(1); pq.add(2);

System.out.println(pq.poll()); // 1



class Deque {

private List<Data> items = new ArrayList<>();

void addFirst(Data value) { items.add(0, value); }

void addLast(Data value) { items.add(value); }

Data removeFirst() { return items.remove(0); }

Data removeLast() { return items.remove(items.size() - 1); }

}

# Васильев, А. Н. Программирование на Java для начинающих

1.1. Синтаксис

лямбда - выраженияЛямбда-выражение можно рассматривать как некоторый блок

кода, определяющий действие. Также можно рассматривать лямбда-выражение какопределение анонимного метода. Используются лямбда-выражения в основном как

альтернатива к созданию анонимных классов.

Блок из команд в лямбда-выражении для удобства будем

называть телом лямбда-выражения. Лямбда-выражение состоит из аргументов и тела.То есть здесь все очень близко к тому, как описывается метод, но только нет

названия и идентификатора типа результата.

Пример: (int x,int y)->{

int z=x+y;

System.out.println(z);

}

Есть некоторые правила, которые позволяют упрощать синтаксисописания лямбда-выражений. Вот они.

• Тип аргумента можно не указывать, если он быть

идентифицируется исходя из контекста команды использования лямбда-выражения.

• Если в лямбда-выражении один аргумент, тип которого неуказан, то круглые скобки можно не использовать.

• Если в лямбда-выражении нет аргументов, то используются

пустые круглые скобки.

• Если в теле лямбда-выражения всего одна команда, тофигурные скобки можно не использовать. В случае, когда единственная команда в

теле лямбда-оператора состоит из return-инструкции, ключевое слово return можноне использовать.

1.2. Функциональный интерфейс

Функциональный интерфейс — это интерфейс с одним и толькоодним абстрактным методам. То есть ничего особенного в функциональном

интерфейсе нет. Просто должно быть выдержано одно условие: в интерфейсеобъявляется всего один абстрактный метод.

Особенность функционального интерфейса в плане использования

лямбда-выражений связана с тем, что лямбда-выражение может быть присвоенозначением переменной интерфейсного типа — но при условии, что интерфейс

функциональный, а сигнатура абстрактного метода в функциональном интерфейсесоответствует параметрам лямбда-выражения (количество и тип параметров).

Одинаковые лямбда-выражения могут присваиваться переменным

разных функциональных интерфейсов. Достаточно, чтобы абстрактный метод в каждомтаком интерфейсе соответствовал параметрам (таким, как количество и тип

аргументов) лямбда-выражения.

1.3. Ссылка на метод объекта Чтобы понять принципы выполнения и использования ссылок на

методы, удобно представлять, что значением выражения со ссылкой на методявляется инструкция, аналогичная лямбда-выражению, которое соответствует коду

метода (на который выполняется ссылка). Проще говоря, ссылка на метод можетрассматриваться как лямбда-выражение, соответствующее коду метода, на который

выполняется ссылка.

Если речь идет о выполнении ссылки на нестатический методконкретного объекта, то выполняется она просто (и пример выполнения такой

ссылки мы видели): указывается имя объекта, и через оператор :: указываетсяназвание метода. То есть шаблон выполнения ссылки на нестатический метод

следующий : объект::имя\_метода.

Если ссылка на объектвыполнена в формате объект::метод, и при этом метод описан с некоторыми параметрами,

то ссылке объект::метод соответствуетлямбда-выражение вида аргументы à объект.метод(аргументы)

1.4. Ссылка на нестатический метод класса

Ссылка может выполняться не только на метод объекта, но и наметод класса. В последнем случае имеет значение, является ли метод статическим,

или нет. Сначала рассмотрим случай, когда метод нестатический. Ссылка нанестатический метод некоторого класса выполняется в следующем виде:

Класс:имя\_метода

Правило такое:

Если используется ссылка вида класс::метод, и при этом метод —нестатический и описан с некоторыми аргументами, то указанная ссылка

эквивалентна лямбда-выражению следующего вида:

(объект, аргументы) -> объект.метод(аргументы)

Более конкретно:

Допустим, что имеется класс MyClass, и в нём описан некоторыйнестатический метод method() с аргументом типа int и типа char.

Тогда ссылке на метод MyClass::method соответствует лямбда-выражениевида:

(MyClass obj, int num, char symb) -> { код\_метода\_method() }

Поэтому если мы хотим ссылку MyClass::method присвоить

интерфейсной переменной, то в этом функциональном интерфейсе абстрактный методдолжен быть объявлен с тремя аргументами соответственно типа MyClass, int и char.

1.5. Ссылка на статический метод класса

Ссылка на статический метод класса формально выполняется так же, каки на нестатический метод, но её логика отличается.

Правило:

Если используется ссылка вида класс::метод, и при этом метод является статическим и

принимает некоторые аргументы, то такая ссылка эквивалентналямбда-выражению вида:

(аргументы) -> класс.метод(аргументы))

1.6. Ссылка на конструктор класса

Ссылку можно выполнять не только на метод, но и на конструкторкласса.

Синтаксис:

После имени класса указывается оператор ::, за которым следует ключевое

слово new. Вся конструкция записывается в виде:

класс::new

Эквивалентная лямбда-запись:

Если использована ссылка на конструктор вида класс::new, то онасоответствует лямбда-выражению:

(аргументы) -> new класс(аргументы)

1.7. Ссылка

на перегруженный методВполне может сложиться ситуация, когда необходимо выполнить ссылку

на метод, который перегружен. В таком случае в классе описано несколько версийметода или, если быть более точными, то несколько методов с одинаковыми

названиями. Собственно по ссылке на метод нельзя однозначно определить, о какойверсии метода идет речь. Вывод о том, какую версию метода следует использовать,

делается на основе интерфейсной переменной, которой в качестве значенияприсваивается ссылка на метод

Лямбда-выражение может использоваться при определении результата

метода. Формальным результатом в таком случае является значение интерфейсноготипа, но собственно в коде метода результат может быть «оформлен» в виде

лямбда-выражения