|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования РФ  Федеральное государственное автономное образовательное учреждение высшего образования  «Пермский государственный национальный исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по лабораторной работе №1  по дисциплине «Языки программирования»  Вариант 1 | | |
|  | | Работу выполнил  студент группы ПМИ-1,2-2023 2 курса  Волегов И.С.  «24» Декабря 2024 г. |
| Работу проверил  Ракина В.Д.  «\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2024 | | |

СОДЕРЖАНИЕ

[Задание 1 4](#_Toc185892455)

[Текст задания 4](#_Toc185892456)

[Алгоритм решения 4](#_Toc185892457)

[Тестирование 4](#_Toc185892458)

[Код программы 4](#_Toc185892459)

[Задание 2 5](#_Toc185892460)

[Текст задания 5](#_Toc185892461)

[Алгоритм решения 5](#_Toc185892462)

[Тестирование 5](#_Toc185892463)

[Код программы 5](#_Toc185892464)

[Задание 3 5](#_Toc185892465)

[Текст задания 5](#_Toc185892466)

[Алгоритм решения 5](#_Toc185892467)

[Тестирование 5](#_Toc185892468)

[Код программы 6](#_Toc185892469)

[Задание 4 6](#_Toc185892470)

[Текст задания 6](#_Toc185892471)

[Алгоритм решения 6](#_Toc185892472)

[Тестирование 6](#_Toc185892473)

[Код программы 6](#_Toc185892474)

[Задание 5 6](#_Toc185892475)

[Текст задания 6](#_Toc185892476)

[Алгоритм решения 6](#_Toc185892477)

[Тестирование 7](#_Toc185892478)

[Код программы 7](#_Toc185892479)

[Задание 6 7](#_Toc185892480)

[Текст задания 7](#_Toc185892481)

[Алгоритм решения 7](#_Toc185892482)

[Тестирование 7](#_Toc185892483)

[Код программы 7](#_Toc185892484)

[Задание 7 8](#_Toc185892485)

[Текст задания 8](#_Toc185892486)

[Алгоритм решения 8](#_Toc185892487)

[Тестирование 8](#_Toc185892488)

[Код программы 8](#_Toc185892489)

[Задание 8 8](#_Toc185892490)

[Текст задания 8](#_Toc185892491)

[Алгоритм решения 8](#_Toc185892492)

[Тестирование 9](#_Toc185892493)

[Код программы 9](#_Toc185892494)

# Задание 1

## Текст задания

В класс Дробь, добавить интерфейс на два метода: получение вещественного значения, установка числителя и установка знаменателя. Сгенерировать такую версию дроби, которая будет кэшировать вычисление вещественного значения. Если раннее в вашем варианте не было Дроби, то создайте сущность Дробь со следующими особенностями:

• Имеет числитель: целое число

• Имеет знаменатель: целое число

• Дробь может быть создана с указанием числителя и знаменателя

• Может вернуть строковое представление вида “числитель/знаменатель”

• Необходимо корректно обрабатывать отрицательные значения. Учтите, что знаменатель не может быть отрицательным.

• Переопределите метод сравнения объектов по состоянию таким образом, чтобы две дроби считались одинаковыми тогда, когда у них одинаковые значения числителя и знаменателя.

## 

## Алгоритм решения

- Определение класса **Fraction**:

* Класс описывает дробь с числителем и знаменателем.
* Конструктор принимает числитель и знаменатель, проверяет, что знаменатель не равен нулю, и нормализует дробь (чтобы знаменатель был положительным).
* Реализуется метод для получения вещественного значения дроби, используя кэширование для ускорения повторных вычислений.
* Реализуются методы для изменения числителя и знаменателя, с обработкой ошибок (нельзя установить знаменатель равным нулю).
* Переопределяется метод toString для вывода дроби в формате числитель/знаменатель и метод equals для сравнения двух дробей, учитывая нормализацию.

- Интерфейс **FractionOperations**:

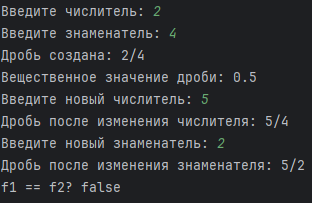
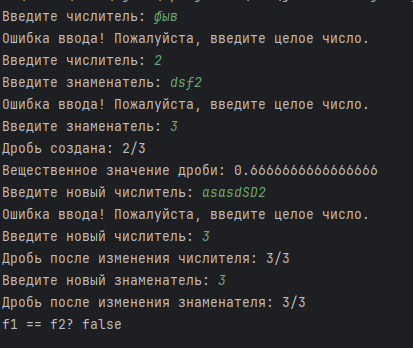
* Интерфейс содержит три метода: для получения вещественного значения дроби, установки числителя и установки знаменателя.

- Реализация в **main**:

* Ввод чисел с проверкой, что знаменатель не равен нулю.
* Создание объекта дроби с введёнными числителем и знаменателем.
* Вывод дроби и её вещественного значения.
* Изменение числителя и знаменателя, с обновлением дроби и повторным выводом.
* Сравнение двух дробей на равенство, с выводом результата.

.

## Тестирование

## Код программы

Ссылка на githab

# Задание 2

## Текст задания

Количество мяуканий. Необходимо воспользоваться классом Кот и методом принимающим всех мяукающих из задачи 2.5.4. Необходимо таким образом передать кота в указанный метод, что бы после окончания его работы узнать сколько раз мяукал кот за время его работы. На рисунке показан пример работы. Перед вызовом метода создаем кота, отправляем ссылку на кота в метод, после окончания его работы выводим количество мяуканий на экран. Кота изменять нельзя. Если раннее в вашем варианте не было Кота, то создайте 1. сущность Кот, которая описывается следующим образом:

• Имеет Имя (строка) • Для создания необходимо указать имя кота.

• Может быть приведен к текстовой форме вида: “кот: Имя”

• Может помяукать, что приводит к выводу на экран следующего текста: “Имя: мяу!”, вызвать мяуканье можно без параметров.

2. интерфейс Мяуканье: разработайте метод, который принимает набор объектов способных мяукать и вызывает мяуканье у каждого объекта. Мяукающие объекты должны иметь метод со следующей сигнатурой: public void meow();.

## Алгоритм решения

- **Класс Cat**:

* Описывает кота с полем name для имени кота.
* Реализует интерфейс Meowable, который требует наличия метода meow().
* В методе meow() кот "мяукает", выводя сообщение вида "Имя: мяу!".
* Переопределен метод toString() для вывода строки с именем кота.

- **Интерфейс Meowable**:

* Описывает контракт для объектов, которые могут "мяукать". Включает метод meow().

- **Класс MeowCounter**:

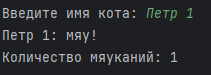
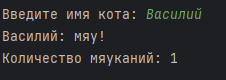
* Содержит переменную meowCount для подсчета количества мяуканий.
* Метод countMeows() принимает список объектов Meowable, вызывает у каждого метод meow() и увеличивает счетчик мяуканий.
* Метод getMeowCount() возвращает общее количество мяуканий.

- **Основной метод (main)**:

* Вводится имя кота с клавиатуры.
* Создается объект кота с введённым именем.
* Создается список meowables, в который добавляется объект кота.
* Создается объект MeowCounter, который подсчитывает количество мяуканий с помощью метода countMeows().
* Выводится общее количество мяуканий.

.

## Тестирование



## Код программы

Ссылка на githab

# Задание 3

## Текст задания

Составить программу, которая удаляет из списка L все элементы с указанным значением.

## 

## Алгоритм решения

- **Класс Item**:

* Хранит одно значение value, которое представляет элемент в списке.
* Имеет метод для получения значения элемента (getValue()).
* Переопределен метод toString() для корректного представления элемента как строки.

- **Интерфейс Removable**:

* Определяет метод removeElement(int value), который должен быть реализован для удаления элементов по значению.

- **Класс ItemList**:

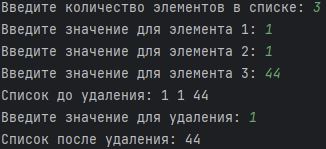
* Содержит список объектов типа Item.
* Реализует интерфейс Removable, позволяя удалять элементы по значению с использованием метода removeElement(int value).
* Метод removeElement использует Iterator для безопасного удаления элементов в процессе обхода списка.
* Переопределен метод toString() для представления всех элементов списка в строковом виде.

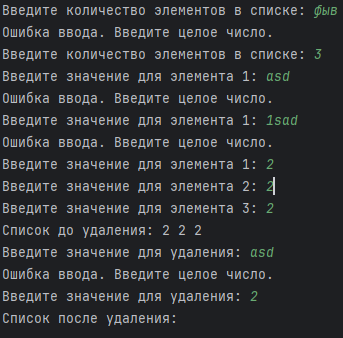
- **Основной метод main**:

* Вводит количество элементов для списка и значения этих элементов.
* Создает объект ItemList и добавляет в него элементы.
* Выводит список до и после удаления элементов с указанным значением.

.

## Тестирование





## Код программы

Ссылка на githab

# Задание 4

## Текст задания

В молочных магазинах города Х продается сметана с жирностью 15, 20 и 25 процентов. В городе X был проведен мониторинг цен на сметану. Напишите эффективную по времени работы и по используемой памяти программу, которая будет определять для каждого вида сметаны, сколько магазинов продают ее дешевле всего. На вход программе сначала подается число магазинов N. В каждой из следующих N строк находится информация в следующем формате: где – строка, состоящая не более, чем из 20 символов без пробелов, – строка, состоящая не более, чем из 20 символов без пробелов, – одно из чисел – 15, 20 или 25, – целое число в диапазоне от 2000 до 5000, обозначающее стоимость одного литра сметаны в копейках. и , и , а также и разделены ровно одним пробелом. Пример входной строки: Перекресток Короленко 25 120 Программа должна выводить через пробел 3 числа – количество магазинов, продающих дешевле всего сметану с жирностью 15, 20 и 25 процентов. Если какой-то вид сметаны нигде не продавался, то следует вывести 0. Пример выходных данных: 12 10 0.

## Алгоритм решения

1. **Класс Store**:
   * Содержит информацию о магазине: фирма, улица, жирность и цена.
   * Конструктор и геттеры для доступа к данным.
2. **Класс PriceMonitor**:
   * Хранит минимальные цены и количество магазинов для каждой жирности.
   * Метод addStore для добавления информации о магазине.
   * Метод getMinCount для получения итогового результата.
3. **Метод readStoresFromFile**:
   * Считывает данные о магазинах из файла и создает список объектов Store.
   * Каждая строка разбивается на части, и создается объект Store, который добавляется в список.

#### Структура данных:

* **Map<Integer, Integer> minPrice**: хранит минимальные цены для жирностей 15, 20, 25.
* **Map<Integer, Integer> minCount**: хранит количество магазинов с минимальной ценой для каждой жирности.

1. **Метод main**:

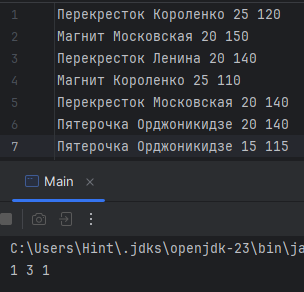
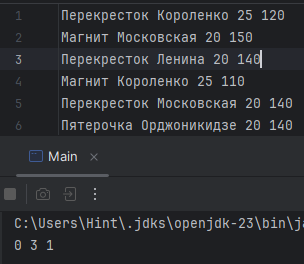
* Это точка входа в программу. При запуске программы выполняется именно этот метод.
* Сначала с помощью метода readStoresFromFile считывается информация о магазинах из файла stores.txt.
* Затем создается объект PriceMonitor, который будет следить за минимальными ценами для различных жирностей.
* Далее для каждого магазина из списка stores вызывается метод addStore объекта priceMonitor, который обновляет минимальные цены.
* В конце выводится результат в виде количества магазинов, предлагающих товар по минимальной цене для каждой жирности.

1. **Метод readStoresFromFile**:

* Этот метод считывает данные из текстового файла. В первой строке содержится количество магазинов, а в последующих строках — данные о каждом магазине: название фирмы, улица, жирность и цена.
* Все данные разбиваются на части и создается объект Store для каждого магазина. Этот объект добавляется в список stores, который в дальнейшем используется для обработки в классе PriceMonitor.

.

## Тестирование

## Код программы

Ссылка на githab

# Задание 5

## Текст задания

Файл содержит текст на русском языке. Напечатать в алфавитном порядке все звонкие согласные буквы, которые входят более чем в одно слово.

## Алгоритм решения

- Для начала создается набор звонких согласных букв, включающий такие буквы как 'б', 'в', 'г', 'д', 'ж', 'з', 'л', 'м', 'н', 'р'.

- Текст из файла загружается и разбивается на отдельные слова. Каждое слово анализируется поочередно.

- Для каждого слова извлекаются все звонкие согласные буквы. Для этого используется метод, который проверяет каждый символ в слове и, если он является звонким согласным, добавляет его в соответствующий набор.

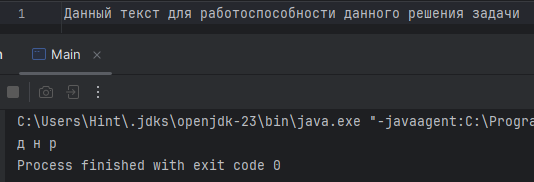
- Для каждой согласной буквы создается запись, которая хранит все слова, в которых эта буква встречается. Для этого используется Map<Character, Set<String>>, где ключом является согласная буква, а значением — множество слов, в которых эта буква присутствует.

- Из множества всех согласных выбираются только те, которые встречаются в более чем одном слове. Для этого перебираются все записи в мапе и проверяется размер множества слов для каждой согласной. Если размер множества больше 1, то такая согласная добавляется в результат.

- Результат — это множество звонких согласных, которые встречаются более чем в одном слове. Согласные выводятся в отсортированном порядке.

.

## Тестирование

.

## Код программы

Ссылка на githab

# Задание 6

## Текст задания

Переписать элементы из очереди L1 в очередь L2 в обратном порядке..

## Алгоритм решения

- **Ввод данных**:  
Программа начинает с запроса у пользователя количества элементов для очереди L1. Затем, для каждого элемента, пользователь вводит число, которое добавляется в очередь L1.

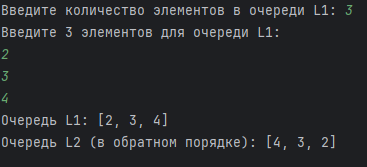
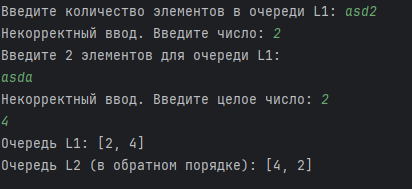
- **Использование стека для инвертирования порядка**:  
Для того чтобы инвертировать порядок элементов из очереди L1 в очередь L2, используется структура данных **стек**. Стек работает по принципу LIFO (Last In, First Out), что идеально подходит для задачи инвертирования порядка.

- **Копирование из L1 в стек**:  
Все элементы из очереди L1 последовательно удаляются и помещаются в стек.

- **Перенос элементов из стека в L2**:  
После того как все элементы оказались в стеке, они поочередно извлекаются (в обратном порядке) и помещаются в очередь L2.

- **Вывод результата**:  
В конце программа выводит содержимое очередей L1 и L2. Очередь L1 остаётся без изменений, а очередь L2 содержит элементы в обратном порядке.

## Тестирование

## Код программы

Ссылка на githab

# 

# Задание 7

## Текст задания

Необходимо написать стрим: Дан набор объектов типа Point, необходимо взять все Point в разных координатах, убрать с одинаковыми X,Y, отсортировать по X, отрицательные Y сделать положительными и собрать это все в ломаную (объект типа Polyline) Если раннее в вашем варианте не было задание с классом Point и Polyline, то написать их:

* 1. класс Point:

• Координата Х: число.

• Координата Y: число.

• Может возвращать текстовое представление вида “{X;Y}”.

* 1. класс Line (Линия), расположенная на двумерной плоскости, которая описывается:

• Координата начала: Точка

• Координата конца: Точка

• Может возвращать текстовое представление вида “Линия от {X1;Y1} до {X2;Y2}”

* 1. класс Polyline (Ломаная), которая будет представлять собой ломаную линию. Ломаная линия представляет собой набор следующих характеристик:

• Имеет массив Точек, через которые линия проходит.

• Может быть приведена к строковой форме вида “Линия [Т1,T2,…,TN]”, где TN – это результат приведения к строке Точки с номером N.

## Алгоритм решения

- **Представление данных**:  
Точки задаются объектами класса Point, которые содержат координаты x и y. Для каждой точки определен метод toString, чтобы выводить её в удобном формате.

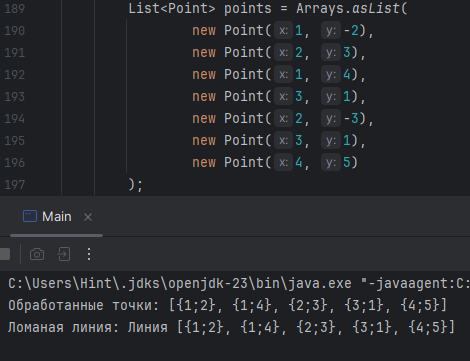
- **Обработка точек**:  
Программа выполняет несколько операций над списком точек:

* Преобразует значения координаты y в абсолютные значения.
* Удаляет дублирующиеся точки.
* Сортирует точки по координате x.

- **Создание ломаной линии**:  
После обработки точек, создается объект класса Polyline, который представляет собой ломаную линию, состоящую из обработанных точек.

- **Вывод результата**:  
Программа выводит обработанные точки и строковое представление ломаной линии.

## Тестирование



## Код программы

Ссылка на githab

# Задание 8

## Текст задания

Дан текстовый файл со строками, содержащими имя человека и его номер в следующей форме: Вася:5 Петя:3 Аня:5 Номера людей могут повторяться. У каких-то людей может не быть номера. Необходимо написать стрим выполняющую следующее: читаются все люди из файла, все имена приводится к нижнему регистру, но с первой буквой в верхнем регистре, убираем из перечня всех людей без номеров, а имена оставшихся группируются по их номеру: [5:[Вася, Аня], 3:[Петя]].

## Алгоритм решения

- **Чтение файла**:  
Программа читает файл input.txt, где каждая строка имеет формат: имя: номер. Каждое имя и номер разделены символом двоеточия.

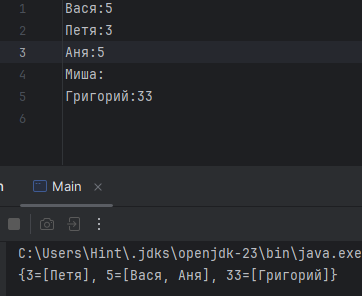
- **Парсинг строк**:  
Каждая строка разбивается на имя и номер. Имя преобразуется в формат с заглавной первой буквой и строчными последующими буквами, а номер преобразуется в целое число. Если формат строки некорректный или номер не является целым числом, строка пропускается.

- **Группировка по номеру**:  
Программа группирует объекты Person по номерам, используя структуру данных Map<Integer, List<String>>, где ключом является номер, а значением — список имен.

- **Вывод результата**:  
Программа выводит сгруппированные данные в виде: для каждого номера показываются все имена, соответствующие этому номеру.

.

## Тестирование



## Код программы

Ссылка на githab