Федеральное государственное автономное образовательное учреждение высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Лабораторная работа № 5**

Вариант 2108

|  |  |
| --- | --- |
| ФИО студента | Милёхина Елизавета Алексеевна |
| Номер группы | P3132 |
| ФИО преподавателя | Пименов Данила Дмитриевич |

г. Санкт-Петербург

2024 г.

Оглавление

[Задание. 3](#_Toc163639701)

[Объяснение работы программы. 6](#_Toc163639702)

[Ссылка на гитхаб. 7](#_Toc163639703)

[Сборка проекта с помощью Maven. 7](#_Toc163639704)

[Создание переменных окружения 7](#_Toc163639705)

[Вывод. 7](#_Toc163639706)

# Задание.

Реализовать консольное приложение, которое реализует управление коллекцией объектов в интерактивном режиме. В коллекции необходимо хранить объекты класса StudyGroup, описание которого приведено ниже.

**Разработанная программа должна удовлетворять следующим требованиям:**

* Класс, коллекцией экземпляров которого управляет программа, должен реализовывать сортировку по умолчанию.
* Все требования к полям класса (указанные в виде комментариев) должны быть выполнены.
* Для хранения необходимо использовать коллекцию типа java.util.LinkedList
* При запуске приложения коллекция должна автоматически заполняться значениями из файла.
* Имя файла должно передаваться программе с помощью: **переменная окружения**.
* Данные должны храниться в файле в формате json
* Чтение данных из файла необходимо реализовать с помощью класса java.io.BufferedInputStream
* Запись данных в файл необходимо реализовать с помощью класса java.io.OutputStreamWriter
* Все классы в программе должны быть задокументированы в формате javadoc.
* Программа должна корректно работать с неправильными данными (ошибки пользовательского ввода, отсутствие прав доступа к файлу и т.п.).

**В интерактивном режиме программа должна поддерживать выполнение следующих команд:**

* help : вывести справку по доступным командам
* info : вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.)
* show : вывести в стандартный поток вывода все элементы коллекции в строковом представлении
* add {element} : добавить новый элемент в коллекцию
* update id {element} : обновить значение элемента коллекции, id которого равен заданному
* remove\_by\_id id : удалить элемент из коллекции по его id
* clear : очистить коллекцию
* save : сохранить коллекцию в файл
* execute\_script file\_name : считать и исполнить скрипт из указанного файла. В скрипте содержатся команды в таком же виде, в котором их вводит пользователь в интерактивном режиме.
* exit : завершить программу (без сохранения в файл)
* remove\_first : удалить первый элемент из коллекции
* head : вывести первый элемент коллекции
* remove\_greater {element} : удалить из коллекции все элементы, превышающие заданный
* count\_less\_than\_semester\_enum semesterEnum : вывести количество элементов, значение поля semesterEnum которых меньше заданного
* print\_descending : вывести элементы коллекции в порядке убывания
* print\_unique\_semester\_enum : вывести уникальные значения поля semesterEnum всех элементов в коллекции

**Формат ввода команд:**

* Все аргументы команды, являющиеся стандартными типами данных (примитивные типы, классы-оболочки, String, классы для хранения дат), должны вводиться в той же строке, что и имя команды.
* Все составные типы данных (объекты классов, хранящиеся в коллекции) должны вводиться по одному полю в строку.
* При вводе составных типов данных пользователю должно показываться приглашение к вводу, содержащее имя поля (например, "Введите дату рождения:")
* Если поле является enum'ом, то вводится имя одной из его констант (при этом список констант должен быть предварительно выведен).
* При некорректном пользовательском вводе (введена строка, не являющаяся именем константы в enum'е; введена строка вместо числа; введённое число не входит в указанные границы и т.п.) должно быть показано сообщение об ошибке и предложено повторить ввод поля.
* Для ввода значений null использовать пустую строку.
* Поля с комментарием "Значение этого поля должно генерироваться автоматически" не должны вводиться пользователем вручную при добавлении.

**Описание хранимых в коллекции классов:**

public class StudyGroup {

private Integer id; //Поле не может быть null, Значение поля должно быть больше 0, Значение этого поля должно быть уникальным, Значение этого поля должно генерироваться автоматически

private String name; //Поле не может быть null, Строка не может быть пустой

private Coordinates coordinates; //Поле не может быть null

private java.time.LocalDateTime creationDate; //Поле не может быть null, Значение этого поля должно генерироваться автоматически

private Long studentsCount; //Значение поля должно быть больше 0, Поле не может быть null

private Integer shouldBeExpelled; //Значение поля должно быть больше 0, Поле может быть null

private FormOfEducation formOfEducation; //Поле может быть null

private Semester semesterEnum; //Поле может быть null

private Person groupAdmin; //Поле может быть null

}

public class Coordinates {

private float x;

private Float y; //Значение поля должно быть больше -964, Поле не может быть null

}

public class Person {

private String name; //Поле не может быть null, Строка не может быть пустой

private Float weight; //Поле может быть null, Значение поля должно быть больше 0

private String passportID; //Значение этого поля должно быть уникальным, Длина строки должна быть не меньше 5, Поле может быть null

private Colour hairColor; //Поле не может быть null

private Country nationality; //Поле может быть null

}

public enum FormOfEducation {

DISTANCE\_EDUCATION,

FULL\_TIME\_EDUCATION,

EVENING\_CLASSES;

}

public enum Semester {

FIRST,

SECOND,

THIRD,

FOURTH,

SIXTH;

}

public enum Colour {

GREEN,

RED,

YELLOW,

WHITE;

}

public enum Country {

RUSSIA,

UNITED\_KINGDOM,

INDIA,

ITALY;

}

**Отчёт по работе должен содержать:**

1. Текст задания.
2. Диаграмма классов разработанной программы.
3. Исходный код программы.
4. Выводы по работе.

**Вопросы к защите лабораторной работы:**

1. Коллекции. Сортировка элементов коллекции. Интерфейсы java.util.Comparable и java.util.Comparator.
2. Категории коллекций - списки, множества. Интерфейс java.util.Map и его реализации.
3. Параметризованные типы. Создание параметризуемых классов. Wildcard-параметры.
4. Классы-оболочки. Назначение, область применения, преимущества и недостатки. Автоупаковка и автораспаковка.
5. Потоки ввода-вывода в Java. Байтовые и символьные потоки. "Цепочки" потоков (Stream Chains).
6. Работа с файлами в Java. Класс java.io.File.
7. Пакет java.nio - назначение, основные классы и интерфейсы.
8. Утилита javadoc. Особенности автоматического документирования кода в Java.

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

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

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

Автоматически созданное описание

# Объяснение работы программы.

Для начала я создала данные мне классы и команды (все команды являются наследниками абстрактного класса Command с методом execute, которому передаются аргументы команды, которые мы пишем с командой на одной строке). Т.к. передавать каждой команде коллекцию не круто, был придуман CollectionManager, в котором есть методы для каждой команды и в котором эта самая коллекция хранится. Затем, рассмотрев повнимательнее все команды, я поняла, что нужен CommandManager, в котором (о Боже) будут храниться все команды и запускаться методом execute. Затем была замечена команда save и был создан FileManager, в котором я считываю содержимое файла и записываю в файл в формате json свою коллекцию (я была очень рада, когда всё хорошо записалось). Именно туда был добавлен флаг о нахождении в файле, потому что, к сожалению, некоторые команды различаются реализации при работе с пользователем и с файлом (пупупу). Потом в процессе создания команд я поняла, что наш пользователь может быть тупой как пробка или кот/кошка пользователя может пробежаться по клаве, поэтому были созданы исключения на случай не валидных аргументов, рекурсии в скриптах (очень хочется покрутить пальцем у виска перед таким пользователем) или введённой команды вообще не существует. И тут же встал вопрос о вводе данных для StudyGroup. Захотелось, чтобы ввод был красивым, поэтому были придуманы ФОРМЫ для каждого составного класса моей StudyGroup (и для неё, само собой разумеется). Был создан абстрактный класс Form<T> с использованием обобщений (крутая штука, мне понравилось). Так каждая форма стала наследником этого абстрактного класса. В них я прописала красивый порядок ввода (Введите, пожалуйста, то-то. Вот выбор для этого то-то) и там же я проверяю полученные данные, и, если пользователь или его кот натворили делов, говорю им об этом с просьбой переделать. Здесь был придумано ещё одно исключение для ввода чего-то не прикольного в форму. Потом до меня дошло, что если мы считываем из файлика, то нам не очень нужно, чтобы у нас выводилось «*Введите то-то*», поэтому я создала интерфейс Printable и классы Console (если работаем с пользователем) и BlankConsole (если работаем с файликом), которые его реализуют. В последнем просто все реализации выводов сделала пустыми. Потом та же участь постигла и пользовательский ввод. Был создан интерфейс UserInput и классы-реализаторы ConsoleInput и ExecuteScriptManager. Потом был добавлен класс ScannerManager, который хранит, откуда мы считываем. И вот, знаменательный момент, собираем это всё в RuntimeManager, чтобы ручками в мэйне это всё не расписывать постоянно. А в Main уже инициализируем менеджеров, добавляем командочки и проверяем на работу с файлом.

А теперь о проблемах! Как оказалось, со сканнером немного оказалось сложновато, потому что он не запоминает, где остановился сканнер в команде, поэтому при запуске файлика с командой add у меня всё кувырком: у меня пытались данные запуститься как команды, но доблестное исключение CommandDoesNotExist хранило мой сон (как оно думало), но ничего подобного! Тогда пришлось разделить в методе letsGo() RuntimeManeger’а случаи работы с пользователем и с файликом (спасибо флагу!) и там уже заниматься всяким неразрешёнными вещами. Неразрешённые вещи включали в себя создание StringBuilder’а, в который я записывала данные для команд add, update и remove\_by\_id, т.к. в формах у меня использовался другой сканнер, а при возвращении к выполнению команд здешний сканнер не видел, куда ушёл тот.

# Ссылка на гитхаб.

[kosnvny/prog\_lab5 (github.com)](https://github.com/kosnvny/prog_lab5)

# Сборка проекта с помощью Maven.

mvn compile

mvn exec:java -Dexec.mainClass="Main"

# Создание переменных окружения

export filePathToRead = \*something\*

export filePathToWrite = \*something\*

# Вывод.

В ходе лабораторной работы я узнала, как работать с коллекциями, в особенности с LinkedList, вспомнила, что такое классы-обёртки и как они связаны с примитивами, познакомилась с потоками ввода/вывода в Java, создала документацию с помощью Javadoc и (самое главное) узнала, как написать консольное приложение.