Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

**Факультет программной инженерии**

# Лабораторная работа №5
# Курса «Программирование»

Вариант 313089

Выполнил: **Нуруллаев Даниил Романович**
Группа: **P3114**

Преподаватель: Алексей Евгеньевич Письмак

Г.Санкт-Петербург

2021 г

# Задание:

Реализовать консольное приложение, которое реализует управление коллекцией объектов в интерактивном режиме. В коллекции необходимо хранить объекты класса `SpaceMarine`, 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: **аргумент командной строки**.
- Данные должны храниться в файле в формате `xml`
- Чтение данных из файла необходимо реализовать с помощью класса `java.io.BufferedReader`
- Запись данных в файл необходимо реализовать с помощью класса `java.io.FileOutputStream`
- Все классы в программе должны быть задокументированы в формате javadoc.
- Программа должна корректно работать с неправильными данными (ошибки пользовательского ввода, отсутсвие 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)
- `insert_at index {element}` : добавить новый элемент в заданную позицию
- `remove_at index` : удалить элемент, находящийся в заданной позиции коллекции (index)
- `remove_greater {element}` : удалить из коллекции все элементы, превышающие заданный
- `filter_contains_name name` : вывести элементы, значение поля name которых содержит заданную подстроку
- `filter_less_than_weapon_type weaponType` : вывести элементы, значение поля weaponType которых меньше заданного
- `print_field_descending_height` : вывести значения поля height всех элементов в порядке убывания

**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 SpaceMarine {
    private Long 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.LocalDate creationDate; //Поле не может быть null, Значение этого поля должно генерироваться автоматически
    private Float health; //Поле может быть null, Значение поля должно быть больше 0
    private double height;
    private AstartesCategory category; //Поле может быть null
    private Weapon weaponType; //Поле не может быть null
    private Chapter chapter; //Поле может быть null
}
public class Coordinates {
    private float x;
    private int y;
}
public class Chapter {
    private String name; //Поле не может быть null, Строка не может быть пустой
    private String parentLegion;
    private Integer marinesCount; //Поле может быть null, Значение поля должно быть больше 0, Максимальное значение поля: 1000
    private String world; //Поле не может быть null
}
public enum AstartesCategory {
    DREADNOUGHT,
    INCEPTOR,
    LIBRARIAN,
    CHAPLAIN,
    HELIX;
}
public enum Weapon {
    COMBI_FLAMER,
    GRENADE_LAUNCHER,
    HEAVY_FLAMER,
    MULTI_MELTA;
}
```

# Выполнение (Код):

https://github.com/susaasus1/Lab5.git

## Run:

```java
import java.io.IOException;
import java.util.Scanner;

/**
 * Основной класс приложения. Создает все экземпляры и запускает программу.
 * @author Нуруллаев Даниил p3114
 * @version 1.0
 */

public class Run {
    public static final String PS1 = "$";
    public static final String PS2 = ">";

    public static void main(String[] args) throws IOException {
        try (Scanner userScanner = new Scanner(System.in)) {
            String envVariable="";
            if (args.length==0){
                throw new ArrayIndexOutOfBoundsException();
            }
            if (args!=null || args.length!=0){
                envVariable=args[0];
            }

            MarineChecker marineAsker = new MarineChecker(userScanner);
            FileParser fileParser = new FileParser(envVariable);
            CollectionManager collectionManager = new CollectionManager(fileParser);
            CommandManager commandManager = new CommandManager(
                    new ExitCommand(),
                    new HelpCommand(),
                    new InfoCommand(collectionManager),
                    new ShowCommand(collectionManager),
                    new AddCommand(collectionManager,marineAsker),
                    new UpdateCommand(collectionManager,marineAsker),
                    new RemoveByIdCommand(collectionManager),
                    new ClearCommand(collectionManager),
                    new ExecuteScriptCommand(),
                    new SaveCommand(collectionManager),
                    new RemoveGreaterCommand(collectionManager,marineAsker),
                    new RemoveAtIndexCommand(collectionManager),
                    new InsertAtIndexCommand(collectionManager,marineAsker),
                    new FilterContainsNameCommand(collectionManager),
                    new FilterLessThanWeaponTypeCommand(collectionManager),
                    new PrintFieldDescendingHeightCommand(collectionManager)
            );
            Console console = new Console(commandManager, userScanner, marineAsker);
            console.interactiveMode();
        } catch (ArrayIndexOutOfBoundsException e){
            Console.printerror("Вы не указали аргумент!!!");
        }
    }

}
```

## SpaceMarine:

```java
import java.time.LocalDate;
import java.util.Comparator;
import java.util.Objects;

/**
 * Глвный персонаж,сохраняется в коллекцию
 */
public class SpaceMarine implements Comparator<SpaceMarine> {
```

```
    private Long id;
    private String name;
    private Coordinates coordinates;
    private java.time.LocalDate creationDate;
    private Float health;
    private double height;
    private AstartesCategory category;
    private Weapon weaponType;
    private Chapter chapter;

    public SpaceMarine(Long id, String name, Coordinates coordinates, LocalDate
creationDate, Float health, double height, AstartesCategory category, Weapon
weaponType, Chapter chapter) {
        this.id = id;
        this.name = name;
        this.coordinates = coordinates;
        this.creationDate = creationDate;
        this.health = health;
        this.height = height;
        this.category = category;
        this.weaponType = weaponType;
        this.chapter = chapter;
    }

    public SpaceMarine() {
    }

    /**
     * Задает ID солдата
     * @param id ID солдата
     */
    public void setId(Long id) {
        this.id = id;
    }

    /**
     * Задает имя солдата
     * @param name Имя солдата
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Задает координаты персонажа
     * @param coordinates Координаты солдата
     */
    public void setCoordinates(Coordinates coordinates) {
        this.coordinates = coordinates;
    }

    /**
     * Задает дату создания персонажа
     * @param creationDate Дата создания солдата
     */
    public void setCreationDate(LocalDate creationDate) {
        this.creationDate = creationDate;
    }

    /**
     * Задает здоровье солдата
     * @param health Здоровье солдата
     */
    public void setHealth(Float health) {
        this.health = health;
    }
```

```java
    /**
     * Задает рост солдата
     * @param height Рост солдата
     */
    public void setHeight(double height) {
        this.height = height;
    }

    /**
     * Задает категорию солдата
     * @param category Категория солдата
     */
    public void setCategory(AstartesCategory category) {
        this.category = category;
    }

    /**
     * Задает тип оружия солдата
     * @param weaponType Тип оружия солдата
     */
    public void setWeaponType(Weapon weaponType) {
        this.weaponType = weaponType;
    }

    /**
     * Задает часть солдата
     * @param chapter Часть солдата
     */
    public void setChapter(Chapter chapter) {
        this.chapter = chapter;
    }

    /**
     *
     * @return ID солдата
     */
    public Long getId() {
        return id;
    }

    /**
     *
     * @return Имя солдата
     */
    public String getName() {
        return name;
    }

    /**
     *
     * @return Координаты солдата
     */
    public Coordinates getCoordinates() {
        return coordinates;
    }

    /**
     *
     * @return Дату создания солдата
     */
    public LocalDate getCreationDate() {
        return creationDate;
    }

    /**
     *
     * @return Здоровье солдата
```

```
     */
    public Float getHealth() {
        return health;
    }

    /**
     *
     * @return Рост солдата
     */
    public double getHeight() {
        return height;
    }

    /**
     *
     * @return Категорию солдата
     */
    public AstartesCategory getCategory() {
        return category;
    }

    /**
     *
     * @return Тип оружия солдата
     */
    public Weapon getWeaponType() {
        return weaponType;
    }

    /**
     *
     * @return Часть солдата
     */
    public Chapter getChapter() {
        return chapter;
    }



    @Override
    public String toString() {
        return  "ID:" + id +
                "\n\tName: " + name  +
                "\n\tCoordinates: " + coordinates +
                "\n\tCreationDate(YYYY-MM-DD): " + creationDate +
                "\n\tHealth: " + health +
                "\n\tHeight: " + height +
                "\n\tCategory: " + category +
                "\n\tWeaponType: " + weaponType +
                "\n\tChapter: " + chapter
                ;
    }

    @Override
    public int compare(SpaceMarine o1, SpaceMarine o2) {
       return o1.getId().compareTo(o2.getId());
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        SpaceMarine that = (SpaceMarine) o;
        return Double.compare(that.height, height) == 0 && Objects.equals(id, that.id)
&& Objects.equals(name, that.name) && Objects.equals(coordinates, that.coordinates) &&
Objects.equals(creationDate, that.creationDate) && Objects.equals(health, that.health)
&& category == that.category && weaponType == that.weaponType &&
```

```
Objects.equals(chapter, that.chapter);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name, coordinates, creationDate, health, height,
category, weaponType, chapter);
    }


    public int compareTo(SpaceMarine marineObj) {
        return id.compareTo(marineObj.getId());
    }
}
```

## Console:

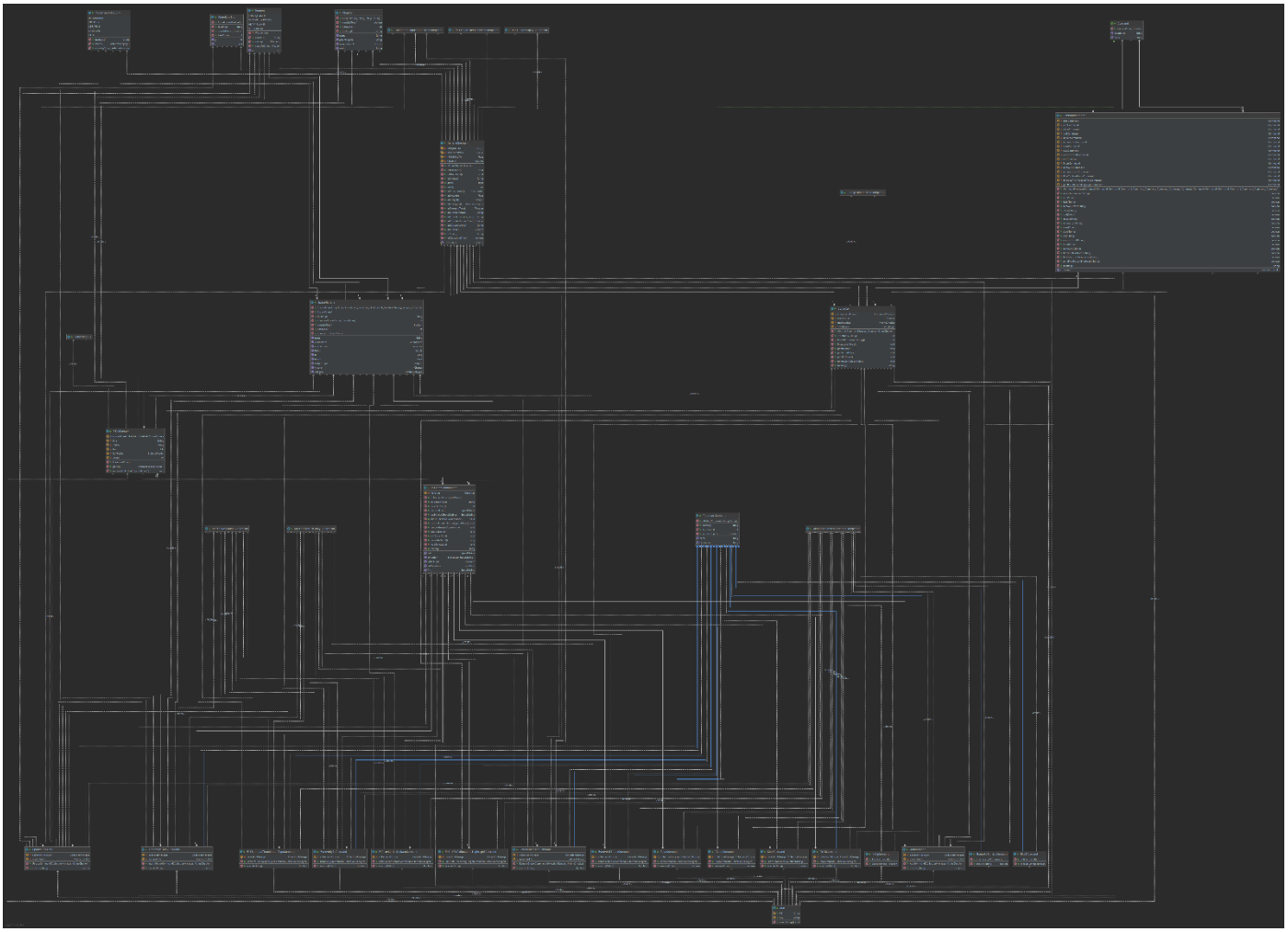
```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Scanner;


/**
 * Управляет вводом команд
 */
public class Console {

    private CommandManager commandManager;
    private Scanner userScanner;
    private MarineChecker marineAsker;
    private List<String> scriptStack=new ArrayList<>();

    public Console(CommandManager commandManager, Scanner userScanner, MarineChecker
marineAsker) {
        this.commandManager = commandManager;
        this.userScanner = userScanner;
        this.marineAsker = marineAsker;
    }

    /**
     * Мод для захвата комманд из скрипта
     * @param argument Этот аргумент
     * @return Финальный код
     */
    public int scriptMode(String argument) {
        String[] userCommand = {"", ""};
        int commandStatus;
        scriptStack.add(argument);
        try (Scanner scriptScanner = new Scanner(new File(argument))) {
            if (!scriptScanner.hasNext()) throw new NoSuchElementException();
            Scanner tmpScanner = marineAsker.getUserScanner();
            marineAsker.setUserScanner(scriptScanner);
            marineAsker.setFileMode();
            do {
                userCommand = (scriptScanner.nextLine().trim() + " ").split(" ", 2);
                userCommand[1] = userCommand[1].trim();
                while (scriptScanner.hasNextLine() && userCommand[0].isEmpty()) {
                    userCommand = (scriptScanner.nextLine().trim() + " ").split(" ",
2);
                    userCommand[1] = userCommand[1].trim();
                }
                Console.println(Run.PS1 + String.join(" ", userCommand));
```

```
                if (userCommand[0].equals("execute_script")) {
                    for (String script : scriptStack) {
                        if (userCommand[1].equals(script)) throw new
ScriptRecursionException();
                    }
                }
                commandStatus = launchCommand(userCommand);
                System.out.println(commandStatus);
            } while (commandStatus == 0 && scriptScanner.hasNextLine());
            marineAsker.setUserScanner(tmpScanner);
            marineAsker.setUserMode();
            if (commandStatus == 1 && !(userCommand[0].equals("execute_script") &&
!userCommand[1].isEmpty()))
                Console.println("Проверьте скрипт на корректность введенных данных!");
            return commandStatus;
        } catch (FileNotFoundException exception) {
            Console.printerror("Файл со скриптом не найден!");
        } catch (NoSuchElementException exception) {
            Console.printerror("Файл со скриптом пуст!");
        } catch (ScriptRecursionException exception) {
            Console.printerror("Скрипты не могут вызываться рекурсивно!");
        } catch (IllegalStateException exception) {
            Console.printerror("Непредвиденная ошибка!");
            System.exit(0);
        } finally {
            scriptStack.remove(scriptStack.size()-1);
        }
        return 1;
    }

    /**
     * Запускает команду
     * @param userCommand Команда для запуска
     * @return Финальный код
     */
    private int launchCommand(String[] userCommand) {
        switch (userCommand[0]) {
            case "":
                break;
            case "help":
                if (!commandManager.help(userCommand[1])) return 1;
                break;
            case "info":
                if (!commandManager.info(userCommand[1])) return 1;
                break;
            case "show":
                if (!commandManager.show(userCommand[1])) return 1;
                break;
            case "add":
                if (!commandManager.add(userCommand[1])) return 1;
                break;
            case "update":
                if (!commandManager.update(userCommand[1])) return 1;
                break;
            case "remove_by_id":
                if (!commandManager.removeById(userCommand[1])) return 1;
                break;
            case "clear":
                if (!commandManager.clear(userCommand[1])) return 1;
                break;
            case "save":
                if (!commandManager.save(userCommand[1])) return 1;
                break;
            case "execute_script":
                if (!commandManager.executeScript(userCommand[1])) return 1;
                else return scriptMode(userCommand[1]);
            case "insert_at":
```

```
                if (!commandManager.insert(userCommand[1])) return 1;
                break;
            case "remove_greater":
                if (!commandManager.removeGreater(userCommand[1])) return 1;
                break;
            case "remove_at":
                if (!commandManager.removeAt(userCommand[1])) return 1;
                break;
            case "filter_contains_name":
                if (!commandManager.filterContainsName(userCommand[1])) return 1;
                break;
            case "filter_less_than_weapon_type":
                if (!commandManager.filterLessThanWeaponType(userCommand[1])) return 1;
                break;
            case "print_field_descending_height":
                if (!commandManager.printFieldDescendingHeight(userCommand[1])) return
1;
                break;
            case "exit":
                if (!commandManager.exit(userCommand[1])) return 1;
                else return 2;
            default:
                if (!commandManager.noSuchCommand(userCommand[0])) return 1;
        }
        return 0;
    }

    /**
     * Режим для захвата команд из пользовательского ввода
     */
    public void interactiveMode() {
        String[] userCommand = {"", ""};
        int commandStatus;
        try {
            do {
                Console.print(Run.PS1);
                userCommand = (userScanner.nextLine().trim() + " ").split(" ", 2);
                userCommand[1] = userCommand[1].trim();
                commandStatus = launchCommand(userCommand);
            } while (commandStatus != 2);
        } catch (NoSuchElementException exception) {
            Console.printerror("Пользовательский ввод не обнаружен!");
        } catch (IllegalStateException exception) {
            Console.printerror("Непредвиденная ошибка!");
        }
    }

    /**
     * Печатает toOut.toString() в консоль
     * @param toOut Объект для печати
     */
    public static void print(Object toOut) {
        System.out.print(toOut);
    }

    /**
     * Печатает error : toOut.toString() в консоль
     * @param toOut Ошибка для печати
     */
    public static void printerror(Object toOut) {
        System.out.println("error: " + toOut);
    }
    /**
     * Печатает toOut.toString()+\n в консоль
     * @param toOut Объект для печати
     */
    public static void println(Object toOut) {
```

```
        System.out.println(toOut);
    }

    /**
     * Печатает два отформотированных элемента в консоль
     * @param element1 Первый элемент
     * @param element2 Второй элемент
     */
    public static void printtable(Object element1, Object element2) {
        System.out.printf("%-45s%-1s%n", element1, element2);
    }

    @Override
    public String toString() {
        return "Console (класс для обработки ввода команд)";
    }
}
```

**Uml:**



**Вывод:** В ходе данной лабораторной работы я реализовал консольное приложение, изучил коллекции, разные паттерны программирования еще больше познакомился с исключениями, поработал с вводом и выводом информации ,разобрался с Javadoc.