Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

**Факультет программной инженерии**

**Лабораторная работа №5**

**Курса «Программирование»**

Вариант 313089

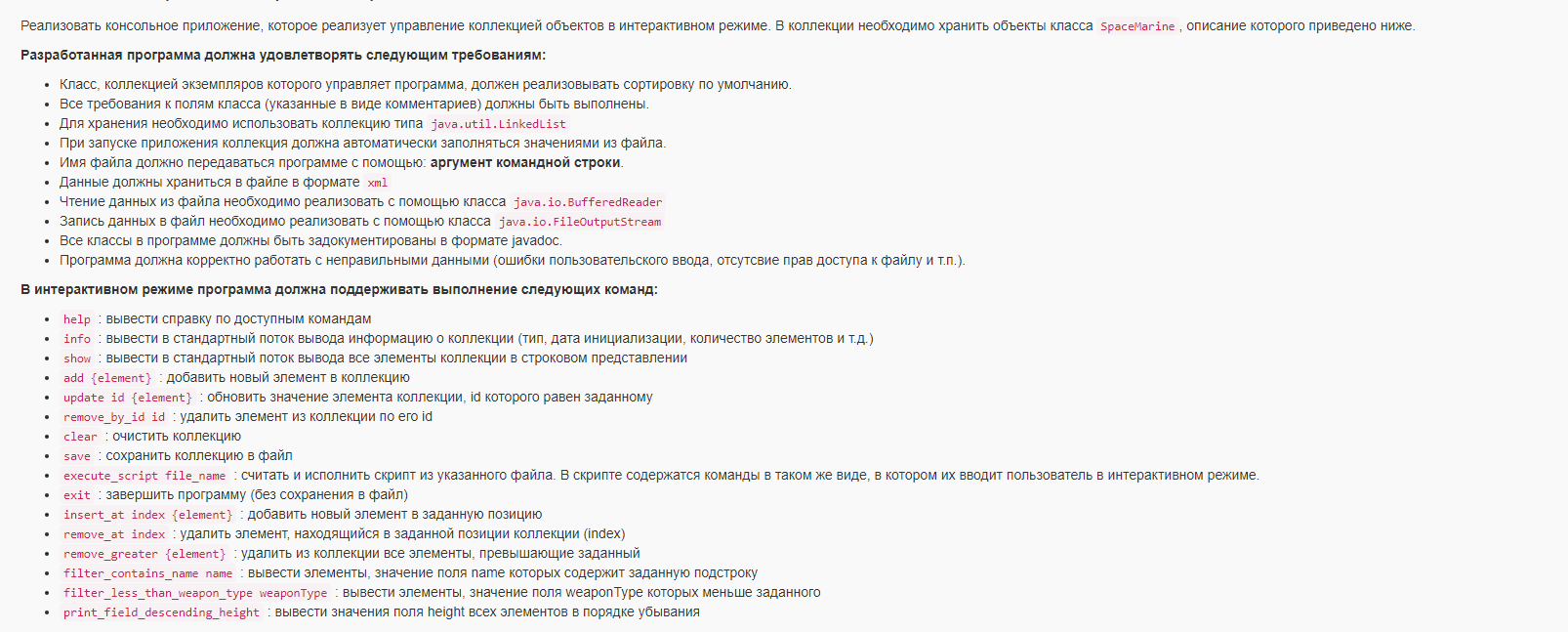
Выполнил: **Нуруллаев Даниил Романович**Группа: **Р3114**

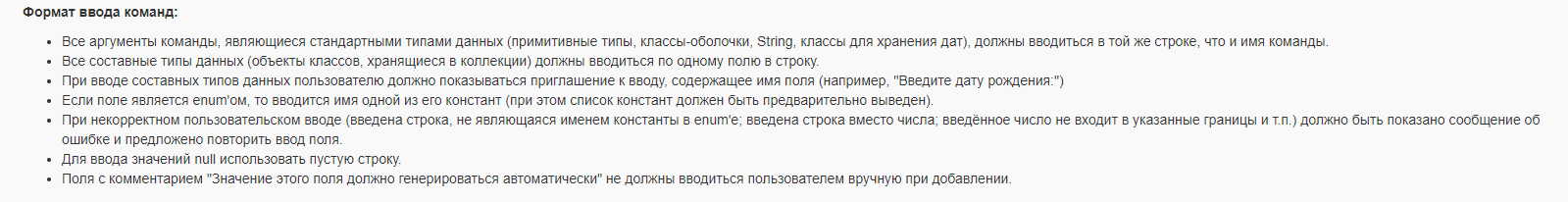
Преподаватель: Алексей Евгеньевич Письмак

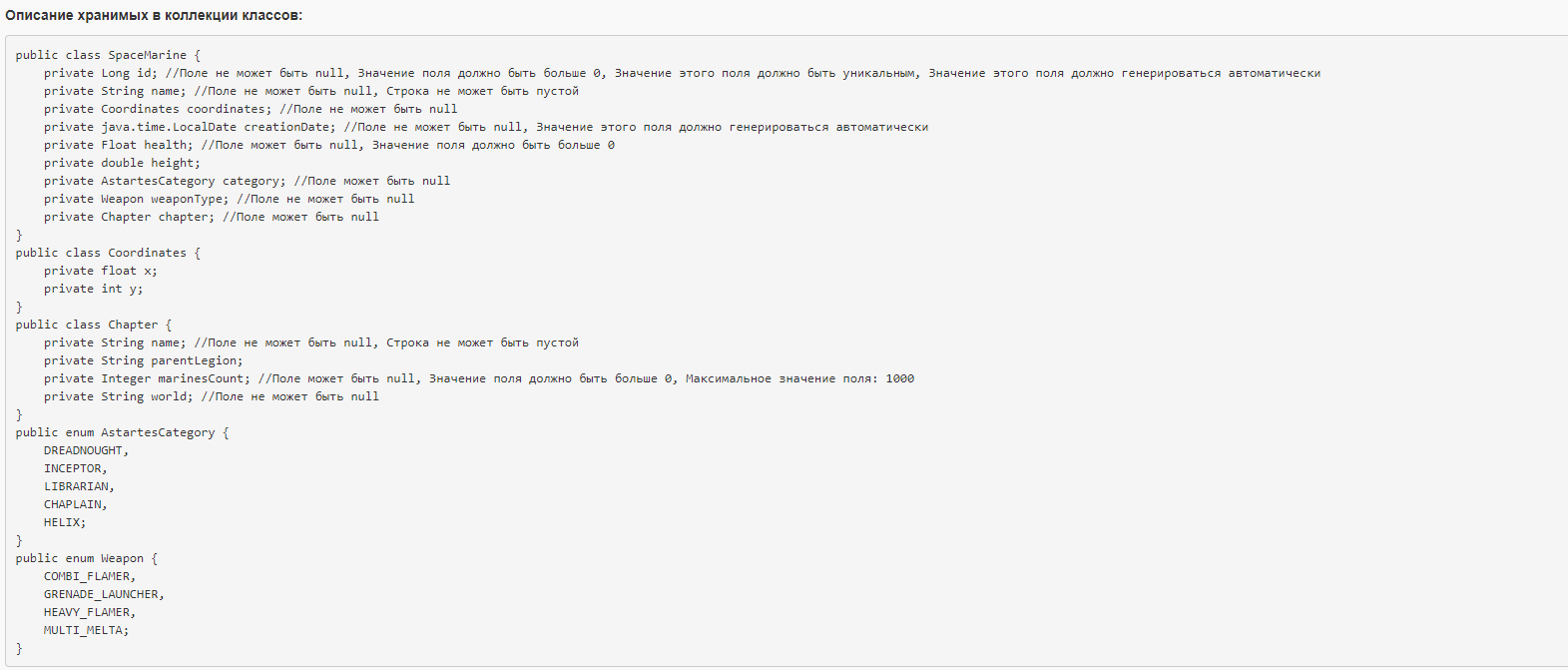
Г.Санкт-Петербург

2021 г

**Задание:**







**Выполнение (Код):**

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

**Run:**

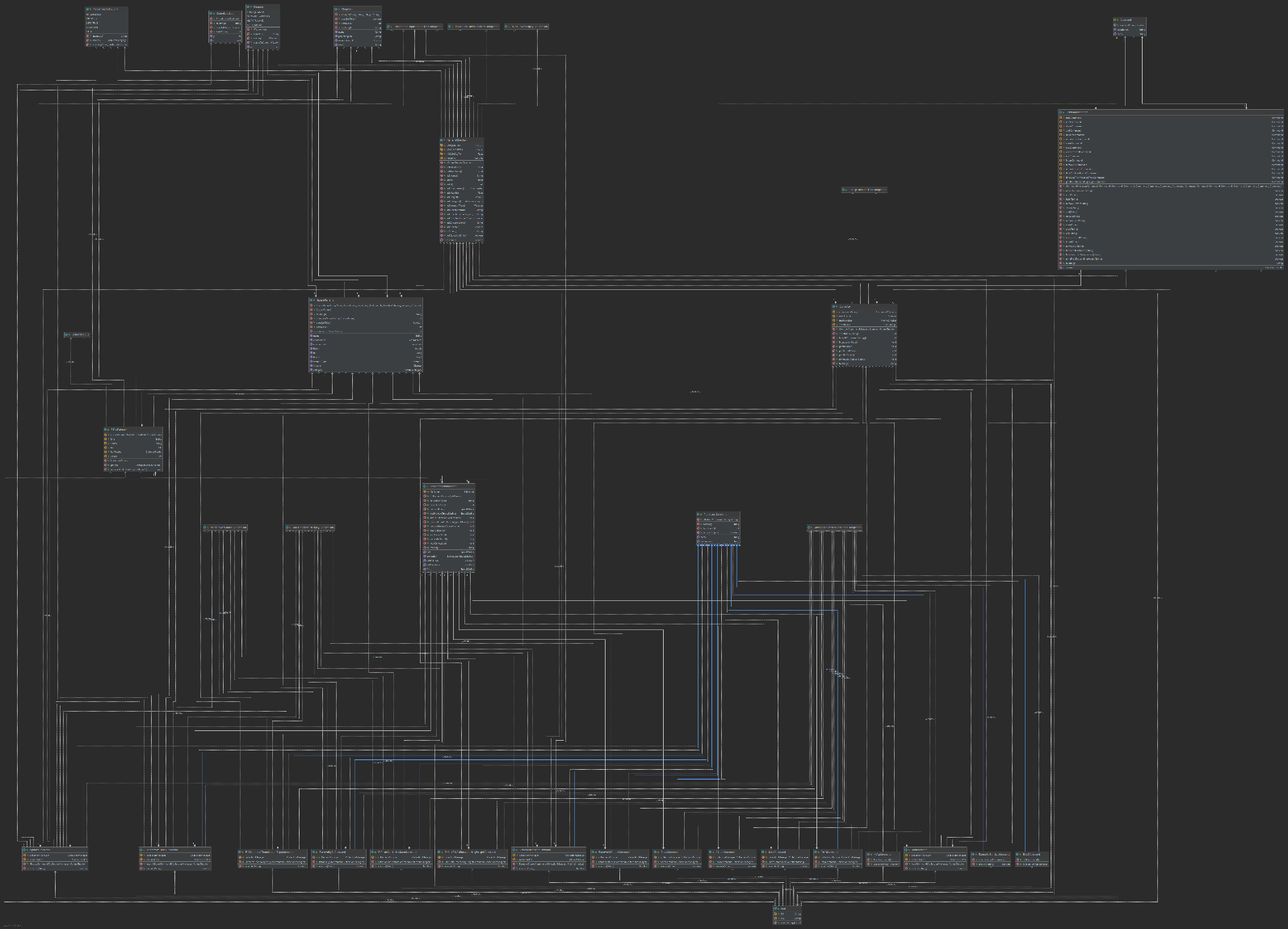
import java.io.IOException;  
import java.util.Scanner;  
  
*/\*\*  
 \* Основной класс приложения. Создает все экземпляры и запускает программу.  
 \** ***@author*** *Нуруллаев Даниил p3114  
 \** ***@version*** *1.0  
 \*/*public class Run {  
 public static final String *PS1* = "$";  
 public static 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:**

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;  
 }  
  
 */\*\*  
 \* 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.