«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет

По лабораторной работе №5

Вариант 1674

Студент:

*Ильин Н. С.*

*Р3110 поток 2.9*

Преподаватель:

*Бойко В.А.*

Санкт-Петербург, 2024 г.

**Оглавление**

[Задание: 3](#_Toc145408419)

[Выполнение работы: 3](#_Toc145408420)

[Выводы: 12](#_Toc145408421)

# **Задание:**

Разделить программу из [лабораторной работы №5](https://se.ifmo.ru/courses/programming#lab5) на клиентский и серверный модули. Серверный модуль должен осуществлять выполнение команд по управлению коллекцией. Клиентский модуль должен в интерактивном режиме считывать команды, передавать их для выполнения на сервер и выводить результаты выполнения.

**Необходимо выполнить следующие требования:**

* Операции обработки объектов коллекции должны быть реализованы с помощью Stream API с использованием лямбда-выражений.
* Объекты между клиентом и сервером должны передаваться в сериализованном виде.
* Объекты в коллекции, передаваемой клиенту, должны быть отсортированы по размеру
* Клиент должен корректно обрабатывать временную недоступность сервера.
* Обмен данными между клиентом и сервером должен осуществляться по протоколу TCP
* Для обмена данными на сервере необходимо использовать **потоки ввода-вывода**
* Для обмена данными на клиенте необходимо использовать **сетевой канал**
* Сетевые каналы должны использоваться в неблокирующем режиме.

**Обязанности серверного приложения:**

* Работа с файлом, хранящим коллекцию.
* Управление коллекцией объектов.
* Назначение автоматически генерируемых полей объектов в коллекции.
* Ожидание подключений и запросов от клиента.
* Обработка полученных запросов (команд).
* Сохранение коллекции в файл при завершении работы приложения.
* Сохранение коллекции в файл при исполнении специальной команды, доступной только серверу (клиент такую команду отправить не может).

**Серверное приложение должно состоять из следующих модулей (реализованных в виде одного или нескольких классов):**

* Модуль приёма подключений.
* Модуль чтения запроса.
* Модуль обработки полученных команд.
* Модуль отправки ответов клиенту.

Сервер должен работать в **однопоточном** режиме.

**Обязанности клиентского приложения:**

* Чтение команд из консоли.
* Валидация вводимых данных.
* Сериализация введённой команды и её аргументов.
* Отправка полученной команды и её аргументов на сервер.
* Обработка ответа от сервера (вывод результата исполнения команды в консоль).
* Команду save из клиентского приложения необходимо убрать.
* Команда exit завершает работу клиентского приложения.

**Важно!**Команды и их аргументы должны представлять из себя объекты классов. Недопустим обмен "простыми" строками. Так, для команды add или её аналога необходимо сформировать объект, содержащий тип команды и объект, который должен храниться в вашей коллекции.

**Дополнительное задание:**  
Реализовать логирование различных этапов работы сервера (начало работы, получение нового подключения, получение нового запроса, отправка ответа и т.п.) с помощью **Log4J2**

**Отчёт по работе должен содержать:**

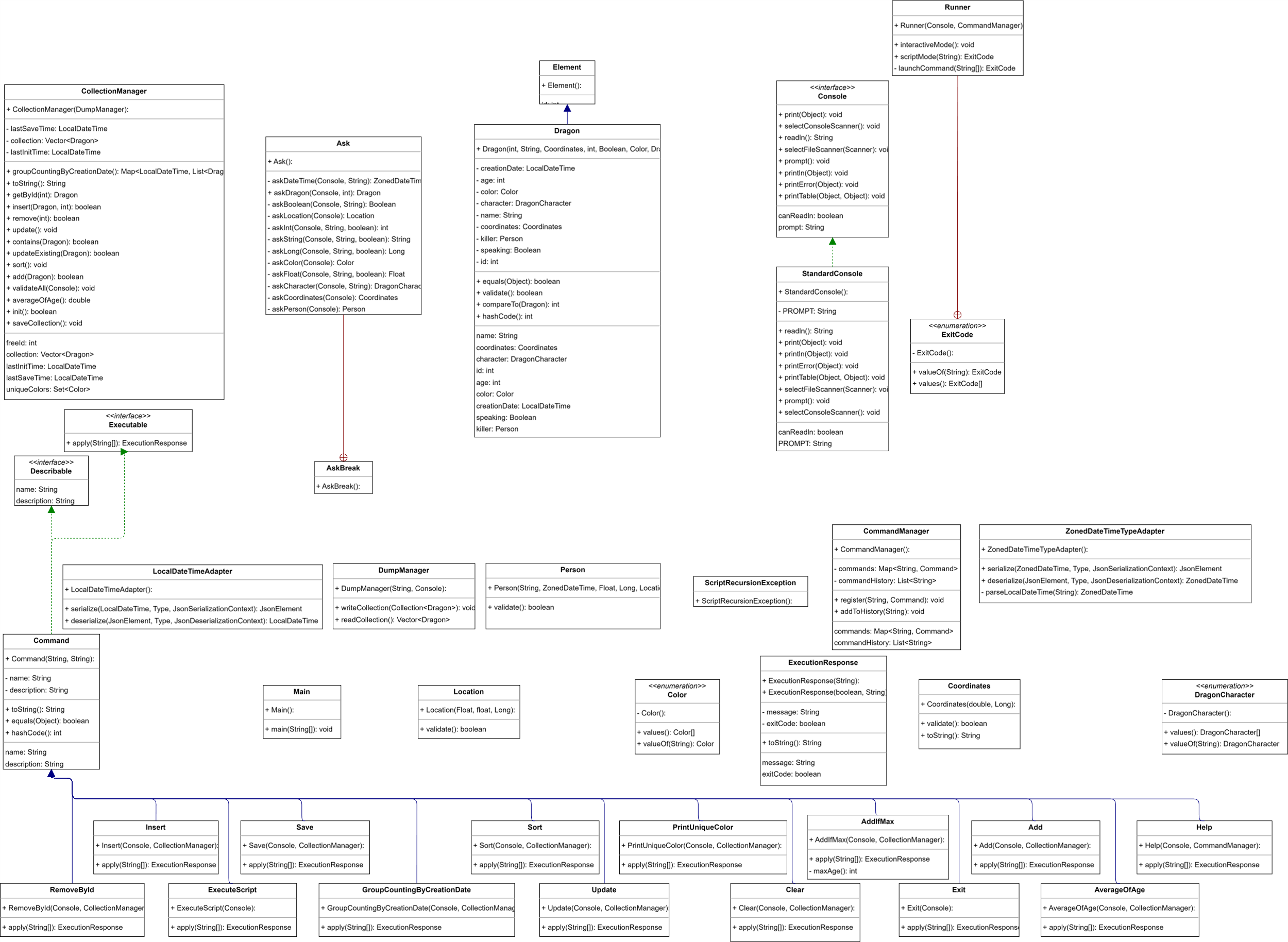
1. Текст задания.
2. Диаграмма классов разработанной программы (как клиентского, так и серверного приложения).
3. Исходный код программы.
4. Выводы по работе.

**Вопросы к защите лабораторной работы:**

1. Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия.
2. Протокол TCP. Классы Socket и ServerSocket.
3. Протокол UDP. Классы DatagramSocket и DatagramPacket.
4. Отличия блокирующего и неблокирующего ввода-вывода, их преимущества и недостатки. Работа с сетевыми каналами.
5. Классы SocketChannel и DatagramChannel.
6. Передача данных по сети. Сериализация объектов.
7. Интерфейс Serializable. Объектный граф, сериализация и десериализация полей и методов.
8. Java Stream API. Создание конвейеров. Промежуточные и терминальные операции.
9. Шаблоны проектирования: Decorator, Iterator, Factory method, Command, Flyweight, Interpreter, Singleton, Strategy, Adapter, Facade, Proxy

# **Выполнение работы:**

**UML-диаграмма:**



**Github:**  
<https://github.com/MrTheFall/study/tree/master/2sem/lab5>

**Наиболее важные части кода:**

# main/java/org/lab5/Main.java

**package** **org.lab5**;

**import** **org.lab5.commands.\***;

**import** **org.lab5.managers.CollectionManager**;

**import** **org.lab5.managers.CommandManager**;

**import** **org.lab5.managers.DumpManager**;

**import** **org.lab5.utils.Runner**;

**import** **org.lab5.utils.console.StandardConsole**;

**public** **class** **Main** {

**public** **static** **void** **main**(String[] args) {

**var** console = **new** StandardConsole();

String filename = System.getenv("FILENAME");

**if** (filename == **null** || filename.isEmpty()) {

System.out.println("Задайте имя загружаемого файла с помощью переменной среды FILENAME");

System.exit(**1**);

}

System.out.println(filename);

**var** dumpManager = **new** DumpManager(filename, console);

**var** collectionManager = **new** CollectionManager(dumpManager);

**if** (!collectionManager.init()) { System.exit(**1**); }

collectionManager.validateAll(console);

**var** commandManager = **new** CommandManager() {{

register("help", **new** Help(console, **this**));

register("add", **new** Add(console, collectionManager));

register("update", **new** Update(console, collectionManager));

register("remove\_by\_id", **new** RemoveById(console, collectionManager));

register("clear", **new** Clear(console, collectionManager));

register("save", **new** Save(console, collectionManager));

register("execute\_script", **new** ExecuteScript(console));

register("exit", **new** Exit(console));

register("add\_if\_max", **new** AddIfMax(console, collectionManager));

register("insert", **new** Insert(console, collectionManager));

register("sort", **new** Sort(console, collectionManager));

register("average\_of\_age", **new** AverageOfAge(console, collectionManager));

register("group\_counting\_by\_creation\_date", **new** GroupCountingByCreationDate(console, collectionManager));

register("print\_unique\_color", **new** PrintUniqueColor(console, collectionManager));

}};

**new** Runner(console, commandManager).interactiveMode();

}

}

# main/java/org/lab5/managers/CollectionManager.java

**package** **org.lab5.managers**;

**import** **org.lab5.models.Color**;

**import** **org.lab5.models.Dragon**;

**import** **org.lab5.utils.console.Console**;

**import** **java.time.LocalDate**;

**import** **java.time.LocalDateTime**;

**import** **java.util.\***;

**import** **java.util.concurrent.atomic.AtomicInteger**;

**import** **java.util.stream.Collectors**;

/\*\*

\* Менеджер коллекции.

\*/

**public** **class** **CollectionManager** {

**private** **int** currentId = **1**;

**private** Vector<Dragon> collection = **new** Vector<>();

**private** LocalDateTime lastInitTime;

**private** LocalDateTime lastSaveTime;

**private** **final** DumpManager dumpManager;

/\*\*

\* @param dumpManager

\*/

**public** **CollectionManager**(DumpManager dumpManager) {

**this**.lastInitTime = **null**;

**this**.lastSaveTime = **null**;

**this**.dumpManager = dumpManager;

}

/\*\*

\* @return последнее время инициализации в формате LocalDateTime

\*/

**public** LocalDateTime **getLastInitTime**() {

**return** lastInitTime;

}

/\*\*

\* @return последнее время сохранения в формате LocalDateTime

\*/

**public** LocalDateTime **getLastSaveTime**() {

**return** lastSaveTime;

}

/\*\*

\* @return коллекция драконов.

\*/

**public** Vector<Dragon> **getCollection**() {

**return** collection;

}

/\*\*

\* @return Элемент коллекции по заданному ID

\*/

**public** Dragon **getById**(**int** id) {

**return** collection.stream().filter(dragon -> dragon.getId() == id).findFirst().orElse(**null**);

}

/\*\*

\* Проверяет, содержится ли переданный экземпляр дракона в коллекции. Проверка производится по id.

\* @return Boolean, есть ли дракон в коллекции.

\*/

**public** **boolean** **contains**(Dragon dragon) {

**return** dragon == **null** || getById(dragon.getId()) != **null**;

}

/\*\*

\* Получает незанятый id.

\* @return Свободный ID.

\*/

**public** **int** **getFreeId**() {

**while** (getById(currentId) != **null**)

**if** (++currentId < **0**)

currentId = **1**;

**return** currentId;

}

/\*\*

\* Добавляет дракона в коллекцию на определенную позицию.

\* @param dragon дракон для добавления.

\* @param insertIndex позиция для добавления.

\* @return true, если дракон был успешно добавлен, иначе - false.

\*/

**public** **boolean** **insert**(Dragon dragon, **int** insertIndex) {

**if** (contains(dragon)) **return** **false**;

collection.insertElementAt(dragon, insertIndex);

//update();

**return** **true**;

}

/\*\*

\* Добавляет дракона в коллекцию.

\* @param dragon дракон для добавления.

\* @return true, если дракон был успешно добавлен, иначе - false.

\*/

**public** **boolean** **add**(Dragon dragon) {

**if** (contains(dragon)) **return** **false**;

collection.add(dragon);

update();

**return** **true**;

}

/\*\*

\* Обновляет существующего дракона в коллекции.

\* @param dragon дракон с обновленными данными.

\* @return true, если дракон был успешно обновлен, иначе - false.

\*/

**public** **boolean** **updateExisting**(Dragon dragon) {

**if** (!contains(dragon)) **return** **false**;

collection.removeIf(existingDragon -> existingDragon.getId() == dragon.getId());

collection.add(dragon);

update();

**return** **true**;

}

/\*\*

\* Удаляет дракона из коллекции по его идентификатору.

\* @param id идентификатор дракона для удаления.

\* @return true, если дракон был успешно удален, иначе - false.

\*/

**public** **boolean** **remove**(**int** id) {

Dragon dragonToRemove = getById(id);

**if** (dragonToRemove == **null**) **return** **false**;

collection.remove(dragonToRemove);

update();

**return** **true**;

}

/\*\*

\* Обновляет и сортирует коллекцию.

\*/

**public** **void** **update**() {

**this**.sort(); // Dragon реализует Comparable<Dragon>

}

/\*\*

\* Сортирует коллекцию.

\*/

**public** **void** **sort**() { Collections.sort(collection); }

/\*\*

\* Расчитывает средний возраст драконов в коллекции.

\* @return средний возраст драконов.

\*/

**public** **double** **averageOfAge**() {

**return** collection.stream()

.mapToInt(Dragon::getAge) // Допущение, что у класса Dragon есть метод getAge

.average() // Вычисляем среднее значение, результат OptionalDouble

.orElse(Double.NaN); // Возвращаем NaN, если коллекция пуста

}

/\*\*

\* Группирует элементы коллекции по дате создания.

\* @return Строковое представление группировки.

\*/

**public** Map<LocalDateTime, List<Dragon>> **groupCountingByCreationDate**() {

**return** collection.stream()

.collect(Collectors.groupingBy(Dragon::getCreationDate));

}

/\*\*

\* Возвращает уникальные цвета драконов в коллекции.

\* @return набор уникальных цветов драконов.

\*/

**public** Set<Color> **getUniqueColors**() {

**return** collection.stream()

.map(Dragon::getColor)

.collect(Collectors.toSet());

}

/\*\*

\* Инициализирует коллекцию, загружая данные из dumpManager.

\* @return true, если инициализация прошла успешно.

\*/

**public** **boolean** **init**() {

collection.clear();

collection = dumpManager.readCollection();

lastInitTime = LocalDateTime.now();

**for** (Dragon dragon : collection) {

**int** id = dragon.getId();

**if** (id > currentId) {

currentId = id;

}

}

update();

**return** **true**;

}

/\*\*

\* Проверяет валидность всех элементов коллекции. В случае обнаружения невалидного элемента выводит ошибку.

\* @param console консоль для вывода сообщений.

\*/

**public** **void** **validateAll**(Console console) {

collection.forEach(dragon -> {

**if** (!dragon.validate()) {

console.printError("Дракон с id=" + dragon.getId() + " имеет невалидные поля.");

}

});

console.println("Выполнена проверка корректности загруженных данных");

}

/\*\*

\* Сохраняет текущую коллекцию с помощью dumpManager.

\*/

**public** **void** **saveCollection**() {

dumpManager.writeCollection(collection);

lastSaveTime = LocalDateTime.now();

}

/\*\*

\* Возвращает строковое представление коллекции.

\* @return строковое представление коллекции.

\*/

**@Override**

**public** String **toString**() {

**if** (collection.isEmpty()) **return** "Коллекция пуста!";

StringBuilder info = **new** StringBuilder();

**for** (Dragon dragon : collection) {

info.append(dragon).append("\n\n");

}

**return** info.toString().trim();

}

}

# main/java/org/lab5/managers/CommandManager.java

**package** **org.lab5.managers**;

**import** **org.lab5.commands.Command**;

**import** **java.util.ArrayList**;

**import** **java.util.HashMap**;

**import** **java.util.List**;

**import** **java.util.Map**;

/\*\*

\* Менеджер команд.

\*/

**public** **class** **CommandManager** {

**private** **final** Map<String, Command> commands = **new** HashMap<>();

**private** **final** List<String> commandHistory = **new** ArrayList<>();

/\*\*

\* Добавляет команду.

\* @param commandName Название команды.

\* @param command Команда.

\*/

**public** **void** **register**(String commandName, Command command) {

commands.put(commandName, command);

}

/\*\*

\* @return Словарь команд.

\*/

**public** Map<String, Command> **getCommands**() {

**return** commands;

}

/\*\*

\* @return История команд.

\*/

**public** List<String> **getCommandHistory**() {

**return** commandHistory;

}

/\*\*

\* Добавляет команду в историю.

\* @param command Команда.

\*/

**public** **void** **addToHistory**(String command) {

commandHistory.add(command);

}

}

# main/java/org/lab5/managers/DumpManager.java

**package** **org.lab5.managers**;

**import** **com.google.gson.\***;

**import** **com.google.gson.reflect.TypeToken**;

**import** **org.lab5.models.Dragon**;

**import** **org.lab5.utils.adapters.LocalDateTimeAdapter**;

**import** **org.lab5.utils.adapters.ZonedDateTimeTypeAdapter**;

**import** **org.lab5.utils.console.Console**;

**import** **java.io.\***;

**import** **java.nio.charset.StandardCharsets**;

**import** **java.time.LocalDateTime**;

**import** **java.time.ZonedDateTime**;

**import** **java.time.format.DateTimeFormatter**;

**import** **java.util.Collection**;

**import** **java.util.NoSuchElementException**;

**import** **java.util.Scanner**;

**import** **java.util.Vector**;

/\*\*

\* Менеджер, отвечающий за работу с файлами

\*/

**public** **class** **DumpManager** {

**private** **final** Gson gson = **new** GsonBuilder()

.setPrettyPrinting()

.serializeNulls()

.registerTypeAdapter(LocalDateTime.class, **new** LocalDateTimeAdapter())

.registerTypeAdapter(ZonedDateTime.class, **new** ZonedDateTimeTypeAdapter())

.create();

**private** **final** String fileName;

**private** **final** Console console;

/\*\*

\* @param fileName

\* @param console

\*/

**public** **DumpManager**(String fileName, Console console) {

**this**.fileName = fileName;

**this**.console = console;

}

/\*\*

\* Записывает коллекцию в файл.

\* @param collection коллекция

\*/

**public** **void** **writeCollection**(Collection<Dragon> collection) {

**try** (OutputStreamWriter writer = **new** OutputStreamWriter(**new** FileOutputStream(**new** File(fileName)), StandardCharsets.UTF\_8)) {

writer.write(gson.toJson(collection));

console.println("Коллекция успешна сохранена в файл!");

} **catch** (IOException exception) {

console.printError("Загрузочный файл не может быть открыт!");

}

}

/\*\*

\* Считывает коллекцию из файл.

\* @return Считанная коллекция

\*/

**public** Vector<Dragon> **readCollection**() {

**if** (fileName != **null** && !fileName.isEmpty()) {

**try** (**var** scanner = **new** Scanner(**new** File(fileName))) {

**var** jsonString = **new** StringBuilder();

**while**(scanner.hasNextLine()) {

String line = scanner.nextLine().trim();

**if** (!line.isEmpty()) {

jsonString.append(line);

}

}

**if** (jsonString.length() == **0**) {

jsonString = **new** StringBuilder("[]");

}

Vector<Dragon> collection;

collection = gson.fromJson(jsonString.toString(), **new** TypeToken<Vector<Dragon>>() {}.getType());

System.out.println(gson.toJson(collection));

console.println("Коллекция успешно загружена!");

**return** collection;

} **catch** (FileNotFoundException exception) {

console.printError("Загрузочный файл не найден!");

} **catch** (NoSuchElementException exception) {

console.printError("Загрузочный файл пуст!");

} **catch** (JsonParseException exception) {

console.printError("В загрузочном файле не обнаружена необходимая коллекция!");

} **catch** (IllegalStateException exception) {

console.printError("Непредвиденная ошибка!");

System.exit(**0**);

}

} **else** {

console.printError("Аргумент командной строки с загрузочным файлом не найден!");

}

**return** **new** Vector<>();

}

}

# **Выводы**:

В процессе выполнения работы я столкнулся с использованием новых технологий. Так как мне было необходимо использовать java.util.Vector как коллекцию для хранения данных, мне пришлось изучить нюансы работы векторов в Java. С одной стороны, Vector синхронизирован. С другой стороны, из-за синхронизации происходит снижение производительности по сравнению с другими коллекциями, такими как ArrayList, и это может быть критичным для высоконагруженных систем, это стоит учитывать в дальнейшем.

Впервые я встретился и с необходимостью использовать интерфейс Comparable, чтобы обеспечить внутренний порядок элементов в коллекции. Реализовать логику переопределяемого метода compareTo было несложно. Теперь я умею организовывать логику сравнения собственных классов, и понимаю разницу между Comparable и Comparator.

Использование java.util.Scanner для чтения данных из файла было простым и удобным решением для разбора текстовых данных, однако этот класс не предоставляет большого контроля за кодировкой файла и может быть не лучшим выбором для обработки больших объёмов данных.

Запись данных с использованием java.io.FileOutputStream позволила мне напрямую управлять процессом записи данных в файл. Этот выбор предоставил мне полный контроль над записью, но в то же время потребовал дополнительного управления ошибками.

Тщательная документация всех классов в формате Javadoc оказалась времязатратной, но она облегчила понимание структуры и назначения всех компонентов программы, что несомненно будет полезным для тех, кто будет работать с кодом в будущем.

Обработка исключений, таких как ошибки пользовательского ввода или проблемы с доступом к файлам, обеспечивает устойчивость приложения. Это превратило множество потенциальных "тупиков" для пользователя в навигируемые предупреждения, что существенно улучшило пользовательский опыт.

В целом, решения, которые я принимал при реализации проекта, имели свои плюсы и минусы. В будущем я буду использовать изученные подходы с осторожностью, балансируя между производительностью, удобством использования и устойчивостью системы. Отказ от использования Vector в пользу более поздних коллекций, таких как ArrayList или LinkedList, может быть оправдан в зависимости от контекста задачи.