Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Дисциплина “Программирование”

Отчёт по лабораторной работе №6

Вариант №7896

Выполнил:

Ануфриев Андрей Сергеевич, Р3119

Проверил:

Шибаев Семен Сергеевич

г. Санкт-Петербург

2025 год

**Оглавление**

[Задание 3](#_Toc199114581)

[Диаграмма классов 4](#_Toc199114582)

[Код программы 6](#_Toc199114583)

[Работа программы: 6](#_Toc199114584)

[Вывод 6](#_Toc199114585)

# 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 местоположению
* 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 **Logback**

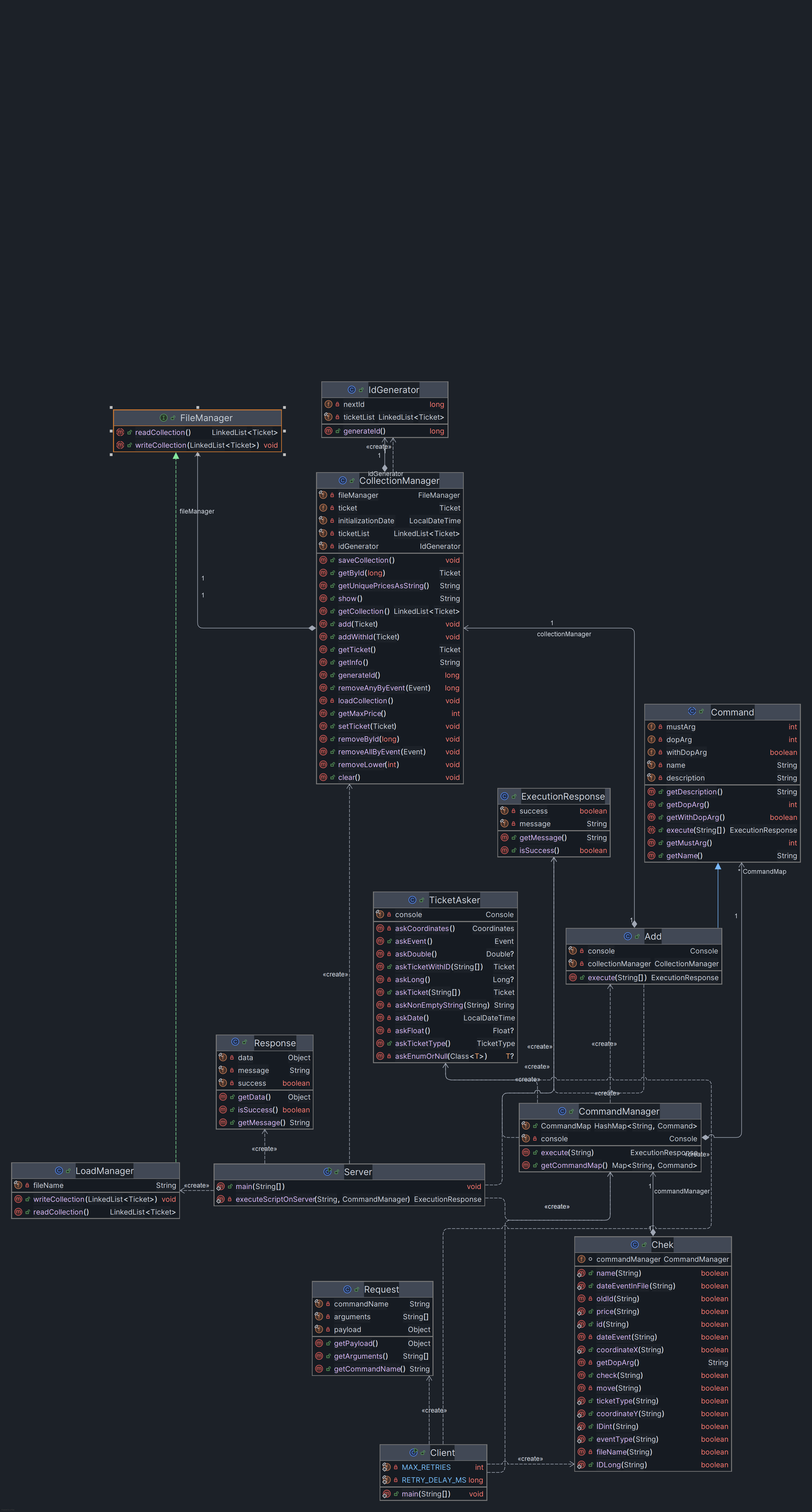
**Отчёт по работе должен содержать:**

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.

# Диаграмма классов



# Код программы

Это не гитхаб(у меня не копируется само из Intellj, а добавлять руками каждый файл не хочется. Это облако в OenDrive.

<https://w6527-my.sharepoint.com/:f:/g/personal/colnsesolotoe2022_w6527_onmicrosoft_com/EjV84SfLTT9LkL5Uotav_igBVCWqFUnKwxHXDCL8V42hhQ?e=Qno9mU>

# Работа программы:

Это видео 6 минут, с демонстрацией всех команд. (можно на х2 смотреть)

[Lab\_6 – script.txt 2025-05-26 00-44-25.mp4](https://w6527-my.sharepoint.com/:v:/g/personal/colnsesolotoe2022_w6527_onmicrosoft_com/ESX87T4t78lCmkz-UmF6SswBuhTl0m34oIiqL259yN6NQg?e=52JR5g)

# Вывод

Я научился писать клиент-серверное консольное приложение, работать с коллекциями. Применил некоторые шаблоны: такие как Command и Iterator. На практике понял, как работает общение между сервером и клиентов через TCP.

Ответы на вопросы

1. Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия.

| **Характеристика** | **TCP** | **UDP** |
| --- | --- | --- |
| Тип соединения | Ориентирован на соединение | Без установления соединения |
| Надежность | Гарантирует доставку и порядок | Нет гарантий доставки и порядка |
| Скорость | Медленнее из-за контроля | Быстрее, меньше накладных расходов |
| Использование | Веб, почта, файлы | Стриминг, игры, DNS |

1. Протокол TCP. Классы Socket и ServerSocket.

**Java классы для TCP:**

* **Socket** — представляет клиентское соединение к серверу. Позволяет отправлять и получать данные через InputStream и OutputStream.
* **ServerSocket** — слушает определенный порт и ожидает входящих соединений от клиентов. При подключении создает объект Socket для общения с клиентом.

1. Протокол UDP. Классы DatagramSocket и DatagramPacket.

**Java классы для UDP:**

* **DatagramSocket** — сокет для отправки и получения UDP-пакетов.
* **DatagramPacket** — объект, представляющий UDP-пакет с данными и адресом назначения/отправителя.

UDP-сокеты не устанавливают соединение, а просто отправляют и принимают пакеты.

1. Отличия блокирующего и неблокирующего ввода-вывода, их преимущества и недостатки. Работа с сетевыми каналами.

**Блокирующий ввод-вывод** — при вызове операций чтения или записи поток блокируется, пока операция не завершится. Прост в реализации, но может приводить к неэффективному использованию ресурсов, если поток простаивает.

**Неблокирующий ввод-вывод** — операции возвращаются сразу, даже если данные еще не готовы. Позволяет одному потоку обслуживать множество соединений, повышая производительность при большом количестве клиентов.

**Сетевые каналы** (в Java NIO) — абстракция для неблокирующего ввода-вывода, позволяют работать с буферами и селекторами для управления множеством соединений.

**Буферы (Buffers)** — это области памяти для временного хранения данных [4](http://www.pascal4eg.com/?post=62). Они используются для чтения данных из канала или записи данных в канал[4](http://www.pascal4eg.com/?post=62). В отличие от потоков в обычном IO, NIO ориентирован на буферы, что позволяет более гибко управлять данными[1](https://habr.com/ru/articles/235585/)[2](https://habr.com/ru/articles/801811/).

**Селекторы (Selectors)** — это механизм, который позволяет одному потоку отслеживать несколько каналов ввода-вывода[1](https://habr.com/ru/articles/235585/)[4](http://www.pascal4eg.com/?post=62). Селектор позволяет определить, какие каналы готовы к чтению или записи, и обрабатывать только те каналы, которые готовы к операции[6](https://coderlessons.com/tutorials/java-tekhnologii/vyuchit-java-nio/java-nio-selektor). Это позволяет эффективно использовать один поток для обслуживания множества соединений[1](https://habr.com/ru/articles/235585/).

1. Классы SocketChannel и DatagramChannel.
2. Передача данных по сети. Сериализация объектов.
3. Интерфейс Serializable. Объектный граф, сериализация и десериализация полей и методов.

* **Serializable** — маркерный интерфейс в Java, который указывает, что объект можно сериализовать.
* **Объектный граф** — все объекты, доступные из корневого объекта, сериализуются рекурсивно.
* При сериализации сохраняются значения полей, а не методы (методы не сериализуются, так как они часть класса).
* Поля с модификатором transient не сериализуются.
* При десериализации восстанавливается состояние объекта.

1. Java Stream API. Создание конвейеров. Промежуточные и терминальные операции.

**Stream API** — инструмент для функциональной обработки коллекций.

* **Конвейер (pipeline)** — цепочка операций над потоком данных.
* **Промежуточные операции** — возвращают новый Stream, позволяют фильтровать, преобразовывать данные (например, filter(), map()).
* **Терминальные операции** — завершают обработку и возвращают результат (например, collect(), forEach(), reduce()).

1. Шаблоны проектирования: Decorator, Iterator, Factory method, Command, Flyweight, Interpreter, Singleton, Strategy, Adapter, Facade, Proxy.

* **Decorator (Декоратор)** — динамическое добавление функциональности объекту без создания подклассов. Оборачивает объект, сохраняя интерфейс[1](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)).
* **Iterator (Итератор)** — предоставляет последовательный доступ к элементам коллекции без раскрытия её внутренней структуры[2](https://appmaster.io/ru/glossary/shablon-iteratora).
* **Factory Method (Фабричный метод)** — определяет интерфейс для создания объектов, позволяя подклассам решать, какой класс инстанцировать[3](https://makedev.org/patterns/creational/factoryMethod.html).
* **Command (Команда)** — превращает запрос в объект, позволяя параметризовать методы вызова, поддерживать отмену и повтор действий[4](https://javarush.com/groups/posts/4133-kofe-breyk-233-shablon-proektirovanija-command-komanda-8-samihkh-kaverznihkh-voprosov-po-java-d).
* **Flyweight (Приспособленец)** — оптимизирует использование памяти, разделяя общие данные между множеством объектов[5](https://ru.ruwiki.ru/wiki/%D0%9F%D1%80%D0%B8%D1%81%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%BB%D0%B5%D0%BD%D0%B5%D1%86_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)).
* **Interpreter** — определяет грамматику и интерпретирует предложения языка.
* **Singleton** — гарантирует существование единственного экземпляра класса.
* **Strategy** — определяет семейство алгоритмов, инкапсулирует их и делает взаимозаменяемыми.
* **Adapter** — преобразует интерфейс класса в другой интерфейс, ожидаемый клиентом.
* **Facade** — предоставляет упрощённый интерфейс к сложной системе.
* **Proxy** — предоставляет суррогат или заместителя другого объекта для контроля доступа.