Буреев Фёдор Григорьевич P3107

**Лабораторная работа №6**

**вар. 8165**

**1. 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 умолчанию
* Клиент должен корректно обрабатывать временную недоступность сервера.
* Обмен данными между клиентом и сервером должен осуществляться по протоколу UDP
* Для обмена данными на сервере необходимо использовать **сетевой канал**
* Для обмена данными на клиенте необходимо использовать **датаграммы**
* 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.

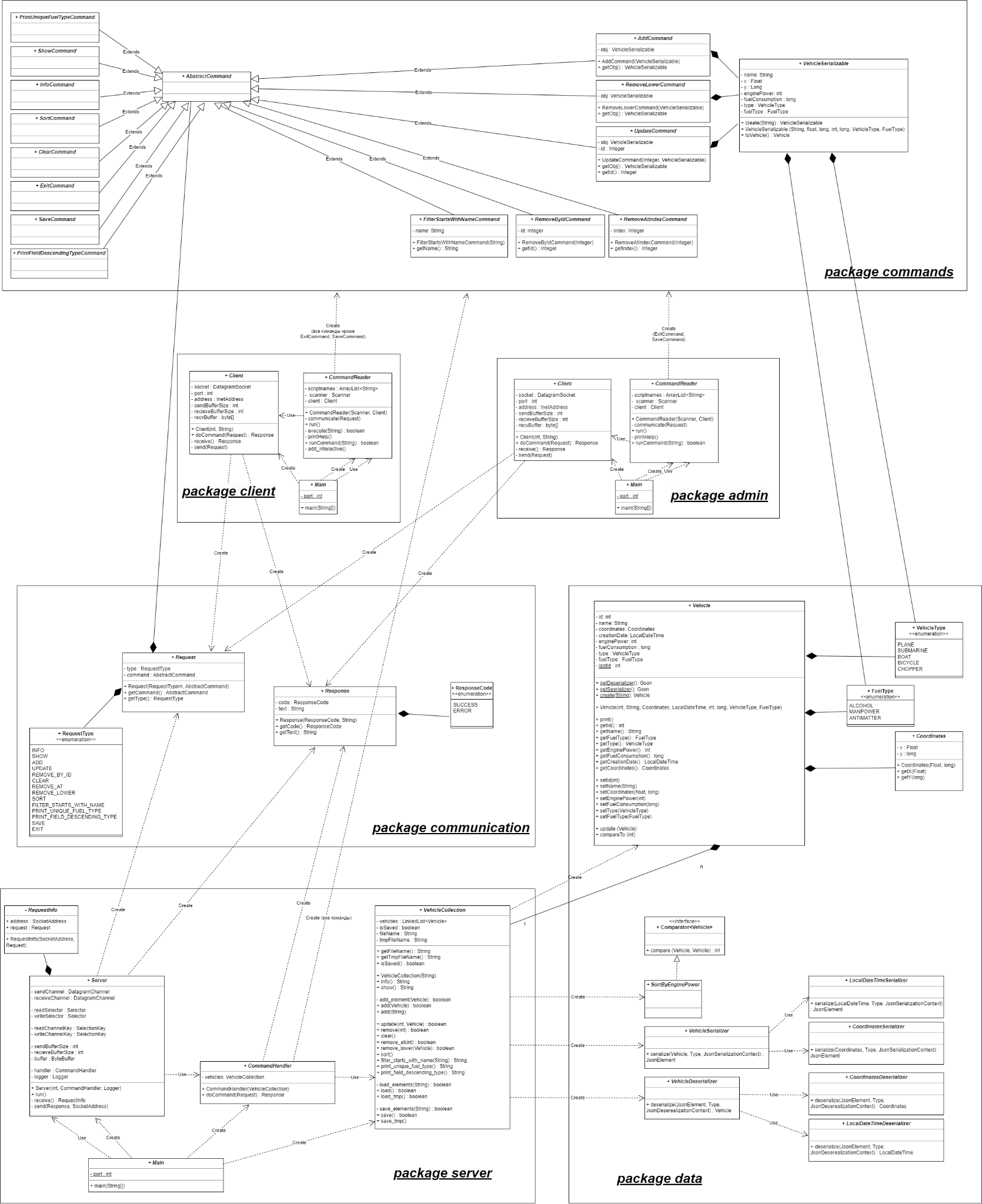
**Обязанности клиентского приложения:**

* 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.

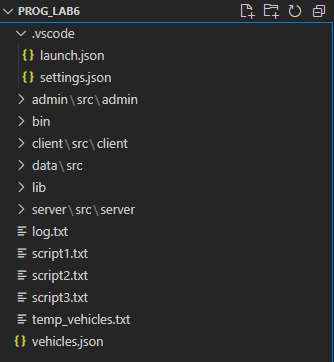
**Дополнительное задание:**Реализовать логирование различных этапов работы сервера (начало работы, получение нового подключения, получение нового запроса, отправка ответа и т.п.) с помощью **Java Util Logging**

**2. Диаграмма классов разработанной программы:**

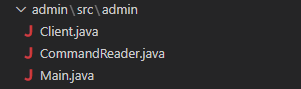
****

**3. Исходный код программы**

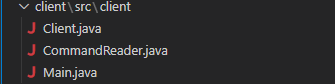
***Список пакетов:***

****

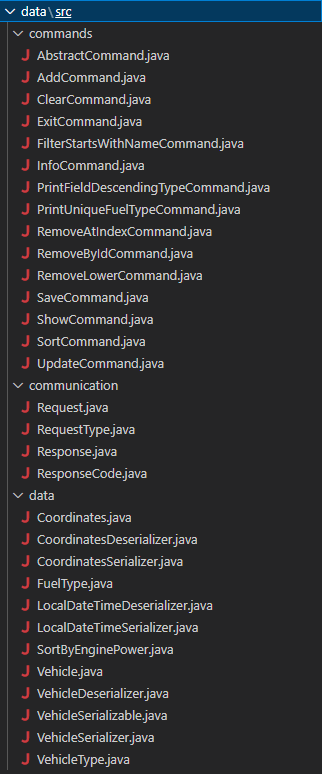
*package admin :*

**

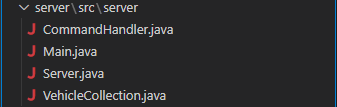
*package client:*

**

*packages commands, communication, data:*

**

*package server:*

**

***Код:***

# package admin :

## Client.java

package admin;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.net.SocketTimeoutException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import communication.Request;

import communication.Response;

public class Client {

private DatagramSocket socket;

private int port;

private InetAddress address;

private int sendBufferSize = 10000;

private int receiveBufferSize = 10000;

private byte[] recvBuffer = new byte[receiveBufferSize];

public Client(int port, String hostname) throws SocketException, UnknownHostException {

socket = new DatagramSocket();

socket.setSoTimeout(1000);

this.port = port;

address = InetAddress.getByName(hostname);

}

public Response doCommand(Request request) {

send(request);

return receive();

}

private Response receive() {

try {

DatagramPacket packet = new DatagramPacket(recvBuffer, recvBuffer.length);

socket.receive(packet);

ObjectInputStream stream = new ObjectInputStream(new ByteArrayInputStream(packet.getData(), packet.getOffset(), packet.getLength()));

Response response = (Response) stream.readObject();

stream.close();

return response;

} catch (SocketTimeoutException e) {

System.out.println("Похоже, связь с сервером прервалась.");

} catch (Exception e) {

System.out.println("Не получилось прочитать ответ из сокета : " + e);

}

return null;

}

private void send(Request request) {

try {

ByteArrayOutputStream byteStream = new ByteArrayOutputStream(sendBufferSize);

ObjectOutputStream stream = new ObjectOutputStream(byteStream);

stream.writeObject(request);

stream.flush();

byte[] sendBuffer = byteStream.toByteArray();

DatagramPacket packet = new DatagramPacket(sendBuffer, sendBuffer.length, address, port);

socket.send(packet);

stream.close();

}

catch (Exception e) {

System.out.println("Не получилось записать запрос в сокет : " + e);

}

}

}

## CommandReader.java

package admin;

import java.util.Scanner;

import commands.\*;

import communication.Response;

import communication.Request;

import communication.RequestType;

public class CommandReader {

private Scanner scanner;

private Client client;

public CommandReader(Scanner scanner, Client client) {

this.scanner = scanner;

this.client = client;

}

private void communicate(Request request) {

if(request != null) {

try {

Response response = client.doCommand(request);

if(response != null) {

System.out.println("Получили ответ (" + response.getCode() + ") : \n" + response.getText());

}

} catch (Exception e) {

System.out.println(e);

}

}

}

public void run()

{

String command = "";

while(true) {

System.out.println("Введите команду управления сервером:");

try {

command = scanner.nextLine();

if(command != null) {

command = command.trim();

}

if (runCommand(command)) {

break;

}

} catch (Exception e) {

System.out.println("Неправильная команда");

}

}

}

private void printHelp() {

System.out.println("help : вывести справку по доступным командам");

System.out.println("exit : завершить программу");

System.out.println("exitServer : завершить работу сервера");

System.out.println("save : сохранить коллекцию в файл");

}

public boolean runCommand (String command)

{

boolean needExit = false;

if(command == null) {

return needExit;

}

if(command.isEmpty()) {

return needExit;

}

String[] args;

args = command.split(" ");

if(args.length == 0) {

return needExit;

}

switch (args[0]) {

case "exit":

needExit = true;

break;

case "help":

printHelp();

break;

case "exitServer":

communicate(new Request(RequestType.EXIT, new ExitCommand()));

break;

case "save":

communicate(new Request(RequestType.SAVE, new SaveCommand()));

break;

default:

System.out.println("Неопознанная команда. Наберите 'help' для справки.");

break;

}

return needExit;

}

}

## Main.java

package admin;

import java.util.Scanner;

public class Main {

private static int port = 2000;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.println("Запускаю админ.");

Client client = new Client(port, "localhost");

CommandReader reader = new CommandReader(scanner, client);

reader.run();

} catch (Exception e) {

System.out.println("Аварийное завершение работы админа : " + e);

}

scanner.close();

}

}

# package client:

## Client.java

package client;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.net.SocketTimeoutException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import communication.Request;

import communication.Response;

public class Client {

private DatagramSocket socket;

private int port;

private InetAddress address;

private int sendBufferSize = 10000;

private int receiveBufferSize = 10000;

private byte[] recvBuffer = new byte[receiveBufferSize];

public Client(int port, String hostname) throws SocketException, UnknownHostException {

socket = new DatagramSocket();

socket.setSoTimeout(1000);

this.port = port;

address = InetAddress.getByName(hostname);

}

public Response doCommand(Request request) {

send(request);

return receive();

}

private Response receive() {

try {

DatagramPacket packet = new DatagramPacket(recvBuffer, recvBuffer.length);

socket.receive(packet);

ObjectInputStream stream = new ObjectInputStream(new ByteArrayInputStream(packet.getData(), packet.getOffset(), packet.getLength()));

Response response = (Response) stream.readObject();

stream.close();

return response;

} catch (SocketTimeoutException e) {

System.out.println("Похоже, связь с сервером прервалась.");

} catch (Exception e) {

System.out.println("Не получилось прочитать ответ из сокета : " + e);

}

return null;

}

private void send(Request request) {

try {

ByteArrayOutputStream byteStream = new ByteArrayOutputStream(sendBufferSize);

ObjectOutputStream stream = new ObjectOutputStream(byteStream);

stream.writeObject(request);

stream.flush();

byte[] sendBuffer = byteStream.toByteArray();

DatagramPacket packet = new DatagramPacket(sendBuffer, sendBuffer.length, address, port);

socket.send(packet);

stream.close();

}

catch (Exception e) {

System.out.println("Не получилось записать запрос в сокет : " + e);

}

}

}

## CommandReader.java

package client;

import java.util.Scanner;

import data.FuelType;

import data.VehicleSerializable;

import data.VehicleType;

import java.util.Arrays;

import java.util.ArrayList;

import java.io.File;

import java.io.FileInputStream;

import java.io.InputStream;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import commands.\*;

import communication.Response;

import communication.Request;

import communication.RequestType;

/\*\*

\* Класс CommandReader

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class CommandReader {

/\*\* Поле список запущенных скриптов \*/

private ArrayList<String> scriptnames;

/\*\* Поле сканнера \*/

private Scanner scanner;

private Client client;

public CommandReader(Scanner scanner, Client client) {

scriptnames = new ArrayList<String>();

this.scanner = scanner;

this.client = client;

}

private void communicate(Request request) {

if(request != null) {

try {

Response response = client.doCommand(request);

if(response != null) {

System.out.println("Получили ответ (" + response.getCode() + ") : \n" + response.getText());

}

} catch (Exception e) {

System.out.println(e);

}

}

}

public void run()

{

String command = "";

while(true) {

System.out.println("Введите команду управления коллекцией:");

try {

command = scanner.nextLine();

if(command != null) {

command = command.trim();

}

if (runCommand(command)) {

break;

}

} catch (Exception e) {

System.out.println("Неправильная команда");

}

}

}

/\*\* Читает и исполняет команды управления коллекцией из указанного файла

\* @param fileName имя файла

\* @return true, если была выполнена команда exit, false, если нет

\*/

private boolean execute (String fileName) {

boolean needExit = false;

if(scriptnames.contains(fileName)) {

System.out.println("Рекурсивный вызов скриптов запрещён");

return needExit;

}

File f = new File(fileName);

if (f.exists() && !f.isDirectory()) {

try {

scriptnames.add(fileName);

System.out.println("scriptnames : " + scriptnames.toString());

InputStream fin = new FileInputStream(fileName);

BufferedReader reader = new BufferedReader(new InputStreamReader(fin));

String line;

while ((line = reader.readLine()) != null) {

line = line.trim();

if(!line.isEmpty()) {

System.out.println("########## ВЫПОЛНЯЮ КОМАНДУ : " + line);

needExit = runCommand(line);

if(needExit) {

break;

}

}

}

reader.close();

} catch (Exception e) {

System.out.println(e);

}

scriptnames.remove(fileName);

System.out.println("scriptnames : " + scriptnames.toString());

} else {

System.out.println("Неправильное имя файла: " + fileName);

}

return needExit;

}

/\*\* Выводит список команд управления коллекцией \*/

private void printHelp() {

System.out.println("help : вывести справку по доступным командам");

System.out.println("info : вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.)");

System.out.println("show : вывести в стандартный поток вывода все элементы коллекции в строковом представлении");

System.out.println("add {element} : добавить новый элемент в коллекцию");

System.out.println("update id {element} : обновить значение элемента коллекции, id которого равен заданному");

System.out.println("remove\_by\_id id : удалить элемент из коллекции по его id");

System.out.println("clear : очистить коллекцию");

System.out.println("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.");

System.out.println("exit : завершить программу");

System.out.println("remove\_at index : удалить элемент, находящийся в заданной позиции коллекции (index)");

System.out.println("remove\_lower {element} : удалить из коллекции все элементы, меньшие, чем заданный");

System.out.println("sort : отсортировать коллекцию в естественном порядке");

System.out.println("filter\_starts\_with\_name name : вывести элементы, значение поля name которых начинается с заданной подстроки");

System.out.println("print\_unique\_fuel\_type : вывести уникальные значения поля fuelType всех элементов в коллекции");

System.out.println("print\_field\_descending\_type : вывести значения поля type всех элементов в порядке убывания");

}

public boolean runCommand (String command)

{

boolean needExit = false;

if(command == null) {

return needExit;

}

if(command.isEmpty()) {

return needExit;

}

String[] args;

args = command.split(" ");

if(args.length == 0) {

return needExit;

}

switch (args[0]) {

case "exit":

needExit = true;

break;

case "help":

printHelp();

break;

case "info":

communicate(new Request(RequestType.INFO, new InfoCommand()));

break;

case "show":

communicate(new Request(RequestType.SHOW, new ShowCommand()));

break;

case "add":

if(args.length == 1) {

add\_interactive();

} else if(args.length == 2) {

try {

VehicleSerializable obj = VehicleSerializable.create(args[1]);

if(obj != null) {

communicate(new Request(RequestType.ADD, new AddCommand(obj)));

}

} catch (Exception e) {

System.out.println("Неправильный формат задания объекта : " + args[1]);

}

} else {

System.out.println("Неверное число аргументов");

}

break;

case "update":

if(args.length == 3) {

try {

int id = Integer.parseInt(args[1]);

VehicleSerializable obj = VehicleSerializable.create(args[2]);

if(obj != null) {

communicate(new Request(RequestType.UPDATE, new UpdateCommand(id, obj)));

}

} catch (Exception e) {

System.out.println("Неправильно введены аргументы : " + e);

}

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "clear":

communicate(new Request(RequestType.CLEAR, new ClearCommand()));

break;

case "execute\_script":

if(args.length == 2) {

needExit = execute(args[1]);

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "remove\_by\_id":

if(args.length == 2) {

try {

int id = Integer.parseInt(args[1]);

communicate(new Request(RequestType.REMOVE\_BY\_ID, new RemoveByIdCommand(id)));

} catch (Exception e) {

System.out.println("Неправильно введены аргументы : " + e);

}

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "remove\_at":

if(args.length == 2) {

try {

int idx = Integer.parseInt(args[1]);

communicate(new Request(RequestType.REMOVE\_AT, new RemoveAtIndexCommand(idx)));

} catch (Exception e) {

System.out.println("Неправильно введены аргументы : " + e);

}

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "remove\_lower":

if(args.length == 2) {

VehicleSerializable obj = VehicleSerializable.create(args[1]);

if(obj != null) {

communicate(new Request(RequestType.REMOVE\_LOWER, new RemoveLowerCommand(obj)));

}

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "sort":

communicate(new Request(RequestType.SORT, new SortCommand()));

break;

case "filter\_starts\_with\_name":

if(args.length == 2) {

String str = args[1];

communicate(new Request(RequestType.FILTER\_STARTS\_WITH\_NAME, new FilterStartsWithNameCommand(str)));

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "print\_unique\_fuel\_type":

communicate(new Request(RequestType.PRINT\_UNIQUE\_FUEL\_TYPE, new PrintUniqueFuelTypeCommand()));

break;

case "print\_field\_descending\_type":

communicate(new Request(RequestType.PRINT\_FIELD\_DESCENDING\_TYPE, new PrintFieldDescendingTypeCommand()));

break;

default:

System.out.println("Неопознанная команда. Наберите 'help' для справки.");

break;

}

return needExit;

}

/\*\* Исполняет команду добавления элемента в коллекцию в интерактивном режиме ввода параметром \*/

private void add\_interactive() {

String name;

Float x;

Long y;

int enginePower;

long fuelConsumption;

VehicleType type = null;

FuelType fuelType = null;

while(true) {

System.out.println("Введите имя транспортного средства:");

try {

name = scanner.nextLine().trim();

} catch (Exception e) {

name = "";

}

if(name.isEmpty()) {

System.out.println("Ошибка: Имя не может быть пустым");

} else {

break;

}

}

while(true) {

System.out.println("Введите координату X объекта:");

try {

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

x = Float.parseFloat(str);

break;

} catch (Exception e) {

System.out.println("Ошибка: Координата X должна быть плавающим числом");

}

}

while(true) {

System.out.println("Введите координату Y объекта:");

try {

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

y = Long.parseLong(str);

break;

} catch (Exception e) {

System.out.println("Ошибка: Координата Y должна быть длинным целым числом");

}

}

while(true) {

System.out.println("Введите мощь двигателя:");

try {

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

enginePower = Integer.parseInt(str);

if(enginePower <= 0) {

throw new Exception();

}

break;

} catch (Exception e) {

System.out.println("Ошибка: Мощь двигателя должна быть целым числом больше нуля");

}

}

while(true) {

System.out.println("Введите уровень потребления топлива:");

try {

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

fuelConsumption = Long.parseLong(str);

if(fuelConsumption <= 0) {

throw new Exception();

}

break;

} catch (Exception e) {

System.out.println("Ошибка: Уровень потребления топлива должен быть числом больше нуля");

}

}

while(true) {

System.out.println("Введите тип топлива ( " + Arrays.toString(FuelType.values()) + " ): ");

try {

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

if(!str.isEmpty()) {

fuelType = FuelType.valueOf(str);

}

break;

} catch (Exception e) {

System.out.println("Ошибка: Неверный тип топлива");

}

}

while(true) {

System.out.println("Введите тип транспортного средства ( " + Arrays.toString(VehicleType.values()) + " ): ");

try {

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

if(!str.isEmpty()) {

type = VehicleType.valueOf(str);

}

break;

} catch (Exception e) {

System.out.println("Ошибка: Неверный тип транспортного средства");

}

}

communicate(new Request(RequestType.ADD, new AddCommand(new VehicleSerializable(name, x, y, enginePower, fuelConsumption, type, fuelType))));

}

}

## Main.java

package client;

import java.util.Scanner;

public class Main {

private static int port = 2000;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.println("Запускаю клиент.");

Client client = new Client(port, "localhost");

CommandReader reader = new CommandReader(scanner, client);

reader.run();

} catch (Exception e) {

System.out.println("Аварийное завершение работы клиента : " + e);

}

scanner.close();

}

}

# package commands:

## AbstractCommand.java

package commands;

import java.io.Serializable;

public class AbstractCommand implements Serializable{

}

## AddCommand.java

package commands;

import data.VehicleSerializable;

public class AddCommand extends AbstractCommand{

private VehicleSerializable obj;

public AddCommand(VehicleSerializable obj) {

this.obj = obj;

}

public VehicleSerializable getObj() {

return obj;

}

}

## ClearCommand.java

package commands;

public class ClearCommand extends AbstractCommand{

}

## ExitCommand.java

package commands;

public class ExitCommand extends AbstractCommand {

}

## FilterStartsWithNameCommand.java

package commands;

public class FilterStartsWithNameCommand extends AbstractCommand{

private String name;

public FilterStartsWithNameCommand(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

## InfoCommand.java

package commands;

public class InfoCommand extends AbstractCommand{

}

## PrintFieldDescendingTypeCommand.java

package commands;

public class PrintFieldDescendingTypeCommand extends AbstractCommand{

}

## PrintUniqueFuelTypeCommand.java

package commands;

public class PrintUniqueFuelTypeCommand extends AbstractCommand{

}

## RemoveAtIndexCommand.java

package commands;

public class RemoveAtIndexCommand extends AbstractCommand{

private Integer index;

public RemoveAtIndexCommand(Integer index) {

this.index = index;

}

public Integer getIndex() {

return index;

}

}

## RemoveByIdCommand.java

package commands;

public class RemoveByIdCommand extends AbstractCommand{

private Integer id;

public RemoveByIdCommand(Integer id) {

this.id = id;

}

public Integer getId() {

return id;

}

}

## RemoveLowerCommand.java

package commands;

import data.VehicleSerializable;

public class RemoveLowerCommand extends AbstractCommand{

private VehicleSerializable obj;

public RemoveLowerCommand(VehicleSerializable obj) {

this.obj = obj;

}

public VehicleSerializable getObj() {

return obj;

}

}

## SaveCommand.java

package commands;

public class SaveCommand extends AbstractCommand {

}

## ShowCommand.java

package commands;

public class ShowCommand extends AbstractCommand{

}

## SortCommand.java

package commands;

public class SortCommand extends AbstractCommand{

}

## UpdateCommand.java

package commands;

import data.VehicleSerializable;

public class UpdateCommand extends AbstractCommand{

private Integer id;

private VehicleSerializable obj;

public UpdateCommand(Integer id, VehicleSerializable obj) {

this.obj = obj;

this.id = id;

}

public VehicleSerializable getObj() {

return obj;

}

public Integer getId() {

return id;

}

}

# package communication:

## Request.java

package communication;

import java.io.Serializable;

import commands.AbstractCommand;

public class Request implements Serializable{

private RequestType type;

private AbstractCommand command;

public Request(RequestType type, AbstractCommand command) {

this.type = type;

this.command = command;

}

public AbstractCommand getCommand() {

return command;

}

public RequestType getType() {

return type;

}

}

## RequestType.java

package communication;

public enum RequestType{

INFO,

SHOW,

ADD,

UPDATE,

REMOVE\_BY\_ID,

CLEAR,

REMOVE\_AT,

REMOVE\_LOWER,

SORT,

FILTER\_STARTS\_WITH\_NAME,

PRINT\_UNIQUE\_FUEL\_TYPE,

PRINT\_FIELD\_DESCENDING\_TYPE,

SAVE,

EXIT

}

## Response.java

package communication;

import java.io.Serializable;

public class Response implements Serializable{

private ResponseCode code;

private String text;

public Response(ResponseCode code, String text) {

this.code = code;

this.text = text;

}

public ResponseCode getCode() {

return code;

}

public String getText() {

return text;

}

}

## ResponseCode.java

package communication;

public enum ResponseCode{

SUCCESS,

ERROR

}

# package data:

## Coordinates.java

package data;

import java.io.Serializable;

public class Coordinates implements Serializable {

private Float x;

private long y;

public Coordinates(Float x, long y)

{

this.x = x;

this.y = y;

}

public Float getX() {

return this.x;

}

public long getY() {

return this.y;

}

}

## CoordinatesDeserializer.java

package data;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс CoordinatesDeserializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class CoordinatesDeserializer implements JsonDeserializer<Coordinates> {

/\*\* Десериализует объект класса Coordinates

\* @param json Json-элемент, который десериализуют

\* @param typeOfT Тип объекта, в который прооисходит десериализация

\* @param context Контекст десериализации

\* @return Объект класса Coordinates

\*/

@Override

public Coordinates deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) throws JsonParseException {

JsonObject jsonObject = json.getAsJsonObject();

Float x = jsonObject.get("x").getAsFloat();

Long y = jsonObject.get("y").getAsLong();

return new Coordinates(x, y);

}

}

## CoordinatesSerializer.java

package data;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс CoordinatesSerializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class CoordinatesSerializer implements JsonSerializer<Coordinates> {

/\*\* Сериализует объект класса Coordinates

\* @param obj объект класса Coordinates, который сериализуют

\* @param type Тип объекта, для которого происходит сериализация

\* @param context Контекст сериализации

\* @return JsonElement сериализованного объекта

\*/

@Override

public JsonElement serialize(Coordinates obj, Type type, JsonSerializationContext context) {

JsonObject result = new JsonObject();

result.addProperty("x", obj.getX());

result.addProperty("y", obj.getY());

return result;

}

}

## FuelType.java

package data;

/\*\*

\* Класс FuelType

\* @author Буреев Фёдор

\* @version 1.0

\*/

public enum FuelType {

/\*\* Спирт \*/

ALCOHOL,

/\*\* Рабочая человеческая сила \*/

MANPOWER,

/\*\* Антиматерия \*/

ANTIMATTER;

}

## LocalDateTimeDeserializer.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс LocalDateTimeDeserializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class LocalDateTimeDeserializer implements JsonDeserializer<LocalDateTime> {

/\*\* Десериализует объект класса LocalDateTime

\* @param json Json-элемент, который десериализуют

\* @param typeOfT Тип объекта, в который прооисходит десериализация

\* @param context Контекст десериализации

\* @return Объект класса LocalDateTime

\*/

@Override

public LocalDateTime deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) throws JsonParseException {

return LocalDateTime.parse(json.getAsString());

}

}

## LocalDateTimeSerializer.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс CoordinatesSerializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class LocalDateTimeSerializer implements JsonSerializer<LocalDateTime> {

/\*\* Сериализует объект класса LocalDateTime

\* @param obj объект класса LocalDateTime, который сериализуют

\* @param type Тип объекта, для которого происходит сериализация

\* @param context Контекст сериализации

\* @return JsonElement сериализованного объекта

\*/

@Override

public JsonElement serialize(LocalDateTime obj, Type type, JsonSerializationContext context) {

return new JsonPrimitive(obj.toString());

}

}

## SortByEnginePower.java

package data;

import java.util.\*;

/\*\*

\* Класс SortByEnginePower

\* @version 1.0

\*/

public class SortByEnginePower implements Comparator<Vehicle> {

/\*\*

\* Возвращает результат сравнения мощности двигателей двух объектов класса Vehicle

\* @param a первый объект

\* @param b второй объект

\* @return результат сравнения

\*/

public int compare(Vehicle a, Vehicle b){

return a.getEnginePower() - b.getEnginePower();

}

}

## Vehicle.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

/\*\*

\* Класс Vehicle

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class Vehicle implements Comparable<Vehicle> {

/\*\* Поле идентификатор объекта\*/

private int id;

/\*\* Поле имя транспортного средства \*/

private String name;

/\*\* Поле координаты объекта \*/

private Coordinates coordinates;

/\*\* Поле время создания объекта \*/

private LocalDateTime creationDate;

/\*\* Поле мощность двигателя объекта\*/

private int enginePower;

/\*\* Поле потребление топлива объекта \*/

private long fuelConsumption;

/\*\* Поле тип транспортного средства \*/

private VehicleType type;

/\*\* Поле тип топлива \*/

private FuelType fuelType;

/\*\* Поле для вычисления свободного id \*/

private static int lastId = 0;

/\*\*

\* Возвращает объект для десериализации класса Vehicle

\* @return объект класса Gson

\*/

public static Gson getDeserializer() {

return new GsonBuilder().

registerTypeAdapter(Vehicle.class, new VehicleDeserializer()).

registerTypeAdapter(Coordinates.class, new CoordinatesDeserializer()).

registerTypeAdapter(LocalDateTime.class, new LocalDateTimeDeserializer()).

create();

}

/\*\*

\* Возвращает объект для сериализации класса Vehicle

\* @return объект класса Gson

\*/

public static Gson getSerializer() {

return new GsonBuilder().

registerTypeAdapter(Vehicle.class, new VehicleSerializer()).

registerTypeAdapter(Coordinates.class, new CoordinatesSerializer()).

registerTypeAdapter(LocalDateTime.class, new LocalDateTimeSerializer()).

create();

}

/\*\*

\* Создаёт объект типа Vehicle по строке

\* @param str Json-строка с параметрами объекта

\* @return объект класса Vehicle

\*/

public static Vehicle create(String str) {

try {

Gson gson = Vehicle.getDeserializer();

Vehicle obj = gson.fromJson(str, Vehicle.class);

return obj;

} catch (Exception e) {

System.out.println("Ошибка чтения объекта : " + e);

return null;

}

}

/\*\*

\* Создаёт объект типа Vehicle по указанным параметрам

\* @param id идентификатор объекта

\* @param name имя объекта

\* @param coordinates координаты транспортного средства

\* @param creationDate время создания объекта

\* @param enginePower мощь двигателя транспортного средства

\* @param fuelConsumption потребление топлива транспортного средства

\* @param type тип транспортного средства

\* @param fuelType тип топлива, потребляемого транспортным средством

\*/

public Vehicle(int id, String name, Coordinates coordinates, LocalDateTime creationDate, int enginePower, long fuelConsumption, VehicleType type, FuelType fuelType) {

if(id == 0) {

lastId++;

this.id = lastId;

} else {

if(lastId < id) {

lastId = id;

}

this.id = id;

}

this.name = name;

this.coordinates = coordinates;

if(creationDate == null) {

this.creationDate = LocalDateTime.now();

} else {

this.creationDate = creationDate;

}

this.enginePower = enginePower;

this.fuelConsumption = fuelConsumption;

this.type = type;

this.fuelType = fuelType;

}

public String print() {

String result =

"id: " + id + "\n" +

"name: " + name + "\n" +

"coordinates: " + coordinates.getX() + ", " + coordinates.getY() + "\n" +

"creationDate: " + creationDate + "\n" +

"enginePower: " + enginePower + "\n" +

"fuelConsumption: " + fuelConsumption + "\n" +

"type: " + type + "\n" +

"fuelType: " + fuelType + "\n";

return result;

}

/\*\*

\* Возвращает идентификатор объекта

\* @return идентификатор объекта

\*/

public int getId () {

return id;

}

/\*\*

\* Возвращает имя объекта

\* @return имя объекта

\*/

public String getName () {

return name;

}

/\*\*

\* Возвращает тип топлива объекта

\* @return тип топлива

\*/

public FuelType getFuelType () {

return fuelType;

}

/\*\*

\* Возвращает тип транспортного средства

\* @return тип транспортного средства

\*/

public VehicleType getType () {

return type;

}

/\*\*

\* Возвращает мощность двигателя

\* @return мощность двигателя

\*/

public int getEnginePower() {

return enginePower;

}

/\*\*

\* Возвращает потребление топлива

\* @return потребление топлива

\*/

public long getFuelConsumption() {

return fuelConsumption;

}

/\*\*

\* Возвращает время создания объекта

\* @return время создания объекта

\*/

public LocalDateTime getCreationDate() {

return creationDate;

}

/\*\*

\* Возвращает координаты транспортного средства

\* @return координаты транспортного средства

\*/

public Coordinates getCoordinates() {

return coordinates;

}

/\*\*

\* Устанавливает идентификатор объекта

\* @param id идентификатор объекта

\*/

public void setId(int id) {

this.id = id;

}

/\*\*

\* Устанавливает имя объекта

\* @param name имя объекта

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* Устанавливает координаты транспортного средства

\* @param x координата x транспортного средства

\* @param y координата y транспортного средства

\*/

public void setCoordinates(float x, long y) {

this.coordinates = new Coordinates(x, y);

}

/\*\*

\* Устанавливает мощность двигателя

\* @param enginePower мощность двигателя

\*/

public void setEnginePower(int enginePower) {

this.enginePower = enginePower;

}

/\*\*

\* Устанавливает потребление топлива

\* @param fuelConsumption потребление топлива

\*/

public void setFuelConsumption(long fuelConsumption) {

this.fuelConsumption = fuelConsumption;

}

/\*\*

\* Устанавливает тип транспортного средства

\* @param type тип транспортного средства

\*/

public void setType(VehicleType type) {

this.type = type;

}

/\*\*

\* Устанавливает тип топлива

\* @param fuelType тип топлива

\*/

public void setFuelType(FuelType fuelType) {

this.fuelType = fuelType;

}

/\*\*

\* Копирует все параметры указанного объекта в этот экземпляр объекта

\* @param vehicle указанный объект

\*/

public void update (Vehicle vehicle) {

//this.id = vehicle.id; // Оставляем id без изменений.

this.name = vehicle.name;

this.coordinates = vehicle.coordinates;

this.creationDate = vehicle.creationDate;

this.enginePower = vehicle.enginePower;

this.fuelConsumption = vehicle.fuelConsumption;

this.type = vehicle.type;

this.fuelType = vehicle.fuelType;

}

@Override

public int compareTo(Vehicle vehicle) {

int id = vehicle.getId();

if(this.id == id) return 0;

if (this.id < id) return -1;

return 1;

}

}

## VehicleDeserializer.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс VehicleDeserializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class VehicleDeserializer implements JsonDeserializer<Vehicle> {

/\*\* Десериализует объект класса Vehicle

\* @param json Json-элемент, который десериализуют

\* @param typeOfT Тип объекта, в который прооисходит десериализация

\* @param context Контекст десериализации

\* @return Объект класса Vehicle

\*/

@Override

public Vehicle deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) throws JsonParseException {

JsonObject jsonObject = json.getAsJsonObject();

JsonElement element;

int id = 0;

element = jsonObject.get("id");

if(element != null) {

id = element.getAsInt();

}

String name = jsonObject.get("name").getAsString();

Coordinates coordinates = context.deserialize(jsonObject.get("coordinates"), Coordinates.class);

LocalDateTime creationDate = null;

element = jsonObject.get("creationDate");

if(element != null) {

creationDate = context.deserialize(element, LocalDateTime.class);

}

int enginePower = jsonObject.get("enginePower").getAsInt();

long fuelConsumption = jsonObject.get("fuelConsumption").getAsLong();

element = jsonObject.get("type");

VehicleType type = null;

if(element != null) {

type = VehicleType.valueOf(element.getAsString());

}

element = jsonObject.get("fuelType");

FuelType fuelType = null;

if(element != null) {

fuelType = FuelType.valueOf(element.getAsString());

}

return new Vehicle(id, name, coordinates, creationDate, enginePower, fuelConsumption, type, fuelType);

}

}

## VehicleSerializable.java

package data;

import com.google.gson.\*;

import java.io.Serializable;

public class VehicleSerializable implements Serializable {

private String name;

private Float x;

private Long y;

private Integer enginePower;

private Long fuelConsumption;

private VehicleType type;

private FuelType fuelType;

public static VehicleSerializable create(String str) {

try {

Gson gson = Vehicle.getDeserializer();

Vehicle obj = gson.fromJson(str, Vehicle.class);

return new VehicleSerializable(obj.getName(), obj.getCoordinates().getX(), obj.getCoordinates().getY(), obj.getEnginePower(), obj.getFuelConsumption(), obj.getType(), obj.getFuelType());

} catch (Exception e) {

System.out.println("Ошибка чтения объекта : " + e);

return null;

}

}

public VehicleSerializable(String name, float x, long y, int enginePower, long fuelConsumption, VehicleType type, FuelType fuelType) {

this.name = name;

this.x = x;

this.y = y;

this.enginePower = enginePower;

this.fuelConsumption = fuelConsumption;

this.type = type;

this.fuelType = fuelType;

}

public Vehicle toVehicle () {

return new Vehicle(0, name, new Coordinates(x, y), null, enginePower, fuelConsumption, type, fuelType);

}

}

## VehicleSerializer.java

package data;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс VehicleSerializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class VehicleSerializer implements JsonSerializer<Vehicle> {

/\*\* Сериализует объект класса Vehicle

\* @param obj объект класса Vehicle, который сериализуют

\* @param type Тип объекта, для которого происходит сериализация

\* @param context Контекст сериализации

\* @return JsonElement сериализованного объекта

\*/

@Override

public JsonElement serialize(Vehicle obj, Type type, JsonSerializationContext context) {

JsonObject result = new JsonObject();

result.addProperty("id", obj.getId());

result.addProperty("name", obj.getName());

result.add("coordinates", context.serialize(obj.getCoordinates()));

result.add("creationDate", context.serialize(obj.getCreationDate()));

result.addProperty("enginePower", obj.getEnginePower());

result.addProperty("fuelConsumption", obj.getFuelConsumption());

VehicleType vehicleType = obj.getType();

if(vehicleType != null) {

result.addProperty("type", vehicleType.toString());

}

FuelType fuelType = obj.getFuelType();

if(fuelType != null) {

result.addProperty("fuelType", fuelType.toString());

}

return result;

}

}

## VehicleType.java

package data;

/\*\*

\* Класс VehicleType

\* @author Буреев Фёдор

\* @version 1.0

\*/

public enum VehicleType {

/\*\* Самолёт \*/

PLANE,

/\*\* Подводная лодка \*/

SUBMARINE,

/\*\* Лодка \*/

BOAT,

/\*\* Велосипед \*/

BICYCLE,

/\*\* Вертолёт \*/

CHOPPER;

}

# package server:

## CommandHandler.java

package server;

import communication.Request;

import communication.RequestType;

import communication.Response;

import communication.ResponseCode;

import commands.\*;

public class CommandHandler {

private VehicleCollection vehicles;

public CommandHandler(VehicleCollection vehicles) {

this.vehicles = vehicles;

}

public Response doCommand(Request request) {

RequestType type = request.getType();

System.out.println("Получил команду : " + type.toString());

switch(type) {

case INFO: {

//InfoCommand cmd = (InfoCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.info());

}

case SHOW: {

//ShowCommand cmd = (ShowCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.show());

}

case ADD: {

AddCommand cmd = (AddCommand) request.getCommand();

if(vehicles.add(cmd.getObj().toVehicle())) {

return new Response(ResponseCode.SUCCESS, "Элемент успешно добавлен в коллекцию.");

} else {

return new Response(ResponseCode.ERROR, "Не получилось добавить элемент в коллекцию.");

}

}

case UPDATE: {

UpdateCommand cmd = (UpdateCommand) request.getCommand();

if(vehicles.update(cmd.getId(), cmd.getObj().toVehicle())) {

return new Response(ResponseCode.SUCCESS, "Элемент успешно обновлён в коллекции.");

} else {

return new Response(ResponseCode.ERROR, "Объект с этим Id не был найден.");

}

}

case REMOVE\_BY\_ID: {

RemoveByIdCommand cmd = (RemoveByIdCommand) request.getCommand();

if(vehicles.remove(cmd.getId())) {

return new Response(ResponseCode.SUCCESS, "Элемент успешно удалён из коллекции.");

} else {

return new Response(ResponseCode.SUCCESS, "Элемент с этим Id не был найден.");

}

}

case CLEAR: {

//ClearCommand cmd = (ClearCommand) request.getCommand();

vehicles.clear();

return new Response(ResponseCode.SUCCESS, "");

}

case REMOVE\_AT: {

RemoveAtIndexCommand cmd = (RemoveAtIndexCommand) request.getCommand();

if(vehicles.remove\_at(cmd.getIndex())) {

return new Response(ResponseCode.SUCCESS, "Элемент успешно удалён из коллекции.");

} else {

return new Response(ResponseCode.SUCCESS, "Элемент по этому индексу не был найден.");

}

}

case REMOVE\_LOWER: {

RemoveLowerCommand cmd = (RemoveLowerCommand) request.getCommand();

if (vehicles.remove\_lower(cmd.getObj().toVehicle()) ) {

return new Response(ResponseCode.SUCCESS, "Элемент(ы) успешно удалены из коллекции. ");

} else {

return new Response(ResponseCode.SUCCESS, "Элементов для удаления не найдено.");

}

}

case SORT: {

//SortCommand cmd = (SortCommand) request.getCommand();

vehicles.sort();

return new Response(ResponseCode.SUCCESS, "Коллекция отсортирована по параметру EnginePower.");

}

case FILTER\_STARTS\_WITH\_NAME: {

FilterStartsWithNameCommand cmd = (FilterStartsWithNameCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.filter\_starts\_with\_name(cmd.getName()));

}

case PRINT\_UNIQUE\_FUEL\_TYPE: {

//PrintUniqueFuelTypeCommand cmd = (PrintUniqueFuelTypeCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.print\_unique\_fuel\_type());

}

case PRINT\_FIELD\_DESCENDING\_TYPE: {

//PrintFieldDescendingTypeCommand cmd = (PrintFieldDescendingTypeCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.print\_field\_descending\_type());

}

case SAVE: {

//SaveCommand cmd = (SaveCommand) request.getCommand();

if (vehicles.save()) {

return new Response(ResponseCode.SUCCESS, "Коллекция успешно сохранена в файле.");

} else {

return new Response(ResponseCode.ERROR, "Ошибка сохранения коллекции.");

}

}

case EXIT: {

//ExitCommand cmd = (ExitCommand) request.getCommand();

break;

}

}

return new Response(ResponseCode.ERROR, "Неизвестный запрос.");

}

}

## Main.java

package server;

import java.io.\*;

import java.util.Scanner;

import java.util.logging.FileHandler;

import java.util.logging.Logger;

import java.util.logging.SimpleFormatter;

import java.nio.charset.StandardCharsets;

public class Main {

private static int port = 2000;

public static void main(String[] args) {

String fileName = System.getenv("FILEPATH");

if((fileName == null) || fileName.isEmpty()) {

System.out.println("Переменная окружения FILEPATH не задана. Пожалуйста, запишите в FILEPATH путь к файлу для сохранения коллекции.");

return;

}

System.out.println(fileName);

VehicleCollection vehicles = new VehicleCollection(fileName);

String tmpFileName = vehicles.getTmpFileName();

Scanner scanner = new Scanner(System.in);

Logger logger = Logger.getLogger("server");

try {

logger.setUseParentHandlers(false); // Отключает дефолтное логирование в консоль

FileHandler fh = new FileHandler("log.txt", true); //true обеспечивает добавление к файлу, а не затирание файла

logger.addHandler(fh);

SimpleFormatter formatter = new SimpleFormatter();

fh.setFormatter(formatter);

fh.setEncoding(StandardCharsets.UTF\_8.name());

} catch (Exception e) {

System.out.println("Не удалось настроить запись лога в файл.");

}

boolean isLoaded = false;

// Проверка состояния временного файла

File f = new File(tmpFileName);

if (f.exists() && !f.isDirectory()) {

System.out.println("У вас есть несохранённые изменения. Хотите ли вы их загрузить в коллекцию? (Y/N)");

try {

String command = scanner.nextLine();

if((command.equals("Y")) || (command.equals("y"))) {

if(isLoaded = vehicles.load\_tmp()) {

System.out.println("Несохранённые изменения загружены. ");

}

}

} catch (Exception e) {

System.out.println("Ответ не распознан.");

scanner.close();

return;

}

}

// Грузимся из основного файла

if (!isLoaded) {

File f1 = new File(fileName);

if (f1.exists() && !f1.isDirectory()) {

isLoaded = vehicles.load();

if (!isLoaded) {

System.out.println("Коллекцию не удалось загрузить из файла");

scanner.close();

return;

}

}

}

CommandHandler handler = new CommandHandler(vehicles);

try {

System.out.println("Запускаю сервер.");

Server server = new Server(port, handler, logger);

server.run();

} catch (Exception e) {

System.out.println("Аварийное завершение работы сервера : " + e);

}

File tmpFile = new File(tmpFileName);

if(tmpFile.exists() && !tmpFile.isDirectory()) {

if (!tmpFile.delete()) {

System.out.println("Не удалось удалить временный файл " + tmpFileName);

}

}

scanner.close();

}

}

## Server.java

package server;

import java.net.InetSocketAddress;

import java.net.SocketAddress;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import communication.Request;

import communication.RequestType;

import communication.Response;

import commands.SaveCommand;

import java.util.logging.Logger;

import java.nio.ByteBuffer;

import java.nio.channels.DatagramChannel;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

public class Server {

private DatagramChannel sendChannel;

private DatagramChannel receiveChannel;

private Selector readSelector;

private Selector writeSelector;

private SelectionKey readChannelKey;

private SelectionKey writeChannelKey;

private int sendBufferSize = 10000;

private int receiveBufferSize = 10000;

private ByteBuffer buffer;

private CommandHandler handler;

private Logger logger;

private class RequestInfo {

SocketAddress address;

Request request;

public RequestInfo(SocketAddress address, Request request) {

this.address = address;

this.request = request;

}

}

public Server(int port, CommandHandler handler, Logger logger) throws IOException {

buffer = ByteBuffer.allocate(receiveBufferSize);

readSelector = Selector.open();

writeSelector = Selector.open();

receiveChannel = DatagramChannel.open();

receiveChannel.bind(new InetSocketAddress("localhost", port));

receiveChannel.configureBlocking(false);

readChannelKey = receiveChannel.register(readSelector, SelectionKey.OP\_READ);

sendChannel = DatagramChannel.open();

sendChannel.configureBlocking(false);

writeChannelKey = receiveChannel.register(writeSelector, SelectionKey.OP\_WRITE);

this.handler = handler;

this.logger = logger;

}

public void run() throws IOException {

logger.info("Начало работы сервера.");

while (true) {

try {

//System.out.println("Жду команду.");

RequestInfo info = receive();

if(info != null) {

logger.info("Получен новый запрос : " + info.request.getType());

if(info.request.getType() == RequestType.EXIT) {

handler.doCommand(new Request(RequestType.SAVE, new SaveCommand()));

break;

}

send(handler.doCommand(info.request), info.address);

}

} catch (Exception e) {

System.out.println(e);

continue;

}

}

System.out.println("Работа сервера завершена.");

logger.info("Конец работы сервера.");

sendChannel.close();

receiveChannel.close();

}

private RequestInfo receive() {

try {

System.out.println("Жду.\n");

if (readSelector.select() > 0) {

if (readSelector.selectedKeys() != null) {

readSelector.selectedKeys().remove(readChannelKey);

}

SocketAddress address = receiveChannel.receive(buffer);

Request request = (Request) (new ObjectInputStream(new ByteArrayInputStream(buffer.array())).readObject());

buffer.clear();

return new RequestInfo(address, request);

}

System.out.println("Выход из селектора.");

return null;

} catch (Exception e) {

System.out.println("Не получилось прочитать запрос из канала : " + e);

logger.warning("Не получилось прочитать запрос из канала : " + e);

}

return null;

}

private void send(Response response, SocketAddress address) {

try {

if (writeSelector.select() > 0) {

ByteArrayOutputStream byteStream = new ByteArrayOutputStream(sendBufferSize);

ObjectOutputStream stream = new ObjectOutputStream(byteStream);

stream.writeObject(response);

stream.flush();

if (writeSelector.selectedKeys() != null) {

writeSelector.selectedKeys().remove(writeChannelKey);

}

sendChannel.send(ByteBuffer.wrap(byteStream.toByteArray()), address);

logger.info("Отправлен ответ.");

stream.close();

return;

}

System.out.println("Выход из селектора для записи.");

}

catch (Exception e) {

System.out.println("Не получилось записать ответ в сокет : " + e);

logger.warning("Не получилось записать ответ в сокет : " + e);

}

}

}

## VehicleCollection.java

package server;

import java.util.LinkedList;

import java.util.stream.Collectors;

import java.util.Collections;

import java.util.Arrays;

import java.util.Comparator;

import com.google.gson.\*;

import data.FuelType;

import data.SortByEnginePower;

import data.Vehicle;

import java.io.FileOutputStream;

import java.io.InputStreamReader;

import java.io.FileInputStream;

import java.io.BufferedReader;

import java.io.InputStream;

import java.nio.file.Files;

import java.nio.file.attribute.BasicFileAttributes;

import java.nio.file.Paths;

import com.google.gson.reflect.TypeToken;

public class VehicleCollection {

private LinkedList<Vehicle> vehicles;

private boolean isSaved = true;

private String fileName = "";

private String tmpFileName = "temp\_vehicles.txt";

public String getFileName() {

return fileName;

}

public String getTmpFileName() {

return tmpFileName;

}

public boolean isSaved () {

return this.isSaved;

}

public VehicleCollection (String fileName) {

vehicles = new LinkedList<>();

this.fileName = fileName;

}

public String info () {

String result = "Тип коллекции: LinkedList \n" + "Объекты коллекции: Vehicle \n" + "Количество элементов: " + vehicles.size() + "\n";

try {

BasicFileAttributes attr = Files.readAttributes(Paths.get(fileName), BasicFileAttributes.class);

result += ("Дата инициализации: " + attr.creationTime());

} catch (Exception e) {

System.out.println("Не удалось получить дату инициализации коллекции");

}

return result;

}

public String show () {

return vehicles.stream()

.map(vehicle -> vehicle.print())

.collect(Collectors.joining("\n"));

}

private boolean add\_element (Vehicle element) {

if(vehicles.stream().anyMatch(vehicle -> vehicle.getId() == element.getId())) {

System.out.println("Id нового объекта совпал с Id уже существующего объекта. Пожалуйста, поменяйте или уберите Id.");

return false;

}

vehicles.add(element);

return true;

}

public boolean add (Vehicle element) {

if(add\_element(element)) {

this.isSaved = false;

save\_tmp();

return true;

}

return false;

}

public void add (String string) {

Vehicle obj = Vehicle.create(string);

if(obj != null) {

add(obj);

}

}

public boolean update (int id, Vehicle element) {

boolean found = vehicles.stream().anyMatch(vehicle -> {

if(vehicle.getId() == id) {

vehicle.update(element);

return true;

}

return false;

} );

if(found) {

this.isSaved = false;

save\_tmp();

}

return found;

}

public boolean remove (int id) {

if(vehicles.removeIf( vehicle -> (vehicle.getId() == id) )) {

this.isSaved = false;

save\_tmp();

return true;

}

return false;

}

public void clear () {

vehicles.clear();

this.isSaved = false;

save\_tmp();

}

public boolean remove\_at (int index) {

try {

vehicles.remove(index);

this.isSaved = false;

save\_tmp();

return true;

} catch (Exception e) {

return false;

}

}

public boolean remove\_lower (Vehicle element) {

if (vehicles.removeIf( vehicle -> (vehicle.getEnginePower() < element.getEnginePower()) ) ) {

this.isSaved = false;

save\_tmp();

return true;

}

return false;

}

public void sort () {

Collections.sort(vehicles, new SortByEnginePower());

this.isSaved = false;

save\_tmp();

}

public String filter\_starts\_with\_name (String name) {

return vehicles.stream()

.filter(vehicle -> vehicle.getName().startsWith(name))

.map(vehicle -> vehicle.print())

.collect(Collectors.joining("\n"));

}

public String print\_unique\_fuel\_type () {

return Arrays.stream(FuelType.values())

.filter( val -> vehicles.stream().filter(vehicle -> vehicle.getFuelType() == val).count() == 1 )

.map( val -> val.toString())

.collect(Collectors.joining("\n"));

}

public String print\_field\_descending\_type () {

return vehicles.stream()

.map(vehicle -> vehicle.getType())

.filter(val -> val != null)

/\*.sorted(new Comparator<VehicleType>() {

public int compare(VehicleType a, VehicleType b) {

return b.ordinal() - a.ordinal();

}

})\*/

.sorted(Comparator.reverseOrder())

.map( val -> val.toString())

.collect(Collectors.joining("\n"));

}

/\*\*

\* Загружает в коллекцкию элементы из указанного файла

\* @param fileName имя файла

\* @return true, если загрузка прошла успешно, false, если нет

\*/

private boolean load\_elements (String fileName) {

try {

Gson gson = Vehicle.getDeserializer();

InputStream fin = new FileInputStream(fileName);

BufferedReader reader = new BufferedReader(new InputStreamReader(fin));

String line = reader.readLine();

if(line != null) {

vehicles = gson.fromJson(line, new TypeToken<LinkedList<Vehicle>>() {}.getType());

}

reader.close();

return true;

} catch (Exception e) {

System.out.println(e);

}

return false;

}

/\*\*

\* Загружает элементы в коллекцию

\* @return true, если загрузка прошла успешно, false, если нет

\*/

public boolean load () {

return load\_elements(this.fileName);

}

/\*\*

\* Загружает элементы из временного файла в коллекцию

\* @return true, если загрузка прошла успешно, false, если нет

\*/

public boolean load\_tmp () {

this.isSaved = false;

return load\_elements(this.tmpFileName);

}

/\*\*

\* Сохраняет коллекцию в указанный файл

\* @param fileName имя файла

\* @return true, если сохранение прошло успешно, false, если нет

\*/

private boolean save\_elements (String fileName) {

try {

Gson gson = Vehicle.getSerializer();

FileOutputStream fout = new FileOutputStream(fileName);

fout.write(gson.toJson(vehicles).getBytes());

fout.close();

return true;

} catch (Exception e) {

System.out.println(e);

}

return false;

}

/\*\*

\* Сохраняет коллекцию в файл

\*/

public boolean save () {

if(save\_elements(this.fileName)) {

this.isSaved = true;

return true;

}

return false;

}

/\*\*

\* Сохраняет коллекцию во временный файл

\*/

public void save\_tmp () {

save\_elements(this.tmpFileName);

}

}

**4. Выводы по работе**

Я реализовал серверное приложение для управления коллекцией, консольное приложение клиента для посылки команд управления коллекции серверу и консольное приложение администратора для сохранения данных коллекции на сервере и остановки сервера.

Пересылка информации между консольными приложениями и серверным осуществляется по протоколу UDP. Консольные приложения используют посылку датаграмм через DatagramSocket, серверное приложение использует сетевые каналы DatagramChannel в неблокирующем режиме.

Объекты между клиентом и сервером передаются в сериализованном виде.

Отдельные операции обработки объектов коллекции выполнены с помощью Stream API с использованием лямбда-выражений.

Этапы работы сервера записываются в лог-файл с помощью пакета java.util.logging.