Буреев Фёдор Григорьевич P3107

**Лабораторная работа №7**

**вар. 36336**

**1.Доработать программу из лабораторной работы №6 следующим образом:**

1. Организовать хранение коллекции в реляционной СУБД (PostgresQL). Убрать хранение коллекции в файле.
2. Для генерации поля id использовать средства базы данных (sequence).
3. Обновлять состояние коллекции в памяти только при успешном добавлении объекта в БД
4. Все команды получения данных должны работать с коллекцией в памяти, а не в БД
5. Организовать возможность регистрации и авторизации пользователей. У пользователя есть возможность указать пароль.
6. Пароли при хранении хэшировать алгоритмом SHA-1
7. Запретить выполнение команд не авторизованным пользователям.
8. При хранении объектов сохранять информацию о пользователе, который создал этот объект.
9. Пользователи должны иметь возможность просмотра всех объектов коллекции, но модифицировать могут только принадлежащие им.
10. Для идентификации пользователя отправлять логин и пароль с каждым запросом.

**Необходимо реализовать многопоточную обработку запросов.**

1. Для многопоточного чтения запросов использовать Cached thread pool
2. Для многопотчной обработки полученного запроса использовать ForkJoinPool
3. Для многопоточной отправки ответа использовать создание нового потока (java.lang.Thread)
4. Для синхронизации доступа к коллекции использовать потокобезопасные аналоги коллекции из java.util.concurrent

**Порядок выполнения работы:**

1. В качестве базы данных использовать PostgreSQL.
2. Для подключения к БД на кафедральном сервере использовать хост pg, имя базы данных - studs, имя пользователя/пароль совпадают с таковыми для подключения к серверу.

Отчёт по работе должен содержать:

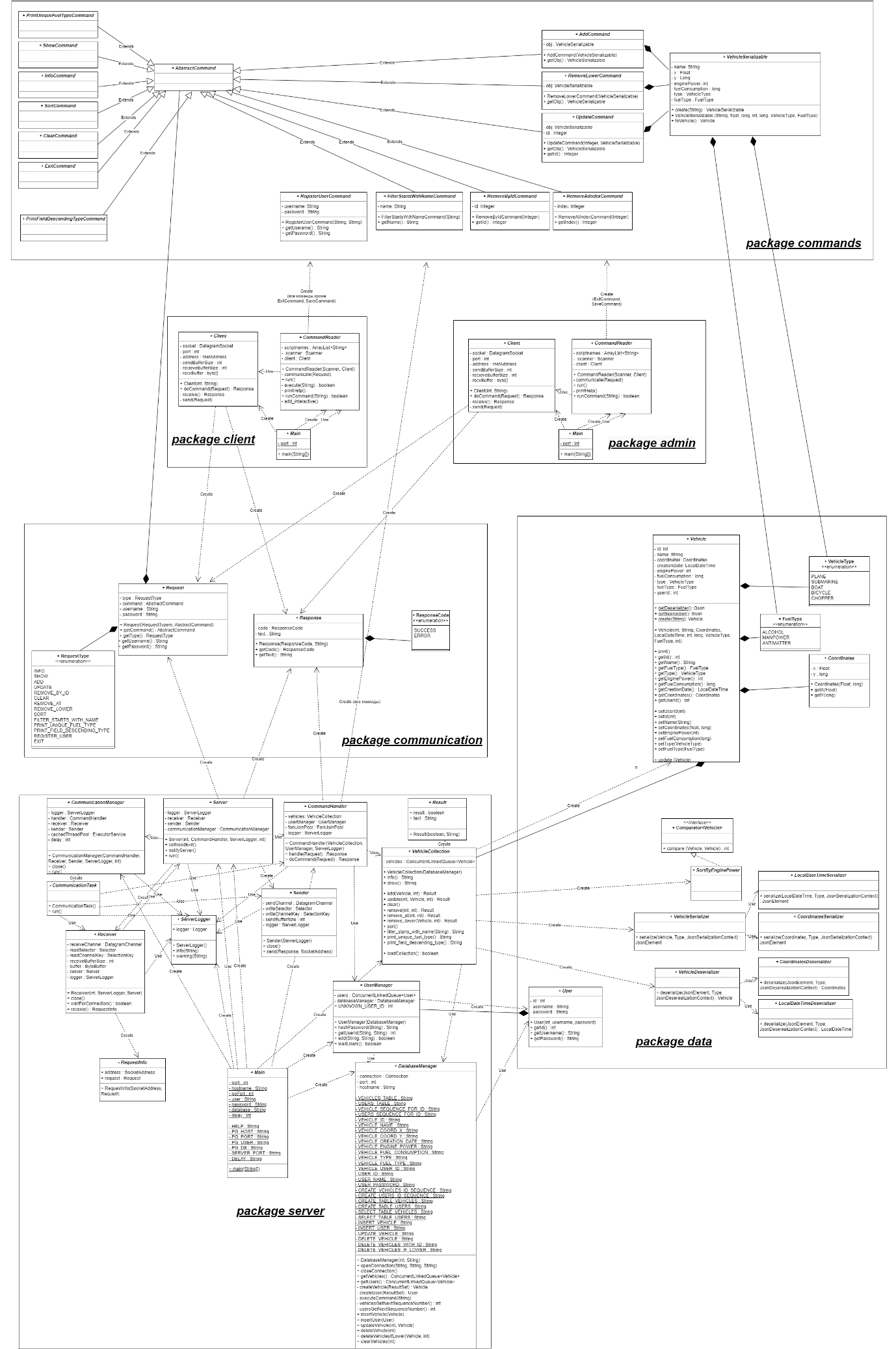
1. Текст задания.
2. Диаграмма классов разработанной программы.
3. Исходный код программы.
4. Выводы по работе.

Вопросы к защите лабораторной работы:

Многопоточность. Класс Thread, интерфейс Runnable. Модификатор synchronized.

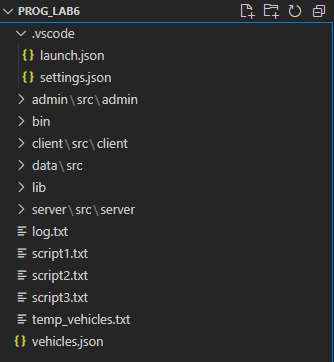
1. Методы wait(), notify() класса Object, интерфейсы Lock и Condition.
2. Классы-сихронизаторы из пакета java.util.concurrent.
3. Модификатор volatile. Атомарные типы данных и операции.
4. Коллекции из пакета java.util.concurrent.
5. Интерфейсы Executor, ExecutorService, Callable, Future
6. Пулы потоков
7. JDBC. Порядок взаимодействия с базой данных. Класс DriverManager. Интерфейс Connection
8. Интерфейсы Statement, PreparedStatement, ResultSet, RowSet
9. Шаблоны проектирования.

**2. Диаграмма классов разработанной программы:**

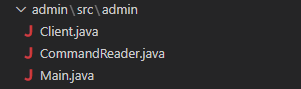
****

**3. Исходный код программы**

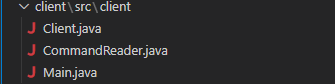
***Список пакетов:***

****

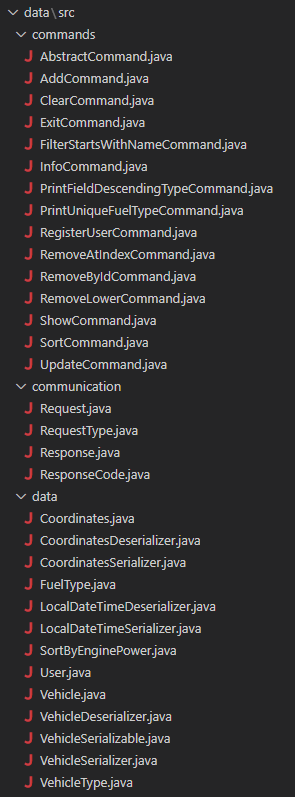
*package admin :*

**

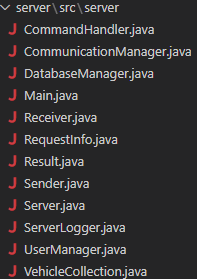
*package client:*

**

*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{

}

## RegisterUserCommand.java

package commands;

public class RegisterUserCommand extends AbstractCommand {

    private String username;

    private String password;

    public RegisterUserCommand(String username, String password) {

        this.username = username;

        this.password = password;

    }

    public String getUsername() {

        return username;

    }

    public String getPassword() {

        return password;

    }

}

## 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;

}

}

## 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;

    private String username;

    private String password;

    public Request(RequestType type, AbstractCommand command, String username, String password) {

        this.type = type;

        this.command = command;

        this.username = username;

        this.password = password;

    }

    public AbstractCommand getCommand() {

        return command;

    }

    public RequestType getType() {

        return type;

    }

    public String getUsername() {

        return username;

    }

    public String getPassword() {

        return password;

    }

}

## 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,

REGISTER\_USER,

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();

}

}

## User.java

package data;

public class User {

    private int id;

    private String username;

    private String password;

    public User(int id, String username, String password) {

        this.id = id;

        this.username = username;

        this.password = password;

    }

    public int getId() {

        return id;

    }

    public String getUsername() {

        return username;

    }

    public String getPassword() {

        return password;

    }

}

## Vehicle.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

public class Vehicle implements Comparable<Vehicle> {

    private int id;

    private String name;

    private Coordinates coordinates;

    private LocalDateTime creationDate;

    private int enginePower;

    private long fuelConsumption;

    private VehicleType type;

    private FuelType fuelType;

    private int userId;

    public static Gson getDeserializer() {

        return new GsonBuilder().

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

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

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

            create();

    }

    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, int userId) {

        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;

        this.userId = userId;

    }

    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;

    }

    public int getUserId() {

        return userId;

    }

    public void setUserId(int id) {

        this.userId = id;

    }

    /\*\*

     \* 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, 0);

}

}

## 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, 0);

}

}

## 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 java.util.concurrent.RecursiveTask;

import java.util.concurrent.ForkJoinPool;

import commands.\*;

public class CommandHandler {

    private VehicleCollection vehicles;

    private UserManager userManager;

    private ForkJoinPool forkJoinPool = ForkJoinPool.commonPool();

    private ServerLogger logger;

    public CommandHandler(VehicleCollection vehicles, UserManager userManager, ServerLogger logger) {

        this.vehicles = vehicles;

        this.userManager = userManager;

        this.logger = logger;

    }

    private class CommandHandlerTask extends RecursiveTask<Response> {

        private Request request;

        public CommandHandlerTask(Request request) {

            this.request = request;

        }

        @Override

        protected Response compute() {

            return doCommand(request);

        }

    }

    public Response handle(Request request) {

        return forkJoinPool.invoke(new CommandHandlerTask(request));

    }

    private synchronized Response doCommand(Request request) {

        RequestType type = request.getType();

        logger.info("CommandHandler получил команду : " + type.toString());

        int userId = UserManager.UNKNOWN\_USER\_ID;

        if(!request.getUsername().isEmpty() || !request.getPassword().isEmpty()) {

            userId = userManager.getUserId(request.getUsername(), request.getPassword());

            if(userId == UserManager.UNKNOWN\_USER\_ID) {

                return new Response(ResponseCode.ERROR, "Пользователь не зарегистрирован");

            }

        }

        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();

                Result res = vehicles.add(cmd.getObj().toVehicle(), userId);

                return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

            }

            case UPDATE: {

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

                Result res = vehicles.update(cmd.getId(), cmd.getObj().toVehicle(), userId);

                return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

            }

            case REMOVE\_BY\_ID: {

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

                Result res = vehicles.remove(cmd.getId(), userId);

                return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

            }

            case CLEAR: {

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

                vehicles.clear(userId);

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

            }

            case REMOVE\_AT: {

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

                Result res = vehicles.remove\_at(cmd.getIndex(), userId);

                return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

            }

            case REMOVE\_LOWER: {

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

                Result res = vehicles.remove\_lower(cmd.getObj().toVehicle(), userId);

                return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

            }

            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 REGISTER\_USER: {

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

                if(userManager.add(cmd.getUsername(), cmd.getPassword())) {

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

                } else {

                    return new Response(ResponseCode.ERROR, "Ошибка регистрации пользователя");

                }

            }

            case EXIT: {

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

                return new Response(ResponseCode.SUCCESS, "Сервер завершает работу");

            }

        }

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

    }

}

## CommunicationManager.java

package server;

import communication.Response;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

import java.lang.Thread;

public class CommunicationManager {

    private ServerLogger logger;

    private CommandHandler handler;

    private Receiver receiver;

    private Sender sender;

    private int delay;

    private int task\_num = 0;

    private ExecutorService cachedThreadPool = Executors.newCachedThreadPool();

    public CommunicationManager(CommandHandler handler, Receiver receiver, Sender sender, ServerLogger logger, int delay) {

        this.handler = handler;

        this.receiver = receiver;

        this.sender = sender;

        this.logger = logger;

        this.delay = delay;

    }

    private class CommunicationTask implements Runnable {

        public CommunicationTask() {}

        @Override

        public void run() {

            logger.info("CommunicationTask : Начало коммуникации");

            RequestInfo info = receiver.receive();

            if(info == null) {

                logger.warning("CommunicationTask : Выход из коммуникации по ошибке");

                return;

            }

            if((delay > 0) && (task\_num % 2 != 0)) {

                logger.info("CommunicationTask : Задержка " + delay + " секунд");

                try {

                    Thread.sleep(delay \* 1000);

                } catch (Exception e) {

                    logger.warning ("CommunicationTask : Ожидание было прервано");

                }

            }

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

            Response response = handler.handle(info.request);

            Thread thread = new Thread(new Runnable() {

                @Override

                public void run() {

                    sender.send(response, info.address);

                }

            });

            thread.start();

            try {

                thread.join();

            } catch (Exception e) {

                logger.warning("CommunicationTask : прерывание выполнения потока" + e);

            }

            logger.info("CommunicationTask : Конец коммуникации");

        }

    }

    public void close() {

        logger.info("CommunicationManager : Подготовка к завершению работы CommunicationManager");

        try {

            cachedThreadPool.shutdown();

            cachedThreadPool.awaitTermination(10, TimeUnit.SECONDS);

            logger.info("CommunicationManager : Работа завершена");

        } catch (InterruptedException e) {

            logger.warning("CommunicationManager : Ошибка завершения работы CommunicationManager : " + e);

        }

    }

    public void run() {

        task\_num++;

        cachedThreadPool.submit(new CommunicationTask());

    }

}

## DatabaseManager.java

package server;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.PreparedStatement;

import java.sql.Statement;

import java.sql.ResultSet;

import java.util.concurrent.ConcurrentLinkedQueue;

import data.Coordinates;

import data.Vehicle;

import data.VehicleType;

import data.FuelType;

import data.User;

import java.time.LocalDateTime;

public class DatabaseManager {

    private Connection connection;

    private int port;

    private String hostname;

    // Tables

    private static final String VEHICLES\_TABLE = "vehicles";

    private static final String USERS\_TABLE = "users";

    private static final String VEHICLE\_SEQUENCE\_FOR\_ID = "vehicle\_sequence";

    private static final String USERS\_SEQUENCE\_FOR\_ID = "users\_sequence";

    // Table Vehicle fields

    private static final String VEHICLE\_ID = "id";

    private static final String VEHICLE\_NAME = "name";

    private static final String VEHICLE\_COORD\_X = "x";

    private static final String VEHICLE\_COORD\_Y = "y";

    private static final String VEHICLE\_CREATION\_DATE = "creation\_date";

    private static final String VEHICLE\_ENGINE\_POWER = "engine\_power";

    private static final String VEHICLE\_FUEL\_CONSUMPTION = "fuel\_consumption";

    private static final String VEHICLE\_TYPE = "vehicle\_type";

    private static final String VEHICLE\_FUEL\_TYPE = "fuel\_type";

    private static final String VEHICLE\_USER\_ID = "user\_id";

    // Table users fields

    private static final String USER\_ID = "id";

    private static final String USER\_NAME = "name";

    private static final String USER\_PASSWORD = "password";

    // Sequence command

    private static final String CREATE\_VEHICLES\_ID\_SEQUENCE = "CREATE SEQUENCE IF NOT EXISTS " + VEHICLE\_SEQUENCE\_FOR\_ID + " START 1 INCREMENT 1";

    private static final String CREATE\_USERS\_ID\_SEQUENCE = "CREATE SEQUENCE IF NOT EXISTS " + USERS\_SEQUENCE\_FOR\_ID + " START 1 INCREMENT 1";

    // Table commands

    private static final String CREATE\_TABLE\_VEHICLES = "CREATE TABLE IF NOT EXISTS " + VEHICLES\_TABLE + " (" +

        VEHICLE\_ID + " integer, " +

        VEHICLE\_NAME + " text, " +

        VEHICLE\_COORD\_X + " real, " +

        VEHICLE\_COORD\_Y + " bigint, " +

        VEHICLE\_CREATION\_DATE + " text, " +

        VEHICLE\_ENGINE\_POWER + " integer, " +

        VEHICLE\_FUEL\_CONSUMPTION + " bigint, " +

        VEHICLE\_TYPE + " text, " +

        VEHICLE\_FUEL\_TYPE + " text, " +

        VEHICLE\_USER\_ID + " integer);";

    private static final String CREATE\_TABLE\_USERS = "CREATE TABLE IF NOT EXISTS " + USERS\_TABLE + " (" +

        USER\_ID + " integer, " +

        USER\_NAME + " text, " +

        USER\_PASSWORD + " text);";

    private static final String SELECT\_TABLE\_VEHICLES = "SELECT \* FROM " + VEHICLES\_TABLE + ";";

    private static final String SELECT\_TABLE\_USERS = "SELECT \* FROM " + USERS\_TABLE + ";";

    private static final String INSERT\_VEHICLE = "INSERT INTO " +

        VEHICLES\_TABLE + " (" +

        VEHICLE\_ID + ", " +

        VEHICLE\_NAME + ", " +

        VEHICLE\_COORD\_X + ", " +

        VEHICLE\_COORD\_Y + ", " +

        VEHICLE\_CREATION\_DATE + ", " +

        VEHICLE\_ENGINE\_POWER + ", " +

        VEHICLE\_FUEL\_CONSUMPTION + ", " +

        VEHICLE\_TYPE + ", " +

        VEHICLE\_FUEL\_TYPE + ", " +

        VEHICLE\_USER\_ID + ") VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);";

    private static final String INSERT\_USER = "INSERT INTO " +

        USERS\_TABLE + " (" +

        USER\_ID + ", " +

        USER\_NAME + ", " +

        USER\_PASSWORD + ") VALUES (?, ?, ?);";

    private static final String UPDATE\_VEHICLE = "UPDATE " +

        VEHICLES\_TABLE + " SET " +

        VEHICLE\_NAME + " = ?, " +

        VEHICLE\_COORD\_X + " = ?, " +

        VEHICLE\_COORD\_Y + " = ?, " +

        VEHICLE\_CREATION\_DATE + " = ?, " +

        VEHICLE\_ENGINE\_POWER + " = ?, " +

        VEHICLE\_FUEL\_CONSUMPTION + " = ?, " +

        VEHICLE\_TYPE + " = ?, " +

        VEHICLE\_FUEL\_TYPE + " = ? WHERE " +

        VEHICLE\_ID + " = ?;";

    /\*private static final String UPDATE\_USER = "UPDATE " +

        USERS\_TABLE + " SET " +

        USER\_ID + " = ?," +

        USER\_NAME + " = ?," +

        USER\_PASSWORD + " ? WHERE " + USER\_ID + " = ?;";

    \*/

    private static final String DELETE\_VEHICLE = "DELETE FROM " +

        VEHICLES\_TABLE + " WHERE " + VEHICLE\_ID + " = ?;";

    /\*private static final String DELETE\_USER = "DELETE FROM " +

        USERS\_TABLE + " WHERE " + USER\_ID + " = ?;";

    \*/

    //private static final String CLEAR\_VEHICLES = "TRUNCATE TABLE " + VEHICLES\_TABLE;

    private static final String DELETE\_VEHICLES\_WITH\_ID = "DELETE FROM " + VEHICLES\_TABLE + " WHERE " + VEHICLE\_USER\_ID + " = ?;";

    private static final String DELETE\_VEHICLE\_IF\_LOWER = "DELETE FROM " +

        VEHICLES\_TABLE + " WHERE " + VEHICLE\_ENGINE\_POWER + " < ? AND " + VEHICLE\_USER\_ID + " = ?;";

    public DatabaseManager(int port, String hostname) {

        this.port = port;

        this.hostname = hostname;

    }

    public void openConnection(String user, String password, String database) throws SQLException, ClassNotFoundException {

        Class.forName("org.postgresql.Driver");

        connection = DriverManager.getConnection("jdbc:postgresql://" + hostname + ":" + port + "/" + database, user, password);

    }

    public void closeConnection() {

        try {

            connection.close();

        } catch (Exception e) {

            System.out.println("Возникла ошибка во время закрытия соединения с базой данных.");

        }

    }

    public ConcurrentLinkedQueue<Vehicle> getVehicles() throws Exception{

        executeCommand(CREATE\_TABLE\_VEHICLES);

        executeCommand(CREATE\_VEHICLES\_ID\_SEQUENCE);

        try {

            ConcurrentLinkedQueue<Vehicle> vehicles = new ConcurrentLinkedQueue<Vehicle>();

            Statement statement = connection.createStatement();

            ResultSet res = statement.executeQuery(SELECT\_TABLE\_VEHICLES);

            while(res.next()) {

                Vehicle vehicle = createVehicle(res);

                vehicles.add(vehicle);

            }

            res.close();

            statement.close();

            return vehicles;

        } catch (Exception e) {

            throw new Exception("Ошибка чтения коллекции vehicles из базы данных : " + e);

        }

    }

    public ConcurrentLinkedQueue<User> getUsers() throws Exception{

        executeCommand(CREATE\_TABLE\_USERS);

        executeCommand(CREATE\_USERS\_ID\_SEQUENCE);

        try {

            ConcurrentLinkedQueue<User> users = new ConcurrentLinkedQueue<User>();

            Statement statement = connection.createStatement();

            ResultSet res = statement.executeQuery(SELECT\_TABLE\_USERS);

            while(res.next()) {

                User user = createUser(res);

                users.add(user);

            }

            res.close();

            statement.close();

            return users;

        } catch (Exception e) {

            throw new Exception("Ошибка чтения коллекции users из базы данных : " + e);

        }

    }

    private Vehicle createVehicle(ResultSet res) throws Exception{

        try {

            int id = res.getInt(VEHICLE\_ID);

            String name = res.getString(VEHICLE\_NAME);

            Float x = res.getFloat(VEHICLE\_COORD\_X);

            Long y = res.getLong(VEHICLE\_COORD\_Y);

            Coordinates coordinates = new Coordinates(x, y);

            LocalDateTime creationDate = LocalDateTime.parse(res.getString(VEHICLE\_CREATION\_DATE));

            int enginePower = res.getInt(VEHICLE\_ENGINE\_POWER);

            long fuelConsumption = res.getLong(VEHICLE\_FUEL\_CONSUMPTION);

            String value;

            value = res.getString(VEHICLE\_TYPE);

            VehicleType type = (value.isEmpty()) ? null : VehicleType.valueOf(value);

            value = res.getString(VEHICLE\_FUEL\_TYPE);

            FuelType fuelType = (value.isEmpty()) ? null : FuelType.valueOf(value);

            int user\_id = res.getInt(VEHICLE\_USER\_ID);

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

        } catch (Exception e) {

            throw new Exception("Ошибка создания объекта Vehicle : " + e);

        }

    }

    private User createUser(ResultSet res) throws Exception{

        try {

            int id = res.getInt(USER\_ID);

            String username = res.getString(USER\_NAME);

            String password = res.getString(USER\_PASSWORD);

            return new User(id, username, password);

        } catch (Exception e) {

            throw new Exception("Ошибка создания объекта User : " + e);

        }

    }

    private void executeCommand(String command) throws Exception{

        try {

            Statement statement = connection.createStatement();

            statement.execute(command);

            statement.close();

        } catch (Exception e) {

            throw new Exception("Ошибка выполнения команды : " + command + " " + e);

        }

    }

    public int vehiclesGetNextSequenceNumber() throws Exception{

        try{

            Statement statement = connection.createStatement();

            ResultSet res = statement.executeQuery("SELECT nextval('" + VEHICLE\_SEQUENCE\_FOR\_ID + "');");

            res.next();

            int value = res.getInt("nextval");

            res.close();

            statement.close();

            return value;

        } catch (Exception e) {

            throw new Exception("Не удалось получить следующий элемент последовательности для базы данных vehicles : " + e);

        }

    }

    public int usersGetNextSequenceNumber() throws Exception{

        try{

            Statement statement = connection.createStatement();

            ResultSet res = statement.executeQuery("SELECT nextval('" + USERS\_SEQUENCE\_FOR\_ID + "');");

            res.next();

            int value = res.getInt("nextval");

            res.close();

            statement.close();

            return value;

        } catch (Exception e) {

            throw new Exception("Не удалось получить следующий элемент последовательности для базы данных users: " + e);

        }

    }

    public void insertVehicle(Vehicle vehicle) throws Exception {

        PreparedStatement statement = null;

        try {

            statement = connection.prepareStatement(INSERT\_VEHICLE);

            statement.setInt(1, vehicle.getId());

            statement.setString(2, vehicle.getName());

            statement.setFloat(3, vehicle.getCoordinates().getX());

            statement.setLong(4, vehicle.getCoordinates().getY());

            statement.setString(5, vehicle.getCreationDate().toString());

            statement.setInt(6, vehicle.getEnginePower());

            statement.setLong(7, vehicle.getFuelConsumption());

            statement.setString(8, (vehicle.getType() == null) ? "" : vehicle.getType().toString());

            statement.setString(9, (vehicle.getFuelType() == null) ? "" : vehicle.getFuelType().toString());

            statement.setInt(10, vehicle.getUserId());

            //System.out.println(statement);

            statement.execute();

        } catch (Exception e) {

            throw new Exception("Не удалось добавить Vehicle в базу данных : " + e);

        }

    }

    public void insertUser(User user) throws Exception {

        PreparedStatement statement = null;

        try {

            statement = connection.prepareStatement(INSERT\_USER);

            statement.setInt(1, user.getId());

            statement.setString(2, user.getUsername());

            statement.setString(3, user.getPassword());

            //System.out.println(statement);

            statement.execute();

        } catch (Exception e) {

            throw new Exception("Не удалось добавить User в базу данных : " + e);

        }

    }

    public void updateVehicle(int id, Vehicle vehicle) throws Exception {

        PreparedStatement statement = null;

        try {

            statement = connection.prepareStatement(UPDATE\_VEHICLE);

            statement.setString(1, vehicle.getName());

            statement.setFloat(2, vehicle.getCoordinates().getX());

            statement.setLong(3, vehicle.getCoordinates().getY());

            statement.setString(4, vehicle.getCreationDate().toString());

            statement.setInt(5, vehicle.getEnginePower());

            statement.setLong(6, vehicle.getFuelConsumption());

            statement.setString(7, (vehicle.getType() == null) ? "" : vehicle.getType().toString());

            statement.setString(8, (vehicle.getFuelType() == null) ? "" : vehicle.getFuelType().toString());

            statement.setInt(9, id);

            //System.out.println(statement);

            statement.execute();

        } catch (Exception e) {

            throw new Exception("Не удалось обновить Vehicle в базу данных : " + e);

        }

    }

    public void deleteVehicle(int id) throws Exception {

        PreparedStatement statement = null;

        try {

            statement = connection.prepareStatement(DELETE\_VEHICLE);

            statement.setInt(1, id);

            statement.execute();

        } catch (Exception e) {

            throw new Exception("Не удалось удалить Vehicle из базы данных : " + e);

        }

    }

    public void deleteVehiclesIfLower(Vehicle vehicle, int userId) throws Exception {

        PreparedStatement statement = null;

        try {

            statement = connection.prepareStatement(DELETE\_VEHICLE\_IF\_LOWER);

            statement.setInt(1, vehicle.getEnginePower());

            statement.setInt(2, userId);

            statement.execute();

        } catch (Exception e) {

            throw new Exception("Не удалось удалить Vehicle из базы данных : " + e);

        }

    }

    public void clearVehicles(int userId) throws Exception {

        PreparedStatement statement = null;

        try {

            statement = connection.prepareStatement(DELETE\_VEHICLES\_WITH\_ID);

            statement.setInt(1, userId);

            statement.execute();

        } catch (Exception e) {

            throw new Exception("Не удалось удалить vehicles данного пользователя из базы данных : " + e);

        }

    }

}

## Main.java

package server;

public class Main {

    private static int port = 2000;

    private static String hostname = "localhost";

    private static int pgPort = 5432;

    private static String user = "postgres";

    private static String password = "1234";

    private static String database = "postgres";

    private static int delay = 0;

    private static final String HELP = "help";

    private static final String PG\_HOST = "pg\_host=";

    private static final String PG\_PORT = "pg\_port=";

    private static final String PG\_USER = "pg\_user=";

    private static final String PG\_PASS = "pg\_pass=";

    private static final String PG\_DB = "pg\_db=";

    private static final String SERVER\_PORT = "srv\_port=";

    private static final String DELAY = "delay=";

    public static void main(String[] args) {

        if((args.length == 1) && (args[0].equals(HELP))) {

            System.out.println("Параметры запуска программы : ");

            System.out.println("    help                      вывод справочной информации");

            System.out.println("    pg\_host=<hostname>        имя хоста, по умолчанию - localhost");

            System.out.println("    pg\_port=<port>            номер порта, по умолчанию - 5432");

            System.out.println("    pg\_user=<user>            имя пользователя postgres, по умолчанию - postgres");

            System.out.println("    pg\_pass=<password>        пароль пользователя postgres, по умолчанию - 1234");

            System.out.println("    pg\_db=<databse>           имя базы данных postgres, по умолчанию - postgres");

            System.out.println("    srv\_port=<port>           порт, на котором запущен сервер, по умолчанию - 2000");

            System.out.println("    delay=<delay>             задержка обработки запроса на указанное кол-во секунд, по умолчанию - 0");

            return;

        }

        try {

            for(String str : args) {

                if(str.startsWith(PG\_HOST)) hostname = str.substring(PG\_HOST.length());

                else if(str.startsWith(PG\_PORT)) pgPort = Integer.parseInt(str.substring(PG\_PORT.length()));

                else if(str.startsWith(PG\_USER)) user = str.substring(PG\_USER.length());

                else if(str.startsWith(PG\_PASS)) password = str.substring(PG\_PASS.length());

                else if(str.startsWith(PG\_DB)) database = str.substring(PG\_DB.length());

                else if(str.startsWith(SERVER\_PORT)) port = Integer.parseInt(str.substring(SERVER\_PORT.length()));

                else if(str.startsWith(DELAY)) delay = Integer.parseInt(str.substring(DELAY.length()));

                else {

                    System.out.println("Неправильно заданы аргументы командной строки");

                    return;

                }

            }

        } catch (Exception e) {

            System.out.println("Неправильно заданы аргументы командной строки");

            return;

        }

        ServerLogger logger = new ServerLogger();

        DatabaseManager databaseManager = new DatabaseManager(pgPort, hostname);

        try {

            databaseManager.openConnection(user, password, database);

            System.out.println("Подключение к базе данных установлено.");

        } catch (Exception e) {

            System.out.println("Не удалось подключиться к базе данных." + e);

            return;

        }

        UserManager userManager = new UserManager(databaseManager);

        if(userManager.loadUsers()) {

            System.out.println("Успешно загрузили коллекцию пользователей из базы данных.");

        } else {

            System.out.println("Ошибка загрузки коллекции пользователей из базы данных.");

            databaseManager.closeConnection();

            return;

        }

        VehicleCollection vehicles = new VehicleCollection(databaseManager);

        if(vehicles.loadCollection()) {

            System.out.println("Успешно загрузили коллекцию vehicles из базы данных.");

        } else {

            System.out.println("Ошибка загрузки коллекции vehicles из базы данных.");

            databaseManager.closeConnection();

            return;

        }

        CommandHandler handler = new CommandHandler(vehicles, userManager, logger);

        try {

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

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

            server.run();

        } catch (Exception e) {

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

        }

        databaseManager.closeConnection();

    }

}

## Receiver.java

package server;

import java.net.InetSocketAddress;

import java.net.SocketAddress;

import java.io.IOException;

import java.nio.ByteBuffer;

import java.nio.channels.DatagramChannel;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

import java.io.ObjectInputStream;

import java.io.ByteArrayInputStream;

import communication.Request;

import communication.RequestType;

public class Receiver {

    private DatagramChannel receiveChannel;

    private Selector readSelector;

    private SelectionKey readChannelKey;

    private int receiveBufferSize = 10000;

    private ByteBuffer buffer;

    private Server server;

    private ServerLogger logger;

    public Receiver(int port, ServerLogger logger, Server server) throws IOException {

        buffer = ByteBuffer.allocate(receiveBufferSize);

        readSelector = Selector.open();

        receiveChannel = DatagramChannel.open();

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

        receiveChannel.configureBlocking(false);

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

        this.logger = logger;

        this.server = server;

    }

    public void close() {

        try {

            receiveChannel.close();

        } catch (Exception e) {

            logger.warning("Receiver : Ошибка закрытия канала приёма : " + e);

        }

    }

    public boolean waitForConnection() {

        logger.info("Receiver : Жду");

        try {

            if(readSelector.select() > 0) {

                return true;

            } else {

                logger.warning("Receiver : Выход из селектора");

                return false;

            }

        } catch (Exception e) {

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

        }

        return false;

    }

    public synchronized RequestInfo receive() {

        try {

            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();

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

                server.setNeedExit();

                logger.info("Receiver : Получен запрос на завершение работы сервера");

            }

            server.notifyServer();

            return new RequestInfo(address, request);

        } catch (Exception e) {

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

        }

        server.notifyServer();

        return null;

    }

}

## RequestInfo.java

package server;

import communication.Request;

import java.net.SocketAddress;

public class RequestInfo {

    SocketAddress address;

    Request request;

    public RequestInfo(SocketAddress address, Request request) {

        this.address = address;

        this.request = request;

    }

}

## Result.java

package server;

public class Result {

    public boolean result;

    public String text;

    public Result(boolean result, String text) {

        this.result = result;

        this.text = text;

    }

}

## Sender.java

package server;

import java.net.SocketAddress;

import java.io.ObjectOutputStream;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import communication.Response;

import java.nio.ByteBuffer;

import java.nio.channels.DatagramChannel;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

public class Sender {

    private DatagramChannel sendChannel;

    private Selector writeSelector;

    private SelectionKey writeChannelKey;

    private int sendBufferSize = 10000;

    private ServerLogger logger;

    public Sender (ServerLogger logger) throws IOException {

        writeSelector = Selector.open();

        sendChannel = DatagramChannel.open();

        sendChannel.configureBlocking(false);

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

        this.logger = logger;

    }

    public void close() {

        try {

            sendChannel.close();

        } catch (Exception e) {

            logger.warning("Sender : Ошибка закрытия канала передачи : " + e);

        }

    }

    public synchronized void send(Response response, SocketAddress address) {

        logger.info("Sender : Посылаю ответ.");

        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("Sender : Отправлен ответ");

                stream.close();

                return;

            }

            logger.warning("Sender : Выход из селектора для записи.");

        }

        catch (Exception e) {

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

        }

    }

}

## Server.java

package server;

import java.io.IOException;

public class Server {

    private ServerLogger logger;

    private Receiver receiver;

    private Sender sender;

    private CommunicationManager communicationManager;

    private volatile boolean needExit = false;

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

        receiver = new Receiver(port, logger, this);

        sender = new Sender(logger);

        communicationManager = new CommunicationManager(handler, receiver, sender, logger, delay);

        this.logger = logger;

    }

    public void setNeedExit() {

        needExit = true;

    }

    public synchronized void notifyServer() {

        this.notify();

    }

    public synchronized void run() {

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

        while (!needExit) {

            if(receiver.waitForConnection()) {

                communicationManager.run();

                try {

                    this.wait();

                } catch (Exception e) {

                    logger.warning("Server : Ожидание коммуникации завершилось ошибкой : " + e);

                }

            }

        }

        logger.info("Server : Подготовка к завершению работы сервера");

        communicationManager.close();

        receiver.close();

        sender.close();

        logger.info("Server : Работа сервера завершена.");

    }

}

## ServerLogger.java

package server;

import java.util.logging.FileHandler;

import java.util.logging.Logger;

import java.util.logging.SimpleFormatter;

import java.nio.charset.StandardCharsets;

public class ServerLogger {

    Logger logger;

    public ServerLogger () {

        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.");

        }

    }

    public void info (String str) {

        logger.info(Thread.currentThread().getName() + " : " + str);

        System.out.println(Thread.currentThread().getName() + " : " + str);

    }

    public void warning (String str) {

        logger.warning(Thread.currentThread().getName() + " : " + str);

        System.out.println(Thread.currentThread().getName() + " : " + str);

    }

}

## UserManager.java

package server;

import java.util.Optional;

import java.util.concurrent.ConcurrentLinkedQueue;

import data.User;

import java.security.MessageDigest;

public class UserManager {

    private ConcurrentLinkedQueue<User> users;

    private DatabaseManager databaseManager;

    public static int UNKNOWN\_USER\_ID = 0;

    public UserManager (DatabaseManager databaseManager) {

        users = new ConcurrentLinkedQueue<>();

        this.databaseManager = databaseManager;

    }

    private String hashPassword(String password) {

        try {

            MessageDigest sha1 = MessageDigest.getInstance("SHA-1");

            byte[] bytes = sha1.digest(password.getBytes());

            String result = "";

            for (byte b : bytes) {

                result += String.format("%02x", b);

            }

            //System.out.println(password + " -> " + result);

            return result;

        } catch(Exception e) {

            System.out.println("Ошибка хэширования пароля : " + e);

        }

        return password;

    }

    public int getUserId(String username, String password) {

        String hash = hashPassword(password);

        Optional<User> result = users.stream()

            .filter(user -> (user.getUsername().equals(username) && user.getPassword().equals(hash)))

            .findFirst();

        if(result.isPresent()) {

            return result.get().getId();

        } else {

            return UNKNOWN\_USER\_ID;

        }

    }

    public boolean add (String username, String password) {

        try {

            int userId = getUserId(username, password);

            if(userId == UNKNOWN\_USER\_ID) {

                String hash = hashPassword(password);

                userId = databaseManager.usersGetNextSequenceNumber();

                User user = new User(userId, username, hash);

                databaseManager.insertUser(user);

                users.add(user);

            } else {

                System.out.println("Пользователь " + username + " уже существует");

            }

            return true;

        } catch (Exception e) {

            System.out.println("Ошибка добавления пользователя : " + e);

            return false;

        }

    }

    public boolean loadUsers() {

        try {

            users = databaseManager.getUsers();

        } catch (Exception e) {

            System.out.println(e);

            return false;

        }

        return true;

    }

}

## 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 java.util.Optional;

import java.util.concurrent.ConcurrentLinkedQueue;

import data.FuelType;

import data.SortByEnginePower;

import data.Vehicle;

public class VehicleCollection {

    private ConcurrentLinkedQueue<Vehicle> vehicles;

    private DatabaseManager databaseManager;

    public VehicleCollection (DatabaseManager databaseManager) {

        vehicles = new ConcurrentLinkedQueue<>();

        this.databaseManager = databaseManager;

    }

    public String info () {

        return "Тип коллекции: ConcurrentLinkedQueue \n" + "Объекты коллекции: Vehicle \n" + "Количество элементов: " + vehicles.size() + "\n";

    }

    public String show () {

        return vehicles.stream()

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

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

    }

    public Result add (Vehicle element, int userId) {

        try {

            int value = databaseManager.vehiclesGetNextSequenceNumber();

            element.setId(value);

            element.setUserId(userId);

            databaseManager.insertVehicle(element);

            vehicles.add(element);

            return new Result(true, "Объект успешно добавлен в коллекцию.");

        } catch (Exception e) {

            return new Result(false, "Не получилось добавить объект в коллекцию.");

        }

    }

    public Result update (int id, Vehicle element, int userId) {

        Optional<Vehicle> result = vehicles.stream()

            .filter(vehicle -> (vehicle.getId() == id))

            .findFirst();

        if(result.isPresent()) {

            if(result.get().getUserId() != userId) {

                return new Result(false, "Объект принадлежит другому пользователю");

            }

        } else {

            return new Result(false, "Объект с этим id не был найден");

        }

        try {

            databaseManager.updateVehicle(id, element);

            result.get().update(element);

            return new Result(true, "Объект успешно обновлён");

        } catch (Exception E) {

            return new Result(false, "Ошибка обновления объекта");

        }

    }

    public void clear (int userId) {

        try {

            databaseManager.clearVehicles(userId);

            vehicles.removeIf( vehicle -> (vehicle.getUserId() == userId));

        } catch (Exception E) {

            System.out.println("Не получилось очистить таблицу Vehicles в базе данных");

        }

    }

    public Result remove (int id, int userId) {

        Optional<Vehicle> result = vehicles.stream()

            .filter(vehicle -> (vehicle.getId() == id))

            .findFirst();

        if(result.isPresent()) {

            if(result.get().getUserId() != userId) {

                return new Result(false, "Объект принадлежит другому пользователю");

            }

        } else {

            return new Result(true, "Объект с этим id не был найден");

        }

        try {

            databaseManager.deleteVehicle(id);

            vehicles.removeIf( vehicle -> (vehicle.getId() == id) );

            return new Result(true, "Объект успешно удалён");

        } catch (Exception E) {

            return new Result(false, "Ошибка удаления объекта");

        }

    }

    public Result remove\_at (int index, int userId) {

        int count = 0;

        Vehicle vehicle = null;

        for(Vehicle v : vehicles) {

            if(count != index) {

                count++;

            } else {

                vehicle = v;

                break;

            }

        }

        if(vehicle == null) return new Result(false, "Объект по этому индексу не был найден : " + index);

        try {

            databaseManager.deleteVehicle(vehicle.getId());

            vehicles.remove(vehicle);

            return new Result(true, "Объект успешно удалён из коллекции.");

        } catch (Exception E) {

            return new Result(false, "Ошибка удаления объекта");

        }

    }

    public Result remove\_lower (Vehicle element, int userId) {

        try {

            databaseManager.deleteVehiclesIfLower(element, userId);

            vehicles.removeIf( vehicle -> ((vehicle.getEnginePower() < element.getEnginePower()) && (vehicle.getUserId() == userId)));

            return new Result(true, "Объект(ы) успешно удалены из коллекции");

        } catch (Exception E) {

            return new Result(false, "Ошибка удаления объектов");

        }

    }

    public void sort () {

        LinkedList<Vehicle> vehicleList = new LinkedList<Vehicle>(vehicles);

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

        vehicles = new ConcurrentLinkedQueue<Vehicle>(vehicleList);

    }

    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(Comparator.reverseOrder())

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

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

    }

    public boolean loadCollection() {

        try {

            vehicles = databaseManager.getVehicles();

        } catch (Exception e) {

            System.out.println(e);

            return false;

        }

        return true;

    }

}

**4. Выводы по работе**

Я изменил устройство программы, убрав хранение коллекции в файле и заменив его на хранение в базе данных. Программа может получать аргументы, диктующие параметры подключения к базе данных, и может создавать и работать с коллекцией в этой базе данных.

Программа состоит из трёх приложений: приложение для управления коллекцией, консольное приложение клиента для посылки команд управления коллекции серверу и консольное приложение администратора для регистрации пользователей и выключения сервера. Пользователь, незарегистрированный в таблице пользователей базы данных, не может пользоваться командами. Также, при хранении объектов коллекции хранится id создателя, и каждый пользователь не может менять объекты других пользователей.

Пароли пользователей хранятся в базе данных в специальной таблице пользователей, и их пароли хэшируются с помощью алгоритма SHA-1.

ID объектов коллекции и пользователей, и транспортных средств, генерируются с помощью sequence (механизма postgresql).

Чтение каждого запроса выполняется многопоточно с помощью cached thread pool.

Обработка запросов осуществляется с помощью ForkJoinPool.

Для многопоточной отправки ответов используется создание нового потока (Thread) для каждого запроса.

Для синхронизации доступа к коллекции используется потокобезопасный аналог коллекции из java.util.concurrent: ConcurrentLinkedQueue