Буреев Фёдор Григорьевич P3107

**Лабораторная работа №5**

**вар. 335998**

**1. Задание:**

Реализовать консольное приложение, которое реализует управление коллекцией объектов в интерактивном режиме. В коллекции необходимо хранить объекты класса Vehicle, описание которого приведено ниже.

**Разработанная программа должна удовлетворять следующим требованиям:**

* Класс, коллекцией экземпляров которого управляет программа, должен реализовывать сортировку по умолчанию.
* Все требования к полям класса (указанные в виде комментариев) должны быть выполнены.
* Для хранения необходимо использовать коллекцию типа java.util.LinkedList
* При запуске приложения коллекция должна автоматически заполняться значениями из файла.
* Имя файла должно передаваться программе с помощью: **переменная окружения**.
* Данные должны храниться в файле в формате json
* Чтение данных из файла необходимо реализовать с помощью класса java.io.InputStreamReader
* Запись данных в файл необходимо реализовать с помощью класса java.io.FileOutputStream
* Все классы в программе должны быть задокументированы в формате javadoc.
* Программа должна корректно работать с неправильными данными (ошибки пользовательского ввода, отсутсвие прав доступа к файлу и т.п.).

**В интерактивном режиме программа должна поддерживать выполнение следующих команд:**

* help : вывести справку по доступным командам
* info : вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.)
* show : вывести в стандартный поток вывода все элементы коллекции в строковом представлении
* add {element} : добавить новый элемент в коллекцию
* update id {element} : обновить значение элемента коллекции, id которого равен заданному
* remove\_by\_id id : удалить элемент из коллекции по его id
* clear : очистить коллекцию
* save : сохранить коллекцию в файл
* 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.
* exit : завершить программу (без сохранения в файл)
* remove\_at index : удалить элемент, находящийся в заданной позиции коллекции (index)
* remove\_lower {element} : удалить из коллекции все элементы, меньшие, чем заданный
* sort : отсортировать коллекцию в естественном порядке
* filter\_starts\_with\_name name : вывести элементы, значение поля name которых начинается с заданной подстроки
* print\_unique\_fuel\_type : вывести уникальные значения поля fuelType всех элементов в коллекции
* print\_field\_descending\_type : вывести значения поля type всех элементов в порядке убывания

**Формат ввода команд:**

* Все аргументы команды, являющиеся стандартными типами данных (примитивные типы, классы-оболочки, String, классы для хранения дат), должны вводиться в той же строке, что и имя команды.
* Все составные типы данных (объекты классов, хранящиеся в коллекции) должны вводиться по одному полю в строку.
* При вводе составных типов данных пользователю должно показываться приглашение к вводу, содержащее имя поля (например, "Введите дату рождения:")
* Если поле является enum'ом, то вводится имя одной из его констант (при этом список констант должен быть предварительно выведен).
* При некорректном пользовательском вводе (введена строка, не являющаяся именем константы в enum'е; введена строка вместо числа; введённое число не входит в указанные границы и т.п.) должно быть показано сообщение об ошибке и предложено повторить ввод поля.
* Для ввода значений null использовать пустую строку.
* Поля с комментарием "Значение этого поля должно генерироваться автоматически" не должны вводиться пользователем вручную при добавлении.

**Описание хранимых в коллекции классов:**

public class Vehicle {

private long id; //Значение поля должно быть больше 0, Значение этого поля должно быть уникальным, Значение этого поля должно генерироваться автоматически

private String name; //Поле не может быть null, Строка не может быть пустой

private Coordinates coordinates; //Поле не может быть null

private java.time.ZonedDateTime creationDate; //Поле не может быть null, Значение этого поля должно генерироваться автоматически

private float enginePower; //Значение поля должно быть больше 0

private double fuelConsumption; //Значение поля должно быть больше 0

private VehicleType type; //Поле может быть null

private FuelType fuelType; //Поле может быть null

}

public class Coordinates {

private Float x; //Поле не может быть null

private long y;

}

public enum VehicleType {

PLANE,

SUBMARINE,

BOAT,

BICYCLE,

CHOPPER;

}

public enum FuelType {

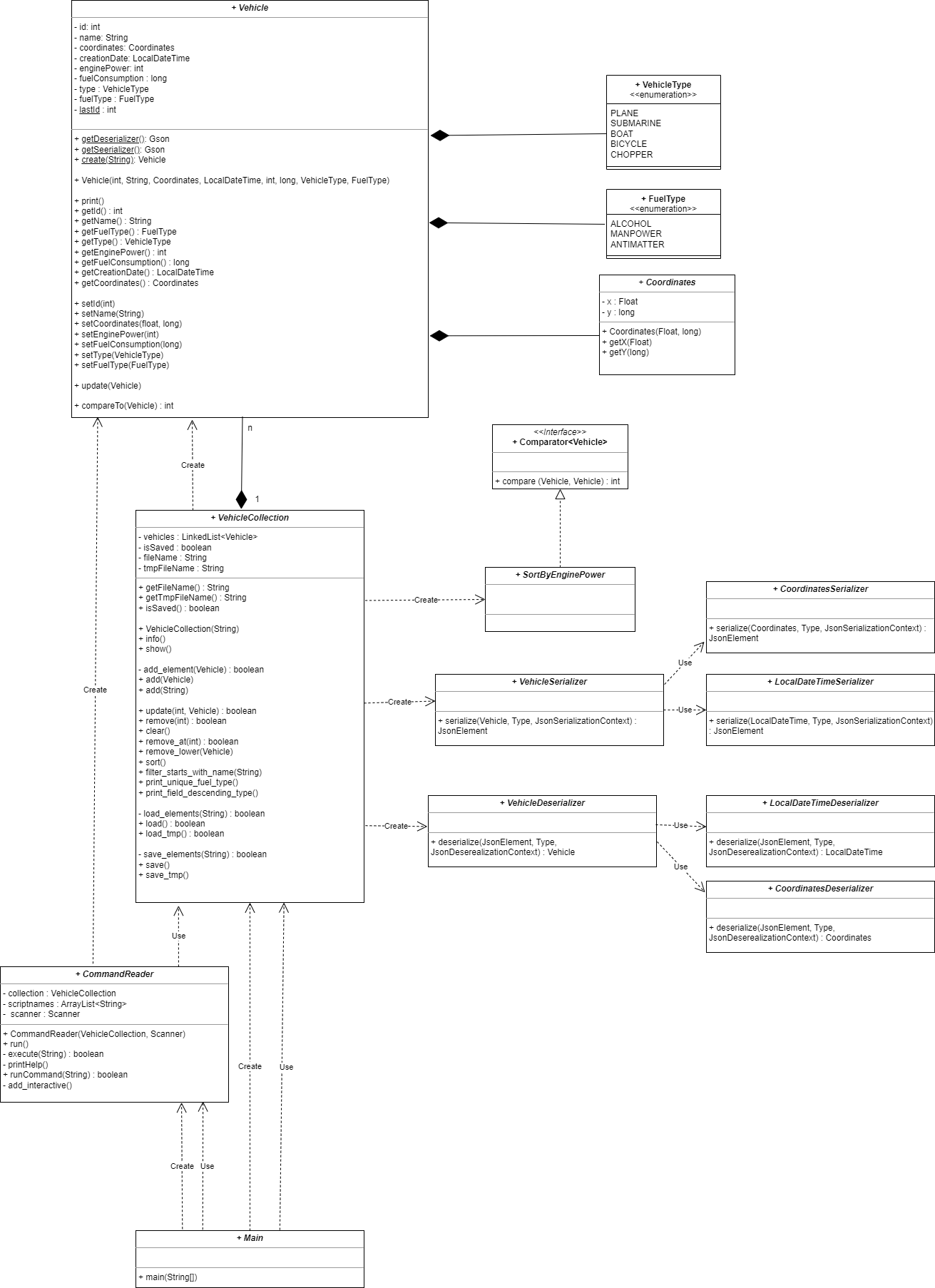
ALCOHOL,

MANPOWER,

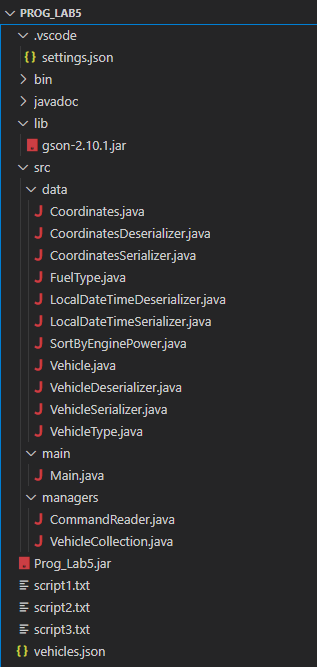
ANTIMATTER;

}

**2. Диаграмма классов разработанной программы:**

****

**3. Исходный код программы:**

****

# Coordinates.java

package data;

/\*\*

\* Класс Coordinates

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class Coordinates {

/\*\* Поле Координата x \*/

private Float x; //Поле не может быть null

/\*\* Поле Координата y \*/

private long y;

/\*\* Конструктор объекта класса Coordinates

\* @param x Координата x

\* @param y Координата y

\*/

public Coordinates(Float x, long y)

{

this.x = x;

this.y = y;

}

/\*\* Возвращает значение координаты x

\* @return Координата x

\*/

public Float getX() {

return this.x;

}

/\*\* Возвращает значение координаты y

\* @return Координата y

\*/

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;

}

/\*\*

\* Печатает свойства объекта

\*/

public void print() {

System.out.println("id: " + id);

System.out.println("name: " + name);

System.out.println("coordinates: " + coordinates.getX() + ", " + coordinates.getY());

System.out.println("creationDate: " + creationDate);

System.out.println("enginePower: " + enginePower);

System.out.println("fuelConsumption: " + fuelConsumption);

System.out.println("type: " + type);

System.out.println("fuelType: " + fuelType);

}

/\*\*

\* 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);

}

}

# 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;

}

# Main.java

package main;

import java.io.\*;

import java.util.Scanner;

import managers.CommandReader;

import managers.VehicleCollection;

/\*\*

\* Класс Main

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class Main {

/\*\*

\* Запускает программу

\* @param args Параметры командной строки (не используются)

\* @version 1.0

\*/

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);

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;

}

}

}

CommandReader reader = new CommandReader(vehicles, scanner);

reader.run();

File tmpFile = new File(tmpFileName);

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

if (!tmpFile.delete()) {

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

}

}

scanner.close();

}

}

# CommandReader.java

package managers;

import java.util.Scanner;

import data.Coordinates;

import data.FuelType;

import data.Vehicle;

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;

/\*\*

\* Класс CommandReader

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class CommandReader {

/\*\* Поле коллекция \*/

private VehicleCollection collection;

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

private ArrayList<String> scriptnames;

/\*\* Поле сканнера \*/

private Scanner scanner;

/\*\* Конструктор

\* @param collection коллекция объектов класса Vehicle

\* @param scanner сканнер для ввода комманд

\*/

public CommandReader(VehicleCollection collection, Scanner scanner) {

this.collection = collection;

scriptnames = new ArrayList<String>();

this.scanner = scanner;

}

/\*\* Выполняет команды управления коллекцией, читая их из командной строки \*/

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)) {

if(collection.isSaved()) {

break;

}

System.out.println("Вы не сохранили коллекцию перед выходом. Вы уверены, что хотите выйти? (Y/N)");

try {

command = scanner.nextLine();

if(command != null) {

command = command.trim();

}

} catch (Exception e) {

command = "";

}

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

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("save : сохранить коллекцию в файл");

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 (без сохранения в файл)");

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");

}

/\*\* Исполняет указанную команду управления коллекцией

\* @param command команда управления коллекцией

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

\*/

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":

collection.info();

break;

case "show":

collection.show();

break;

case "add":

if(args.length == 1) {

add\_interactive();

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

try {

collection.add(args[1]);

} 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]);

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

if(obj != null) {

if(!collection.update(id, obj)) {

System.out.println("Объект с таким id не был найден : " + id);

}

}

} catch (Exception e) {

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

}

} else {

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

}

break;

case "clear":

collection.clear();

break;

case "save":

collection.save();

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]);

if (!collection.remove(id)) {

System.out.println("Объект с этим Id не найден : " + 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]);

if(!collection.remove\_at(idx)) {

System.out.println("Объект с этим индексом не найден : " + 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) {

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

if(obj != null) {

collection.remove\_lower(obj);

}

} else {

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

}

break;

case "sort":

collection.sort();

break;

case "filter\_starts\_with\_name":

if(args.length == 2)

collection.filter\_starts\_with\_name(args[1]);

else

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

break;

case "print\_unique\_fuel\_type":

collection.print\_unique\_fuel\_type();

break;

case "print\_field\_descending\_type":

collection.print\_field\_descending\_type();

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");

}

}

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

}

}

# VehicleCollection.java

package managers;

import java.util.LinkedList;

import java.util.ArrayList;

import com.google.gson.\*;

import data.FuelType;

import data.SortByEnginePower;

import data.Vehicle;

import data.VehicleType;

import java.io.FileOutputStream;

import java.io.InputStreamReader;

import java.io.FileInputStream;

import java.io.BufferedReader;

import java.io.InputStream;

import java.util.Collections;

import java.nio.file.Files;

import java.nio.file.attribute.BasicFileAttributes;

import java.nio.file.Paths;

import com.google.gson.reflect.TypeToken;

/\*\*

\* Класс VehicleCollection

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class VehicleCollection {

/\*\* Поле коллекции объектов класса Vehicle \*/

private LinkedList<Vehicle> vehicles;

/\*\* Поле флага сохранения коллекции в файле \*/

private boolean isSaved = true;

/\*\* Поле имени файла для сохранения коллекции \*/

private String fileName = "";

/\*\* Поле имени временного файла для сохранения коллекции \*/

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

/\*\*

\* Возвращает имя файла для сохранения коллеккции

\* @return имя файла

\*/

public String getFileName() {

return fileName;

}

/\*\*

\* Возвращает имя временного файла для сохранения коллеккции

\* @return имя временного файла

\*/

public String getTmpFileName() {

return tmpFileName;

}

/\*\*

\* Возвращает состояние флага сохранения коллекции в файле

\* @return true, если коллекцкия нуждается в сохранении, false, если не нуждается

\*/

public boolean isSaved () {

return this.isSaved;

}

/\*\*

\* Конструктор класса VehicleCollection

\* @param fileName имя файла для сохранения коллекции

\*/

public VehicleCollection (String fileName) {

vehicles = new LinkedList<>();

this.fileName = fileName;

}

/\*\*

\* Выводит информацию о коллекции

\*/

public void info () {

System.out.println("Тип коллекции: LinkedList");

System.out.println("Объекты коллекции: Vehicle");

System.out.println("Количество элементов: " + vehicles.size());

try {

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

System.out.println("Дата инициализации: " + attr.creationTime());

} catch (Exception e) {

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

}

}

/\*\*

\* Печатает все элементы коллекции

\*/

public void show () {

vehicles.forEach(vehicle -> {

vehicle.print();

});

}

/\*\*

\* Добавляет элемент в коллекцию

\* @param element Объект класса Vehicle

\* @return true, если добавление успешно, false, если нет

\*/

private boolean add\_element (Vehicle element) {

for(int i = 0; i < vehicles.size(); i++) {

if(element.getId() == vehicles.get(i).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;

}

/\*\*

\* Добавляет элемент в коллекцию

\* @param element Объект класса Vehicle

\*/

public void add (Vehicle element) {

if(add\_element(element)) {

this.isSaved = false;

save\_tmp();

}

}

/\*\*

\* Добавляет элемент в коллекцию

\* @param string Json-строка с параметрами нового объекта класса Vehicle

\*/

public void add (String string) {

Vehicle obj = Vehicle.create(string);

if(obj != null) {

add(obj);

}

}

/\*\*

\* Обновляет элемент с указанным идентификатором

\* @param id идентификатор изменяемого объекта

\* @param element Объект класса Vehicle, заменяющий объект коллекции с заданным идентификатором

\* @return true, если обновление успешно, false, если нет

\*/

public boolean update (int id, Vehicle element) {

boolean found = false;

for(int i = 0; i < vehicles.size(); i++) {

if (vehicles.get(i).getId() == id) {

vehicles.get(i).update(element);

found = true;

break;

}

}

if(found) {

this.isSaved = false;

save\_tmp();

}

return found;

}

/\*\*

\* Удаляет элемент с указанным идентификатором

\* @param id идентификатор удаляемого объекта

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

\*/

public boolean remove (int id) {

boolean found = false;

for (int i = 0; i < vehicles.size(); i++) {

if (vehicles.get(i).getId() == id) {

found = true;

vehicles.remove(i);

break;

}

}

if(found) {

this.isSaved = false;

save\_tmp();

}

return found;

}

/\*\*

\* Удаляет все элементы коллекции

\*/

public void clear () {

vehicles.clear();

this.isSaved = false;

save\_tmp();

}

/\*\*

\* Удаляет элемент с указанным индексом

\* @param index индекс удаляемого объекта

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

\*/

public boolean remove\_at (int index) {

if(index < vehicles.size()) {

vehicles.remove(index);

this.isSaved = false;

save\_tmp();

return true;

}

return false;

}

/\*\*

\* Удаляет элементы коллекции с параметром enginePower, меньшим, чем у указанного объекта

\* @param element указанный объект класса Vehicle

\*/

public void remove\_lower (Vehicle element) {

vehicles.removeIf( vehicle -> (vehicle.getEnginePower() < element.getEnginePower()) );

this.isSaved = false;

save\_tmp();

}

/\*\*

\* Сортирует объекты коллекции по значению поля enginePower

\*/

public void sort () {

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

this.isSaved = false;

save\_tmp();

}

/\*\*

\* Печатает все элементы коллекции, имена которых начинаются с заданной подстроки (префикса)

\* @param name префикс

\*/

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

vehicles.forEach(vehicle -> {

if (vehicle.getName().startsWith(name)) {

vehicle.print();

}

});

}

/\*\*

\* Печатает типы топлива, попадающиеся в коллекции ровно один раз

\*/

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

int[] array = new int[FuelType.values().length];

for(int i = 0; i < FuelType.values().length; i++) {

array[i] = 0;

}

for(int i = 0; i < vehicles.size(); i++) {

FuelType fuelType = vehicles.get(i).getFuelType();

if(fuelType != null) {

array[fuelType.ordinal()]++;

}

}

for(int i = 0; i < FuelType.values().length; i++) {

if(array[i] == 1) {

System.out.println(FuelType.values()[i]);

}

}

}

/\*\*

\* Печатает значения поля type всех элементов коллекции в порядке убывания

\*/

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

int n = VehicleType.values().length + 1;

ArrayList<ArrayList<Integer>> array = new ArrayList<ArrayList<Integer>>(n);

for(int i = 0; i < n; i++) {

array.add(new ArrayList<Integer>());

}

for(int i = 0; i < vehicles.size(); i++) {

VehicleType type = vehicles.get(i).getType();

if(type != null) {

array.get(vehicles.get(i).getType().ordinal() + 1).add(i);

} else {

array.get(0).add(i);

}

}

for(int i = array.size() - 1; i >= 0; i--) {

for(int j = 0; j < array.get(i).size(); j++) {

System.out.println(vehicles.get(array.get(i).get(j)).getType());

}

}

}

/\*\*

\* 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 void save () {

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

this.isSaved = true;

}

}

/\*\*

\* Сохраняет коллекцию во временный файл

\*/

public void save\_tmp () {

save\_elements(this.tmpFileName);

}

}

**4. Выводы по работе:**

Я реализовал консольное приложение для управления коллекцией объектов с интерактивным режимом ввода различных команд. Научился работать с коллекцией LinkedList, сортировать её, сериализовать и десериализовывать объекты в формат JSON с помощью пакета gson. Также ознакомился с созданием документации с помощью javadoc.