Отчет

По лабораторной работе №5

Выполнил:
Козин Богдан Алексеевич
группа R3140

Санкт-Петербург,
университет ИТМО

1. Текст задания:

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.PriorityQueue`
- 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 `xml`
- Чтение данных из файла необходимо реализовать с помощью класса `java.io.BufferedInputStream`
- Запись данных в файл необходимо реализовать с помощью класса `java.io.FileWriter`
- Все классы в программе должны быть задокументированы в формате javadoc.
- Программа должна корректно работать с неправильными данными (ошибки пользовательского ввода, отсутствие 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_first` : удалить первый элемент из коллекции
- `remove_greater {element}` : удалить из коллекции все элементы, превышающие заданный
- `history` : вывести последние 7 команд (без их аргументов)

- `filter_greater_than_fuel_type fuelType` : вывести элементы, значение поля fuelType которых больше заданного
- `print_ascending` : вывести элементы коллекции в порядке возрастания
- `print_field_descending_engine_power` : вывести значения поля enginePower всех элементов в порядке убывания

Формат ввода команд:

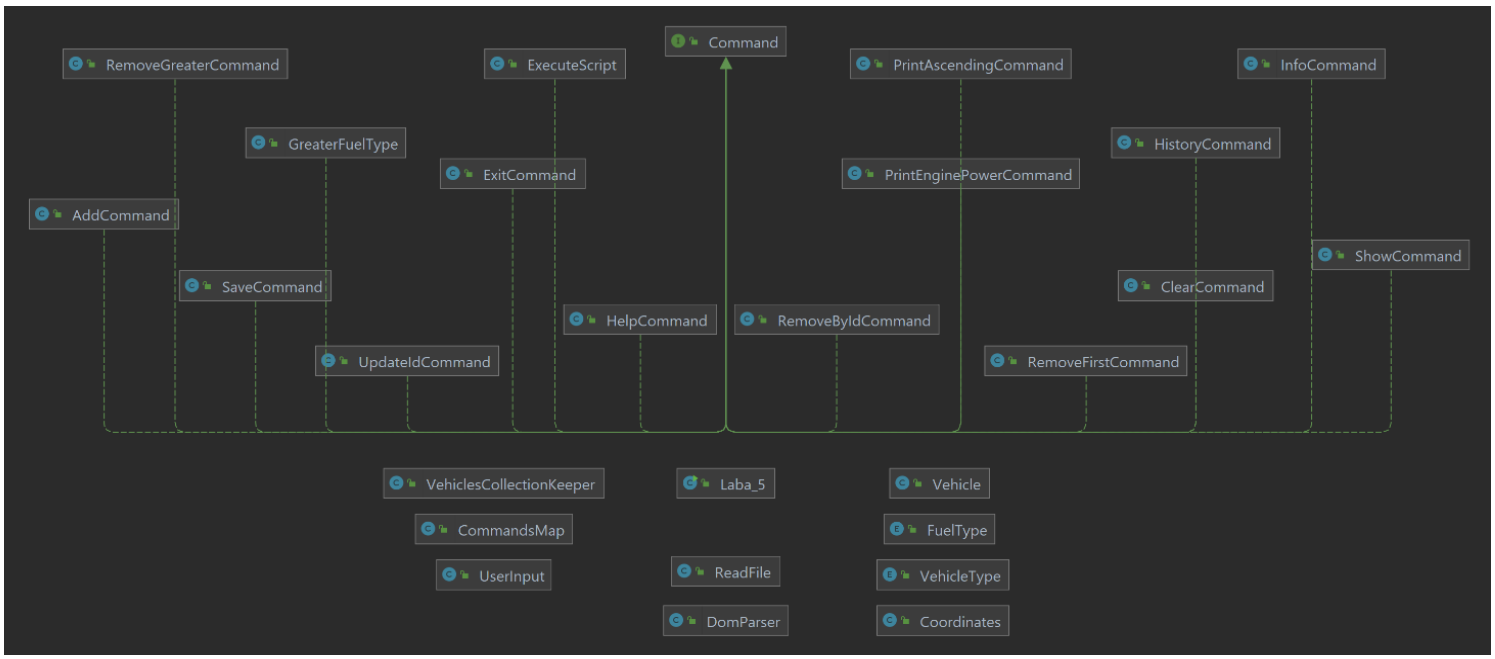
- Все аргументы команды, являющиеся стандартными типами данных (примитивные типы, классы-оболочки, String, классы для хранения дат), должны вводиться в той же строке, что и имя команды.
- Все составные типы данных (объекты классов, хранящиеся в коллекции) должны вводиться по одному полю в строку.
- При 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; //Поле не может быть null, 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.util.Date creationDate; //Поле не может быть null, Значение
этого поля должно генерироваться автоматически

    private Integer enginePower; //Поле может быть null, Значение поля должно
быть больше 0

    private VehicleType type; //Поле может быть null

    private FuelType fuelType; //Поле может быть null

}

public class Coordinates {
```

```
    private Integer x; //Максимальное значение поля: 956, Поле не может быть null
    private int y; //Максимальное значение поля: 368
}
public enum VehicleType {
    CAR,
    HELICOPTER,
    SHIP,
    SPACESHIP;
}
public enum FuelType {
    ALCOHOL,
    MANPOWER,
    NUCLEAR,
    ANTIMATTER;
}
```

Диаграмма классов:



Исходный код программы:

Главный, запускающий класс:

```
import java.io.*;
import java.util.*;

public class Laba_5 {
    public static String[] userInputMas = new String[2];
    public static ArrayList<String> commandsHistory = new
ArrayList<>(7);
    public static CommandsMap commands;

    public static void main(String[] args) throws IOException {
        VehiclesCollectionKeeper vehicles = new
VehiclesCollectionKeeper(ReadFile.getCollectionFromFile());
        commands = new CommandsMap(vehicles);
        UserInput.start();
    }

    public static String getCommand() {
        return userInputMas[0];
    }
    public static String getArgument() throws NullPointerException
{
        if (userInputMas.length > 1) {
            return userInputMas[1];
        } else {
            throw new NullPointerException("Аргумент команды не был
введен");
        }
    }
    public static CommandsMap getCommandsMap() {
        return commands;
    }
}
```

Класс для работы с пользовательским вводом:

```
import java.util.Scanner;

public class UserInput {

    public static void start(){
        while (true) {
            Scanner scan = new Scanner(System.in);
            String userInput = scan.nextLine();
            Laba_5.userInputMas = userInput.split(" ", 3);
            if
    (Laba_5.commands.getKeys().contains(Laba_5.getCommand())) {
                Laba_5.commands.get(Laba_5.getCommand()).execute();
                Laba_5.commandsHistory.add(Laba_5.getCommand());
            } else {
                System.out.println("Было введено значение не из
    списка команд");
            }
            Laba_5.userInputMas = new String[2];
        }
    }
}
```

Интерфейс для всех команд:

```
public interface Command {
    void execute();
}
```

Класс для чтения файла и получения из него коллекции:

```
import java.io.File;
import java.util.Map;
import java.util.PriorityQueue;

public class ReadFile {

    public static PriorityQueue<Vehicle> getCollectionFromFile(){
        //Передаем имя файла из переменной окружения
        File file = new File(getEnvireFileName());  //file1 =
    vehicles1.xml
```

```
        PriorityQueue<Vehicle> vehicles1 =
    DomParser.ParseFile(file);
        return vehicles1;
    }

    public static String getEnvireFileName(){
        Map<String, String> envireVariables = System.getenv();
        String fileName = envireVariables.get("file1");     //file1
    = vehicles1.xml
        return fileName;
    }
}
```

Класс, хранящий Map, где ключ - название команды, значение - объект соответствующей команды.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class CommandsMap {
    private Map<String, Command> commands = new HashMap<>();
    private VehiclesCollectionKeeper collectionKeeper;

    public CommandsMap(VehiclesCollectionKeeper collectionKeeper){
        this.collectionKeeper = collectionKeeper;
        commands.put("help", new HelpCommand());
        commands.put("info", new InfoCommand(collectionKeeper));
        commands.put("show", new ShowCommand(collectionKeeper));
        commands.put("add", new AddCommand(collectionKeeper));
        commands.put("update", new UpdateIdCommand(collectionKeeper));
        commands.put("remove_by_id", new
RemoveByIdCommand(collectionKeeper));
        commands.put("clear", new ClearCommand(collectionKeeper));
        commands.put("save", new SaveCommand(collectionKeeper));
        commands.put("execute_script", new
ExecuteScript(collectionKeeper));
        commands.put("exit", new ExitCommand());
        commands.put("remove_first", new
RemoveFirstCommand(collectionKeeper));
        commands.put("remove_greater", new
RemoveGreaterCommand(collectionKeeper));
        commands.put("history", new HistoryCommand(collectionKeeper));
```

```
        commands.put("filter_greater_than_fuel_type", new
GreaterFuelType(collectionKeeper));
        commands.put("print_ascending", new
PrintAscendingCommand(collectionKeeper));
        commands.put("print_field_descending_engine_power", new
PrintEnginePowerCommand(collectionKeeper));
    }

    public void put(String commandName, Command command){
        this.commands.put(commandName, command);
    }

    public Command get(String key){
        return this.commands.get(key);
    }

    public Set<String> getKeys(){
        return this.commands.keySet();
    }

    public void setElement(String element){
    }
}
```

Класс, хранящий в себе коллекцию и данные о ней:

```
import java.util.Date;
import java.util.PriorityQueue;


public class VehiclesCollectionKeeper {
    private final Date DATE;
    private PriorityQueue<Vehicle> collection;

    public VehiclesCollectionKeeper(PriorityQueue<Vehicle>
collection){
        this.collection = collection;
        this.DATE = new Date();
    }

    public Date getDATE() {
        return DATE;
    }
```

```
    public PriorityQueue<Vehicle> getCollection() {
        return collection;
    }

    public void setCollection(PriorityQueue<Vehicle> collection) {
        this.collection = collection;
    }

    public void add(Vehicle vehicle){
        this.collection.add(vehicle);
    }
}
```

Класс для записи объектов в xml файл. Оказалось, что для этого существуют библиотеки, но я сделал свой :)

```
import org.w3c.dom.*;
import org.xml.sax.SAXException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerFactoryConfigurationError;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;
import java.util.PriorityQueue;

public class DomParser {
    private File file;

    public DomParser(File file) {
        this.file = file;
    }
```

```
    public static PriorityQueue<Vehicle> ParseFile(File file) {
        PriorityQueue<Vehicle> vehicles = null;
        try {
            vehicles = getQueueFromFile(file);

        } catch (ParserConfigurationException ex) {
            ex.printStackTrace(System.out);
        } catch (SAXException ex) {
            ex.printStackTrace(System.out);
        } catch (IOException ex) {
            System.out.println("Имя файла задано неверно");
        }
        return vehicles;
    }

    public static PriorityQueue<Vehicle> getQueueFromFile(File file) throws
IOException, SAXException, ParserConfigurationException,
FileNotFoundException {
        // Строитель документа
        DocumentBuilder documentBuilder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
        // Дерево DOM документа из файла
        //file = new File("VehicleCatalog.xml");
        Document document = documentBuilder.parse(file);

        Node vehicleNode = document.getFirstChild(); //VehicleCatalogue
        //System.out.println(vehicleNode.getNodeName());

        PriorityQueue<Vehicle> vehiclesQueue = new PriorityQueue<>();

        NodeList vehicleChilds = vehicleNode.getChildNodes(); //Vehicle
        for (int i = 0; i < vehicleChilds.getLength(); i++) {
            Vehicle vehicle = new Vehicle();

            if (vehicleChilds.item(i).getNodeType() != Node.ELEMENT_NODE)
{
                continue;
            }
            //System.out.println(vehicleChilds.item(i).getNodeName()); //
Vehicle

            if (!vehicleChilds.item(i).getNodeName().equals("Vehicle")) {
                continue;
            }
```

```
            NodeList vehicleFields = vehicleChilds.item(i).getChildNodes();
            //Поля Id, Name...
            for (int j = 0; j < vehicleFields.getLength(); j++) {
                if (vehicleFields.item(j).getNodeType() !=
Node.ELEMENT_NODE) {
                    continue;
                }

                switch (vehicleFields.item(j).getNodeName()) {
                    case "Id": {

vehicle.setId(Long.parseLong(vehicleFields.item(j).getTextContent()));
                        break;
                    }
                    case "Name": {

vehicle.setName(vehicleFields.item(j).getTextContent());
                        break;
                    }
                    case "Coordinates": {
                        String[] coordinates =
vehicleFields.item(j).getTextContent().split(",");
                        Integer x = Integer.parseInt(coordinates[0]);
                        int y = Integer.parseInt(coordinates[1]);
                        vehicle.setCoordinates(new Coordinates(x, y));
                        break;
                    }
                    case "CreationDate": {

//System.out.println(vehicleFields.item(j).getTextContent());
                        Date date = new
Date(Long.parseLong(vehicleFields.item(j).getTextContent()));
                        vehicle.setCreationDate(date);
                        break;
                    }
                    case "EnginePower": {

vehicle.setEnginePower(Integer.parseInt(vehicleFields.item(j).getTextCont
ent()));
                        break;
                    }
                    case "Type": {

vehicle.setType(VehicleType.valueOf(vehicleFields.item(j).getTextContent(
));
```

```
                    break;
                }
                case "FuelType": {

vehicle.setFuelType(FuelType.valueOf(vehicleFields.item(j).getTextContent
()));
                    break;
                }
            }
        }
        vehiclesQueue.add(vehicle);
    }
    return vehiclesQueue;
}

public static void addNewVehicle(Document document, Vehicle vehicle1,
File file) throws DOMException, TransformerFactoryConfigurationError {
    // Получаем корневой элемент
    Node root = document.getDocumentElement();

    // Сам транспорт <Vehicle>
    Element vehicle = document.createElement("Vehicle");
    // <ID>
    Element id = document.createElement("Id");
    id.setTextContent(vehicle1.getId().toString());
    // <Name>
    Element name = document.createElement("Name");
    name.setTextContent(vehicle1.getName());
    // <Coordinates>
    Element coordinates = document.createElement("Coordinates");
    coordinates.setTextContent(vehicle1.getCoordinates().toString());
    // <CreationDate>
    Element date = document.createElement("CreationDate");
    date.setTextContent(vehicle1.getCreationDate().toString());
    // <EnginePower>
    Element enginePower = document.createElement("EnginePower");
    enginePower.setTextContent(vehicle1.getEnginePower().toString());
    // <Type>
    Element type = document.createElement("Type");
    type.setTextContent(vehicle1.getType().toString());
    // <FuelType>
    Element fuelType = document.createElement("FuelType");
    fuelType.setTextContent(vehicle1.getFuelType().toString());

    // Добавляем внутренние элементы транспорта в элемент <Vehicle>
```

```
        vehicle.appendChild(id);
        vehicle.appendChild(name);
        vehicle.appendChild(coordinates);
        vehicle.appendChild(date);
        vehicle.appendChild(enginePower);
        vehicle.appendChild(type);
        vehicle.appendChild(fuelType);
        // Добавляем транспорт в корневой элемент
        root.appendChild(vehicle);

        // Записываем XML в файл
        writeDocument(document, file);
    }

    // Функция для сохранения DOM в файл
    public static void writeDocument(Document document, File file) {
        try {
            Transformer tr =
TransformerFactory.newInstance().newTransformer();
            DOMSource source = new DOMSource(document);

            //String fileName = "VehicleCatalog.xml";
            FileWriter fw = new FileWriter(file);

            StreamResult result = new StreamResult(fw);
            tr.transform(source, result);
        } catch (TransformerException | IOException e) {
            e.printStackTrace(System.out);
        }
    }
}
```

Пример класса команды:

```
public class ShowCommand implements Command{
    private final VehiclesCollectionKeeper collectionKeeper;

    public ShowCommand(VehiclesCollectionKeeper collection){
        this.collectionKeeper = collection;
    }
    @Override
    public void execute(){
        Vehicle [] veh =
    collectionKeeper.getCollection().toArray(new Vehicle[0]);
        while (!collectionKeeper.getCollection().isEmpty()){
```

```
        System.out.println(collectionKeeper.getCollection().poll());
        }
        for (Vehicle vehicle : veh){
            this.collectionKeeper.add(vehicle);
        }
        if (collectionKeeper.getCollection().isEmpty()){
            System.out.println("Коллекция пуста");
        }
    }
}
```

Полный код здесь: https://github.com/BokoLife/Laba_5

Вывод:
Я изучил виды коллекций, принцип их работы, их отличия. Понял, как происходит сравнение объектов в коллекциях, познакомился с интерфейсом Comparator. Поработал с потоками ввода-вывода, а именно файловыми. Узнал про сериализацию, как происходит запись и чтение объектов в файл. Также пригодились в работе параметризованные типы. Оказалось удобным использовать шаблон проектирования Command, с его помощью можно быстро создавать новые команды и взаимодействовать с ними.