Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный исследовательский университет ИТМО"
Факультет Программной инженерии и компьютерной техники

Лабораторная работа №5 по курсу "Программирование"

Группа: **P3131**
Студент: Друян Эдуард Андреевич
Вариант: 0000

Санкт-Петербург
2022 г.

Задание

Реализовать консольное приложение, которое реализует управление коллекцией объектов в интерактивном режиме. В коллекции необходимо хранить объекты класса Flat, 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.HashMap.
- 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 csv
- Чтение данных из файла необходимо реализовать с помощью класса java.util.Scanner
- 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, отсутсвие прав доступа к файлу и т.п.).

В интерактивном режиме программа должна поддерживать выполнение следующих команд

1. [**help**] : вывести справку по доступным командам.
2. [**info**] : вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.).
3. [**show**] : вывести в стандартный поток вывода все элементы коллекции в строковом представлении.
4. [**insert key {flat}**] : добавить новый элемент с заданным ключом.
5. [**update id {flat}**] : обновить значение элемента коллекции, id которого равен заданному.
6. [**remove_key key**] : удалить элемент из коллекции по его ключу.
7. [**clear**] : очистить коллекцию.
8. [**save**] : сохранить коллекцию в файл.
9. [**execute_script file_path**] : считать и исполнить скрипт из указанного файла. В скрипте содержатся команды в таком же виде, в котором их вводит пользователь в интерактивном режиме (запрещена для вызова из скрипта).
10. [**exit**] : завершить программу (без сохранения в файл).
11. [**remove_lower id**] : удалить из коллекции все элементы, меньшие, чем заданный.

12. [**replace_if_greater key {flat}**] : заменить значение по ключу, если новое значение больше старого.
13. [**replace_if_lower key {flat}**] : заменить значение по ключу, если новое значение меньше старого.
14. [**min_by_id**] : вывести любой объект из коллекции, значение поля id которого является минимальным.
15. [**filter_starts_with_name name**] : вывести элементы, значение поля name которых начинается с заданной подстроки.
16. [**print_descending**] : вывести элементы коллекции 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.

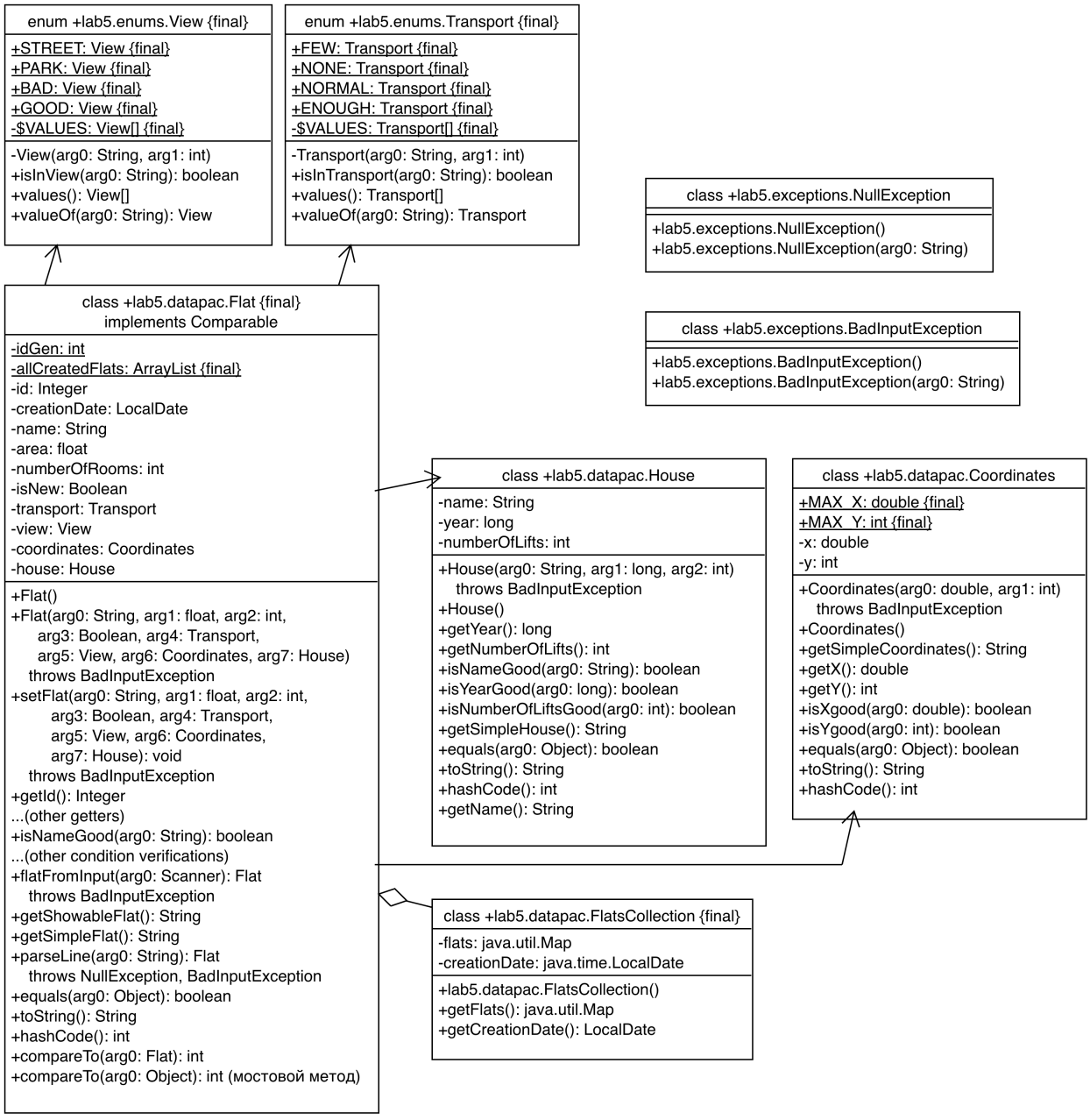
Описание хранимых в коллекции классов

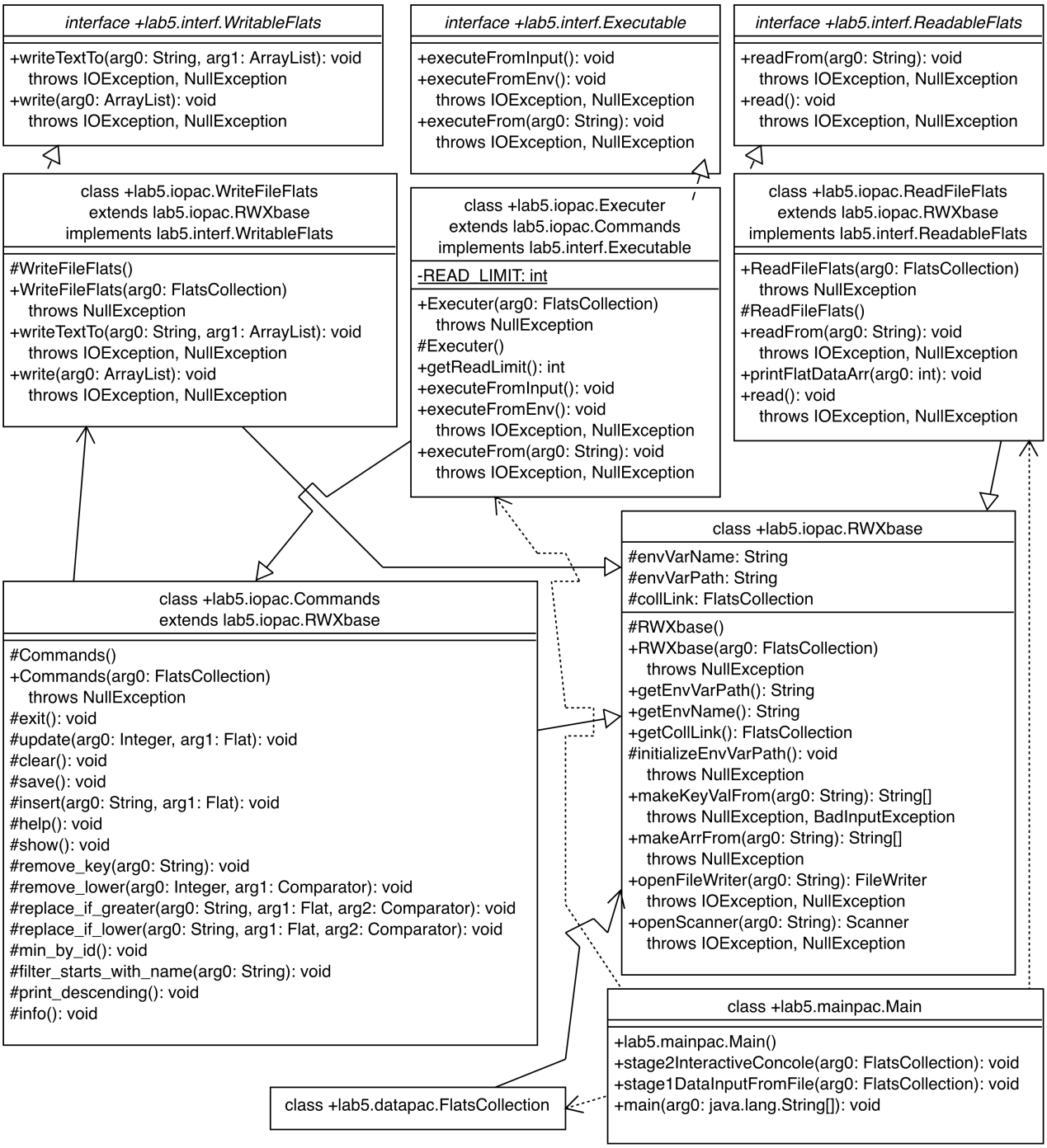
```
1.  public class Flat {
2.      private Integer id; //Поле не может быть null, Значение поля должно быть больше 0, Значение этого поля должно быть уникальным, Значение этого поля должно генерироваться автоматически
3.      private String name; //Поле не может быть null, Строка не может быть пустой
4.      private Coordinates coordinates; //Поле не может быть null
5.      private java.time.LocalDate creationDate; //Поле не может быть null, Значение этого поля должно генерироваться автоматически
6.      private float area; //Значение поля должно быть больше 0
7.      private int numberOfRooms; //Значение поля должно быть больше 0
8.      private Boolean new; //Поле не может быть null
9.      private View view; //Поле не может быть null
10.     private Transport transport; //Поле не может быть null
11.     private House house; //Поле может быть null
12. }
```

```
13. public class Coordinates {
14.     private double x; //Максимальное значение поля: 165
15.     private int y;
16. }
17. public class House {
18.     private String name; //Поле не может быть null
19.     private long year; //Значение поля должно быть больше 0
20.     private int numberOfLifts; //Значение поля должно быть
    больше 0
21. }
22. public enum View { STREET, PARK, BAD, GOOD; }
23. public enum Transport { FEW, NONE, NORMAL, ENOUGH; }
```

Выполнение

Диаграмма классов разработанной программы





Исходный код программы

datapac

Coordinates.java

```
1.   package lab5.datapac;
2.   import java.util.Objects;
3.   import lab5.exceptions.BadInputException;
4.
5.   public class Coordinates {
6.       public static final double MAX_X = 165D;
7.       public static final int MAX_Y = 165;
8.       private double x;
9.       public double getX() { return x; }
10.      private int y;
11.      public int getY() { return y; }
```

```
12.
13.      public static boolean isXgood(double argX) {
14.          return Double.compare(argX, 0) >= 0 &&
15.                 Double.compare(argX, MAX_X) <= 0;
16.      }
17.      public static boolean isYgood(int argY) {
18.          return argY >= 0 && argY <= MAX_Y;
19.      }
20.
21.      public Coordinates() { x = 0D; y = 0; }
22.      public Coordinates(double argX, int argY)
23.      throws BadInputException {
24.          if (isXgood(argX) && isYgood(argY)) {
25.              x = argX;
26.              y = argY;
27.          }
28.          else { throw new BadInputException(
29.              "Ошибка в каком-то параметре Coordinates:" +
30.              "\n argX: " + argX +
31.              "\n argY: " + argY
32.          ); }
33.      }
34.
35.      public String getSimpleCoordinates() {
36.          return Double.valueOf(x).toString() + " " +
37.                 Integer.valueOf(y).toString();
38.      }
39.
40.      @Override
41.      public boolean equals(Object o) {
42.          if (o == null ||
43.              !o.getClass().equals(this.getClass())) {
44.              return false;
45.          }
46.          Coordinates otherCoordinates = (Coordinates) o;
47.          return  ( Double.valueOf(x) ).equals(otherCoordinates.getX()) &&
48.                  y == otherCoordinates.getY();
49.      }
50.      @Override
51.      public String toString() {
52.          return("[x=" + x + ", y=" + y + "]");
53.      }
54.      @Override
55.      public int hashCode() {
56.          return Objects.hash(x, y);
57.      }
58.  }
```

House.java

```
1.   package lab5.datapac;
2.   import java.util.*;
3.   import lab5.exceptions.*;
4.
5.   public class House {
6.       private String name;
7.       public String getName() { return name; }
8.       private long year;
9.       public long getYear() { return year; }
10.      private int numberOfLifts;
11.      public int getNumberOfLifts() { return numberOfLifts; }
12.
13.      public static boolean isNameGood(String argN) {
14.          return argN != null && argN.length() > 0;
15.      }
16.      public static boolean isYearGood(long argY) {
```

```
17.          return argY > 0;
18.      }
19.      public static boolean isNumberOfLiftsGood(int argNOL) {
20.          return argNOL > 0;
21.      }
22.
23.      public House() {
24.          name = "NoNamedHouse";
25.          year = 1900;
26.          numberOfLifts = 1;
27.      }
28.      public House(String n, long y, int nOL)
29.      throws BadInputException {
30.          if (!isNameGood(n) || !isYearGood(y) ||
31.              !isNumberOfLiftsGood(nOL)) {
32.              throw new BadInputException(
33.                  "Ошибка в каком-то параметре House:" +
34.                  "\n argName: " + n +
35.                  "\n argYear: " + y +
36.                  "\n argNOfL: " + nOL
37.              );
38.          }
39.          name = n;
40.          year = y;
41.          numberOfLifts = nOL;
42.      }
43.
44.      public String getSimpleHouse() {
45.          return name + " " +
46.                 Long.valueOf(year).toString() + " " +
47.                 Integer.valueOf(numberOfLifts).toString();
48.      }
49.
50.      @Override
51.      public boolean equals(Object o) {
52.          if (o == null ||
53.              !o.getClass().equals(this.getClass())) {
54.              return false;
55.          }
56.          House otherHouse = (House) o;
57.          return  name.equals(otherHouse.getName()) &&
58.                  year == otherHouse.getYear() &&
59.                  numberOfLifts == otherHouse.getNumberOfLifts();
60.      }
61.      @Override
62.      public String toString() {
63.          return(
64.              "[name=" + name + ", year=" + year +
65.              ", numberOfLifts=" + numberOfLifts + "]"
66.          );
67.      }
68.      @Override
69.      public int hashCode() {
70.          return Objects.hash(name, year, numberOfLifts);
71.      }
72.  }
```

Flat

```
1.   package lab5.datapac;
2.   import java.time.LocalDate;
3.   import java.util.*;
4.   import lab5.enums.*;
5.   import lab5.exceptions.*;
6.   import lab5.iopac.RWXbase;
7.
```

```java
public final class Flat implements Comparable<Flat> {
    private static int idGen = 1;
    public static int getIdGen() { return idGen; }
    private static final ArrayList<Flat> allCreatedFlats;
    // статический блок инициализации
    static { allCreatedFlats = new ArrayList<>(); }
    public ArrayList<Flat> getAllCreatedFlats() {
        return allCreatedFlats;
    }

    private Integer id;
    private LocalDate creationDate;

    private String name;
    private float area;
    private int numberOfRooms;
    private Boolean isNew;
    private Transport transport;
    private View view;
    private Coordinates coordinates;
    private House house;

    // блок инициализации
    { id = idGen++; creationDate = LocalDate.now(); }

    // getters
    public Integer getId() { return id; }
    public LocalDate getCreationDate() { return creationDate; }
    public String getName() { return name; }
    public float getArea() { return area; }
    public int getNumberOfRooms() { return numberOfRooms; }
    public Boolean getIsNew() { return isNew; }
    public Transport getTransport() { return transport; }
    public View getView() { return view; }
    public Coordinates getCoordinates() { return coordinates; }
    public House getHouse() throws NullException {
        if (house == null) { throw new NullException(
            "house == null"
        ); }
        return house;
    }

    public static boolean isNameGood(String argN) {
        return argN != null && argN.length() > 0;
    }
    public static boolean isAreaGood(float argA) {
        return Float.compare(argA, 0F) > 0;
    }
    public static boolean isNumberOfRoomsGood(int argNOR) {
        return argNOR > 0;
    }
    public static boolean isIsNewGood(Boolean argIsNew) {
        return argIsNew != null;
    }
    public static boolean isTransportGood(Transport argT) {
        return argT != null;
    }
    public static boolean isViewGood(View argV) {
        return argV != null;
    }
    public static boolean isCoordinatesGood(Coordinates argC) {
        return argC != null;
    }
    public static boolean isAllGood(String argN, float argA,
                                    int argNOR, Boolean argIsN,
                                    Transport argT, View argV,
```

```
                                        Coordinates argC) {
        return isNameGood(argN) && isAreaGood(argA) &&
               isNumberOfRoomsGood(argNOR) && isIsNewGood(argIsN) &&
               isTransportGood(argT) && isViewGood(argV) &&
               isCoordinatesGood(argC);
    }

    public Flat() {
        name = "NoNamedFlat";
        area = 0.1F;
        numberOfRooms = 1;
        isNew = false;
        transport = Transport.NONE;
        view = View.GOOD;
        coordinates = new Coordinates();
        house = null;
        // для учета созданных квартир
        allCreatedFlats.add(this);
    }
    public Flat(String argN, float argA, int argNOR,
                Boolean argIsN, Transport argT, View argV,
                Coordinates argC, House argH)
    throws BadInputException {
        if (!isAllGood(argN,argA,argNOR,argIsN,argT,argV,argC)) {
            throw new BadInputException(
                "Ошибка в каком-то параметре Flat:" +
                "\n argN: " + argN +
                "\n argA: " + argA +
                "\n argNOR: " + argNOR +
                "\n argIsN: " + argIsN +
                "\n argT: " + argT +
                "\n argV: " + argV +
                "\n argC: " + argC
            );
        }
        name = argN;
        area = argA;
        numberOfRooms = argNOR;
        isNew = argIsN;
        transport = argT;
        view = argV;
        coordinates = argC;
        house = argH;
        // для учета созданных квартир
        allCreatedFlats.add(this);
    }

    public static Flat parseLine(String line)
    throws NullException, BadInputException {
        if (line == null) { throw new NullException(
            "Передана строка line: line == null"
        ); }
        // Количество элементов: 6 + 2 + 3
        String[] sArr = RWXbase.makeArrFrom(line);
        int l = sArr.length;
        if (l != 8 && l != 11) { throw new BadInputException(
            "Количество элементов L в переданной строке \"" +
            line + "\" недопустимо, т.е. (L != 8 && L != 11)."
        ); }

        String argN = null;
        float argA = 0F;
        int argNOR = 0;
        Boolean argIsN = false;
        Transport argT = null;
        View argV = null;
```

```
        Coordinates argC = null;
        House argH = null;

        argN = sArr[0];
        try {
            argA = Float.valueOf(sArr[1]);
            argNOR = Integer.valueOf(sArr[2]);
            argIsN = Boolean.valueOf(sArr[3]);
        } catch (NumberFormatException e) {
            throw new BadInputException(
                "Ошибка в каком-то из элементов строки:" +
                "\n argA: " + sArr[1] +
                "\n argNOR: " + sArr[2] +
                "\n argIsN: " + sArr[3]
            );
        }

        if (!Transport.isInTransport(sArr[4]) ||
            !View.isInView(sArr[5])) {
            throw new BadInputException(
                "Ошибка в каком-то из элементов строки:" +
                "\n argT: " + sArr[4] +
                "\n argV: " + sArr[5]
            );
        }
        argT = Transport.valueOf(sArr[4]);
        argV = View.valueOf(sArr[5]);

        double x = 0D;
        int y = 0;
        try {
            x = Double.valueOf(sArr[6]);
            y = Integer.valueOf(sArr[7]);
        } catch (NumberFormatException e) {
            throw new BadInputException(
                "Ошибка в каком-то из элементов строки:" +
                "\n CoordinatesX: " + sArr[6] +
                "\n CoordinatesY: " + sArr[7]
            );
        }
        argC = new Coordinates(x, y);

        if (l == 11) {
            String houseN = sArr[8];
            long houseY = 0;
            int houseNOL = 0;
            try {
                houseY = Long.valueOf(sArr[9]);
                houseNOL = Integer.valueOf(sArr[10]);
            } catch (NumberFormatException e) {
                throw new BadInputException(
                    "Ошибка в каком-то из элементов строки:" +
                    "\n houseN: " + sArr[8] +
                    "\n houseY: " + sArr[9] +
                    "\n houseNOL: " + sArr[10]
                );
            }
            argH = new House(houseN, houseY, houseNOL);
        }

        Flat f = new Flat(argN, argA, argNOR, argIsN,
                          argT, argV, argC, argH);
        // для учета созданных квартир
        allCreatedFlats.add(f);
        return f;
    }
```

```
206.
207.     public static Flat flatFromInput(Scanner sc)
208.     throws BadInputException {
209.         String argN = null;
210.         float argA = 0F;
211.         int argNOR = 0;
212.         Boolean argIsN = false;
213.         Transport argT = null;
214.         View argV = null;
215.         Coordinates argC = null;
216.         House argH = null;
217.
218.         System.out.println("Ввод данных о квартире.");
219.         String line;
220.
221.         try {
222.             System.out.print(" Имя: ");
223.             line = sc.nextLine();
224.             String[] sArr = RWXbase.makeArrFrom(line);
225.             argN = sArr[0];
226.
227.             System.out.print(" Площадь (float): ");
228.             line = sc.nextLine();
229.             argA = Float.valueOf(line);
230.
231.             System.out.print(" Кол-во комнат: ");
232.             line = sc.nextLine();
233.             argNOR = Integer.valueOf(line);
234.
235.             System.out.print(" Новизна (true, что-либо): ");
236.             line = sc.nextLine();
237.             argIsN = Boolean.valueOf(line);
238.         } catch (NumberFormatException | NullException e) {
239.             throw new BadInputException();
240.         }
241.
242.         System.out.print(" Кол-во транспорта " +
243.             "(FEW, NONE, NORMAL, ENOUGH): ");
244.         line = sc.nextLine();
245.         if (!Transport.isInTransport(line)) {
246.             throw new BadInputException();
247.         }
248.         argT = Transport.valueOf(line);
249.
250.         System.out.print(" Вид из окна " +
251.             "(STREET, PARK, BAD, GOOD): ");
252.         line = sc.nextLine();
253.         if (!View.isInView(line)) {
254.             throw new BadInputException();
255.         }
256.         argV = View.valueOf(line);
257.
258.         System.out.println(" Ввод координат квартиры.");
259.         double x = 0D;
260.         int y = 0;
261.         try {
262.             System.out.print("  x (double): ");
263.             line = sc.nextLine();
264.             x = Double.valueOf(line);
265.
266.             System.out.print("  y (int): ");
267.             line = sc.nextLine();
268.             y = Integer.valueOf(line);
269.         } catch (NumberFormatException e) {
270.             throw new BadInputException();
271.         }
```

```
272.         argC = new Coordinates(x, y);
273.
274.         System.out.println(
275.             " Ввод данных о доме квартиры " +
276.             "(пустая строка, если их нет)."
277.         );
278.         System.out.print("  Имя: ");
279.         line = sc.nextLine();
280.         if (line.length() != 0) {
281.             String houseN = "";
282.             long houseY = 0;
283.             int houseNOL = 0;
284.             try {
285.                 String[] sArr = RWXbase.makeArrFrom(line);
286.                 houseN = sArr[0];
287.                 System.out.print("  Год (long): ");
288.                 line = sc.nextLine();
289.                 houseY = Long.valueOf(line);
290.
291.                 System.out.print("  Кол-во лифтов: ");
292.                 line = sc.nextLine();
293.                 houseNOL = Integer.valueOf(line);
294.             } catch (NumberFormatException | NullException e) {
295.                 throw new BadInputException();
296.             }
297.             argH = new House(houseN, houseY, houseNOL);
298.         }
299.
300.         Flat f = new Flat(argN, argA, argNOR, argIsN,
301.                           argT, argV, argC, argH);
302.         // для учета созданных квартир
303.         allCreatedFlats.add(f);
304.         return f;
305.     }
306.
307.     public void setFlat(String argN, float argA, int argNOR,
308.                         Boolean argIsN, Transport argT, View argV,
309.                         Coordinates argC, House argH)
310.     throws BadInputException {
311.         if (!isAllGood(argN,argA,argNOR,argIsN,argT,argV,argC)) {
312.             throw new BadInputException(
313.                 "Ошибка в каком-то параметре Flat:" +
314.                 "\n argN: " + argN +
315.                 "\n argA: " + argA +
316.                 "\n argNOR: " + argNOR +
317.                 "\n argIsN: " + argIsN +
318.                 "\n argT: " + argT +
319.                 "\n argV: " + argV +
320.                 "\n argC: " + argC
321.             );
322.         }
323.         name = argN;
324.         area = argA;
325.         numberOfRooms = argNOR;
326.         isNew = argIsN;
327.         transport = argT;
328.         view = argV;
329.         coordinates = argC;
330.         house = argH;
331.     }
332.
333.     public String getShowableFlat() {
334.         String s =
335.             "Flat with ID: " + id +
336.             "\n creationDate: " + creationDate +
337.             "\n name: " + name +
```

```
            "\n area: " + area +
            "\n numberOfRooms: " + numberOfRooms +
            "\n isNew: " + isNew +
            "\n transport: " + transport +
            "\n view: " + view +
            "\n coordinates: " + coordinates;
        if (house == null) { s += "\n house: none"; }
        else { s += "\n house: " + house; }
        return s;
    }
    public String getSimpleFlat() {
        String s =
            id + " " +
            creationDate + " " +
            name + " " +
            area + " " +
            numberOfRooms + " " +
            isNew + " " +
            transport + " " +
            view + " " +
            coordinates.getSimpleCoordinates();
        if (house != null) { s += " " + house.getSimpleHouse(); }
        return s;
    }

    @Override
    public boolean equals(Object o) {
        if (o == null ||
            !o.getClass().equals(this.getClass())) {
            return false;
        }
        Flat otherFlat = (Flat) o;
        try {
            return id.equals(otherFlat.getId()) &&
                   creationDate.equals(otherFlat.getCreationDate()) &&
                   name.equals(otherFlat.getName()) &&
                   ( Float.valueOf(area) ).equals(otherFlat.getArea()) &&
                   numberOfRooms == otherFlat.getNumberOfRooms() &&
                   isNew == getIsNew() &&
                   transport.equals(otherFlat.getTransport()) &&
                   view.equals(otherFlat.getView()) &&
                   coordinates.equals(otherFlat.getCoordinates()) &&
                   house.equals(otherFlat.getHouse());
        } catch (NullException e) { return false; }
    }
    @Override
    public String toString() {
        if (house == null) {
            return(
                "[id=" + id +
                ", creationDate=" + creationDate +
                ", name=" + name +
                ", area=" + area +
                ", numberOfRooms=" + numberOfRooms +
                ", isNew=" + isNew +
                ", transport=" + transport +
                ", view=" + view +
                ", coordinates=" + coordinates +
                ", house=none]"
            );
        }
        else {
            return(
                "[id=" + id +
                ", creationDate=" + creationDate +
                ", name=" + name +
```

```
                ", area=" + area +
                ", numberOfRooms=" + numberOfRooms +
                ", isNew=" + isNew +
                ", transport=" + transport +
                ", view=" + view +
                ", coordinates=" + coordinates +
                ", house=" + house + "]"
            );
        }
    }
    @Override
    public int compareTo(Flat o) {
        return id.compareTo(o.id);
    }
    @Override
    public int hashCode() {
        return Objects.hash(id,creationDate,name,area,
                            numberOfRooms,isNew,transport,
                            view,coordinates,house);
    }
}
```

FlatsCollection.java

```
package lab5.datapac;
import java.time.LocalDate;
import java.util.*;

public final class FlatsCollection {
    private Map<String, Flat> flats;
    private LocalDate creationDate;

    public FlatsCollection() {
        flats = new HashMap<>();
        creationDate = LocalDate.now();
    }

    public LocalDate getCreationDate() { return creationDate; }
    public Map<String, Flat> getFlats() { return flats; }
}
```

enums

Transport.java

```
package lab5.enums;

public enum Transport {
    FEW, NONE, NORMAL, ENOUGH;
    public static boolean isInTransport(String s) {
        if (s == null) { return false; }
        Transport[] tArr = Transport.values();
        for (Transport elem : tArr) {
            if (elem.toString().equals(s)) { return true; }
        }
        return false;
    }
}
```

View.java

```
package lab5.enums;

public enum View {
    STREET, PARK, BAD, GOOD;
```

```
5.      public static boolean isInView(String s) {
6.          if (s == null) { return false; }
7.          View[] vArr = View.values();
8.          for (View elem : vArr) {
9.              if (elem.toString().equals(s)) { return true; }
10.         }
11.         return false;
12.     }
13. }
```

exceptions

BadInputException.java

```
1.  package lab5.exceptions;
2.
3.  public class BadInputException extends Exception {
4.      public BadInputException() {
5.          super("::::ПЛОХИЕ ВХОДНЫЕ ДАННЫЕ::::");
6.      }
7.      public BadInputException(String message) {
8.          super(message);
9.      }
10. }
```

NullException.java

```
1.  package lab5.exceptions;
2.
3.  public class NullException extends Exception {
4.      public NullException() {
5.          super("::::ОБРАЩЕНИЕ К null::::");
6.      }
7.      public NullException(String message) {
8.          super(message);
9.      }
10. }
```

interf

Executable.java

```
1.  package lab5.interf;
2.  import java.io.*;
3.  import lab5.exceptions.*;
4.
5.  public interface Executable {
6.      public void executeFromInput();
7.      public void executeFromEnv()
8.      throws IOException, NullException;
9.      public void executeFrom(String filePath)
10.     throws IOException, NullException;
11. }
```

ReadableFlats.java

```
1.  package lab5.interf;
2.  import java.io.IOException;
3.  import lab5.exceptions.NullException;
4.
5.  public interface ReadableFlats {
6.      void readFrom(String filePath) throws IOException, NullException;
7.      void read() throws IOException, NullException;
8.  }
```

WritableFlats.java

```
1.  package lab5.interf;
```

```
2.  import java.io.*;
3.  import java.util.ArrayList;
4.  import lab5.exceptions.NullException;
5.
6.  public interface WritableFlats {
7.      void writeTextTo(String filePath, ArrayList<String> text)
8.      throws IOException, NullException;
9.      void write(ArrayList<String> text)
10.     throws IOException, NullException;
11. }
```

iopac

Commands.java

```
1.  package lab5.iopac;
2.  import java.io.*;
3.  import java.util.*;
4.  import lab5.datapac.Flat;
5.  import lab5.datapac.FlatsCollection;
6.  import lab5.exceptions.*;
7.
8.  public class Commands extends RWXbase {
9.      protected Commands() {}
10.     public Commands(FlatsCollection toLink)
11.     throws NullException {
12.         super(toLink);
13.         envVarName = "EX_COMMANDS";
14.         initializeEnvVarPath();
15.     }
16.
17.     protected void help() {
18.         System.out.println(
19.             "Справка по доступным командам:\n" +
20.             " [help] : вывести справку по доступным командам.\n" +
21.             " [info] : вывести в стандартный поток вывода информацию о
    коллекции (тип, дата инициализации, количество элементов и т.д.).\n" +
22.             " [show] : вывести в стандартный поток вывода все элементы
    коллекции в строковом представлении.\n" +
23.             " [insert key {flat}] : добавить новый элемент с заданным
    ключом.\n" +
24.             " [update id {flat}] : обновить значение элемента коллекции, id
    которого равен заданному.\n" +
25.             " [remove_key key] : удалить элемент из коллекции по его ключу.
    \n" +
26.             " [clear] : очистить коллекцию.\n" +
27.             " [save] : сохранить коллекцию в файл.\n" +
28.             " [execute_script file_path] : считать и исполнить скрипт из
    указанного файла. В скрипте содержатся команды в таком же виде, в котором
    их вводит пользователь в интерактивном режиме (запрещена для вызова из
    скрипта).\n" +
29.             " [exit] : завершить программу (без сохранения в файл).\n" +
30.             " [remove_lower id] : удалить из коллекции все элементы,
    меньшие, чем заданный.\n" +
31.             " [replace_if_greater key {flat}] : заменить значение по ключу,
    если новое значение больше старого.\n" +
32.             " [replace_if_lower key {flat}] : заменить значение по ключу,
    если новое значение меньше старого.\n" +
33.             " [min_by_id] : вывести любой объект из коллекции, значение
    поля id которого является минимальным.\n" +
34.             " [filter_starts_with_name name] : вывести элементы, значение
    поля name которых начинается с заданной подстроки.\n" +
35.             " [print_descending] : вывести элементы коллекции в порядке
    убывания."
36.         );
37.         System.out.println(
38.             "Ограничения ввода данных квартиры:\n" +
```

```
            " Имя: не пустое, не более 1 слова\n" +
            " Площадь (float): больше 0\n" +
            " Кол-во комнат: больше 0\n" +
            " Ограничения ввода координат квартиры:\n" +
            "  x (double): 0 <= x <= 165\n" +
            "  y (int): 0 <= y <= 165\n" +
            " Ограничения ввода данных о доме квартиры:\n" +
            "  Имя: не пустое, не более 1 слова\n" +
            "  Год (long): больше 0\n" +
            "  Кол-во лифтов: больше 0\n" +
            "Ключ: не пустой, не более 1 слова"
        );
    }
    protected void info() {
        Map<String, Flat> flats = collLink.getFlats();
        System.out.println(
            "Информация о коллекции:" +
            "\n Тип: " + flats.getClass() +
            "\n Дата создания: " + collLink.getCreationDate() +
            "\n Количество элементов: " + flats.size()
        );
    }
    protected void show() {
        Map<String, Flat> flats = collLink.getFlats();
        if (flats.isEmpty()) {
            System.out.println("Коллекция пустая.");
            return;
        }
        System.out.println("Все элементы коллекции (ключ: зачение):");
        flats.forEach((key, flat) -> {
            System.out.println(
                key + ": " + flat.getShowableFlat()
            );
        });
    }
    protected void insert(String key, Flat flat) {
        Map<String, Flat> flats = collLink.getFlats();
        if (flats.containsKey(key)) {
            System.out.println("Ключ \"" + key + "\" уже добавлен.");
            return;
        }
        flats.put(key, flat);
        System.out.println("Квартира по ключу \"" +
            key + "\" успешно добавлена.");
    }
    protected void update(Integer id, Flat flat) {
        Map<String, Flat> flats = collLink.getFlats();
        if (flats.isEmpty()) {
            System.out.println(
                "Обновить по ID \"" + id + "\" " +
                "не получилось, т.к. коллекция пустая."
            );
            return;
        }
        Collection<Flat> flatsVals = flats.values();
        Iterator<Flat> iter = flatsVals.iterator();
        while (iter.hasNext()) {
            Flat iterFlat = iter.next();
            if (iterFlat.getId().equals(id)) {
                try {
                    iterFlat.setFlat(flat.getName(), flat.getArea(),
                                flat.getNumberOfRooms(), flat.getIsNew(),
                                flat.getTransport(), flat.getView(),
                                flat.getCoordinates(), flat.getHouse());
                } catch (BadInputException | NullException e) {
                    e.printStackTrace();
```

```
                }
                System.out.println("Изменения в квартиру " +
                    "по ID \"" + id + "\" внесены.");
                return;
            }
        }
        System.out.println("ID \"" + id + "\" не найден в коллекции.");
    }
    protected void remove_key(String key) {
        Map<String, Flat> flats = collLink.getFlats();
        if (flats.containsKey(key)) {
            flats.remove(key);
            System.out.println("Квартира по ключу \"" +
                key + "\" успешно удалена.");
            return;
        }
        System.out.println("Нет ключа \"" +
            key + "\" в коллекции.");
    }
    protected void clear() {
        Map<String, Flat> flats = collLink.getFlats();
        flats.clear();
        System.out.println("Коллекция очищена.");
    }
    protected void save() {
        Map<String, Flat> flats = collLink.getFlats();
        if (flats.isEmpty()) {
            System.out.println(
                "Сохранить коллекцию в файл не удасться," +
                " т.к. коллекция пустая."
            );
            return;
        }
        WriteFileFlats writeFileFlats = null;
        try {
            writeFileFlats = new WriteFileFlats(collLink);
        } catch (NullException e) {
            e.printStackTrace();
        }

        ArrayList<String> text = new ArrayList<>();
        flats.forEach((k, v) -> {
            text.add(k + " " + v.getSimpleFlat());
        });

        try { writeFileFlats.write(text); }
        catch (IOException | NullException e) {
            e.printStackTrace();
        }
        System.out.println(
            "Данные коллекции успешно записаны в файл: \"" +
            writeFileFlats.getEnvVarPath() + "\"."
        );
    }
    protected void exit() {
        System.out.println("Завершение выполнения программы...");
    }
    protected void remove_lower(Integer id,
                                Comparator<Flat> comparator) {
        Map<String, Flat> flats = collLink.getFlats();
        if (flats.isEmpty()) {
            System.out.println("Коллекция пустая.");
            return;
        }
        Set<Map.Entry<String, Flat>> flatsSet = flats.entrySet();
        Iterator<Map.Entry<String, Flat>> iter = flatsSet.iterator();
```

```
171.         Flat flat = null;
172.         while (iter.hasNext()) {
173.             Flat iterFlat = iter.next().getValue();
174.             if (id.equals(iterFlat.getId())) { flat = iterFlat; }
175.         }
176.         if (flat == null) {
177.             System.out.println("В коллекции нет элемента " +
178.                 "с ID \"" + id + "\".");
179.             return;
180.         }
181.         iter = flatsSet.iterator();
182.         while (iter.hasNext()) {
183.             Flat iterFlat = iter.next().getValue();
184.             if (comparator.compare(iterFlat, flat) < 0) { iter.remove(); }
185.         }
186.         System.out.println("Удаление элементов, меньших" +
187.             " заданного по ID \"" + id + "\", прошло успешно.");
188.     }
189.     protected void replace_if_greater(String key, Flat flat,
190.                                       Comparator<Flat> comparator) {
191.         Map<String, Flat> flats = collLink.getFlats();
192.         Flat keyFlat = flats.get(key);
193.         if (keyFlat == null) {
194.             System.out.println("Ключ \"" + key + "\" не найден.");
195.             return;
196.         }
197.         if (comparator.compare(flat, keyFlat) > 0) {
198.             flats.put(key, flat);
199.             System.out.println("БОльшие данные " +
200.                 "заменены по ключу \"" + key + "\".");
201.             return;
202.         }
203.         System.out.println("Данные НЕ заменены по ключу \"" +
204.             key + "\", т.к. не оказались бОльшими.");
205.     }
206.     protected void replace_if_lower(String key, Flat flat,
207.                                       Comparator<Flat> comparator) {
208.         Map<String, Flat> flats = collLink.getFlats();
209.         Flat keyFlat = flats.get(key);
210.         if (keyFlat == null) {
211.             System.out.println("Ключ \"" + key + "\" не найден.");
212.             return;
213.         }
214.         if (comparator.compare(flat, keyFlat) < 0) {
215.             flats.put(key, flat);
216.             System.out.println("Меньшие данные " +
217.                 "заменены по ключу \"" + key + "\".");
218.             return;
219.         }
220.         System.out.println("Данные НЕ заменены по ключу \"" +
221.             key + "\", т.к. не оказались меньшими.");
222.     }
223.     protected void min_by_id() {
224.         Map<String, Flat> flats = collLink.getFlats();
225.         Iterator<Map.Entry<String, Flat>> iter =
    flats.entrySet().iterator();
226.         Flat iterFlat;
227.         if (iter.hasNext()) { iterFlat = iter.next().getValue(); }
228.         else { System.out.println("Пустая коллекция."); return; }
229.         while (iter.hasNext()) {
230.             Flat v = iter.next().getValue();
231.             if (iterFlat.getId().compareTo(v.getId()) > 0) { iterFlat =
    v; }
232.         }
233.         System.out.println("Квартира с минимальным ID: " +
234.             iterFlat.getShowableFlat());
```

```
235.     }
236.     protected void filter_starts_with_name(String subName) {
237.         Map<String, Flat> flats = collLink.getFlats();
238.         Iterator<Map.Entry<String, Flat>> iter =
    flats.entrySet().iterator();
239.         ArrayList<Flat> subNamedFlats = new ArrayList<>();
240.         while (iter.hasNext()) {
241.             Flat iterFlat = iter.next().getValue();
242.             int lSub = subName.length();
243.             int lIter = iterFlat.getName().length();
244.             if (lIter >= lSub) {
245.                 String subV = iterFlat.getName().substring(0, lSub);
246.                 if (subV.equals(subName)) { subNamedFlats.add(iterFlat); }
247.             }
248.         }
249.         if (subNamedFlats.isEmpty()) {
250.             System.out.println("Нет квартир в коллекции, " +
251.                 "имена которых начинаются на \"" + subName + "\".");
252.             return;
253.         }
254.         System.out.println("Квартиры в коллекции, " +
255.             "имена которых начинаются на \"" + subName + "\":");
256.         for (Flat flat : subNamedFlats) {
257.             System.out.println(flat.getShowableFlat());
258.         }
259.     }
260.     protected void print_descending() {
261.         NavigableMap<String, Flat> flats =
262.         (new TreeMap<String, Flat>(collLink.getFlats())).descendingMap();
263.         if (flats.size() == 0) {
264.             System.out.println(
265.                 "Нельзя вывести элементы коллекции " +
266.                 "в порядке убывания, т.к. коллекция пустая."
267.             );
268.             return;
269.         }
270.         System.out.println("Коллекция в обратном порядке:");
271.         flats.forEach((k, v) -> {
272.             System.out.println(k + ": " + v.getShowableFlat());
273.         });
274.     }
275. }
```

Executer.java

```
1.  package lab5.iopac;
2.  import java.io.*;
3.  import java.util.*;
4.  import lab5.datapac.*;
5.  import lab5.exceptions.*;
6.  import lab5.interf.Executable;
7.
8.  public class Executer extends Commands implements Executable {
9.      private static int READ_LIMIT = 65535;
10.     public static int getReadLimit() { return READ_LIMIT; }
11.
12.     protected Executer() {}
13.     public Executer(FlatsCollection toLink)
14.     throws NullException {
15.         super(toLink);
16.     }
17.
18.     @Override
19.     public void executeFromInput() {
20.         Scanner sc = new Scanner(System.in);
21.         int i;
```

```
22.          for (i = 0; i < READ_LIMIT; ++i) {
23.              System.out.print("Команда: ");
24.              String line = sc.nextLine();
25.              String[] lineArr;
26.              try {
27.                  lineArr = RWXbase.makeArrFrom(line);
28.                  String key, subName;
29.                  Flat flat;
30.                  Integer id;
31.                  Map<String, Flat> flats = getCollLink().getFlats();
32.                  switch (lineArr[0]) {
33.                      case "exit": exit(); sc.close(); return;
34.                      case "help": help(); break;
35.                      case "info": info(); break;
36.                      case "show": show(); break;
37.                      case "insert":
38.                          key = lineArr[1];
39.                          if (flats.containsKey(key)) {
40.                              System.out.println("Ключ \"" + key +
41.                                  "\" уже добавлен.");
42.                              continue;
43.                          }
44.                          flat = Flat.flatFromInput(sc);
45.                          insert(key, flat);
46.                          break;
47.                      case "update":
48.                          id = Integer.valueOf(lineArr[1]);
49.                          flat = Flat.flatFromInput(sc);
50.                          update(id, flat);
51.                          break;
52.                      case "remove_key":
53.                          key = lineArr[1];
54.                          remove_key(key);
55.                          break;
56.                      case "clear": clear(); break;
57.                      case "save": save(); break;
58.                      case "execute_script":
59.                          if (lineArr.length == 1) { executeFromEnv(); }
60.                          else { executeFrom(lineArr[1]); }
61.                          break;
62.                      case "remove_lower":
63.                          id = Integer.valueOf(lineArr[1]);
64.                          remove_lower(id, (a, b) -> {
65.                              return Float.compare(a.getArea(), b.getArea());
66.                          });
67.                          break;
68.                      case "replace_if_greater":
69.                          key = lineArr[1];
70.                          if (!flats.containsKey(key)) {
71.                              System.out.println("Ключа \"" +
72.                                  key + "\" нет в коллекции.");
73.                              continue;
74.                          }
75.                          flat = Flat.flatFromInput(sc);
76.                          replace_if_greater(key, flat, (a, b) -> {
77.                              return Float.compare(a.getArea(), b.getArea());
78.                          });
79.                          break;
80.                      case "replace_if_lower":
81.                          key = lineArr[1];
82.                          if (!flats.containsKey(key)) {
83.                              System.out.println("Ключа \"" +
84.                                  key + "\" нет в коллекции.");
85.                              continue;
86.                          }
87.                          flat = Flat.flatFromInput(sc);
```

```
                        replace_if_lower(key, flat, (a, b) -> {
                            return Float.compare(a.getArea(), b.getArea());
                        });
                        break;
                    case "min_by_id": min_by_id(); break;
                    case "filter_starts_with_name":
                        subName = lineArr[1];
                        filter_starts_with_name(subName);
                        break;
                    case "print_descending":
                        print_descending();
                        break;

                    default: throw new BadInputException(
                        "Нет команды \"" + lineArr[0] + "\"."
                    );
                }
            } catch (BadInputException |
                     NullException |
                     IOException e) {
                System.out.println(e.getMessage());
                System.out.println("\"help\" для справки.");
                continue;
            } catch (RuntimeException e) {
                System.out.println(
                    "Плохой ввод. \"help\" для справки."
                );
                continue;
            }
        }
        if (i >= READ_LIMIT) {
            System.out.println("Достигнут предел ввода.");
        }
        sc.close();
    }
    @Override
    public void executeFromEnv()
    throws IOException, NullException {
        executeFrom(envVarPath);
    }
    @Override
    public void executeFrom(String filePath)
    throws IOException, NullException {
        // opening the file
        Scanner sc = openScanner(filePath);

        int i = 0;
        for (i = 0; i < READ_LIMIT && sc.hasNext(); ++i) {
            String line = sc.nextLine();
            String[] lineArr;
            try {
                lineArr = RWXbase.makeArrFrom(line);
                String key, subName;
                Flat flat;
                Integer id;
                Map<String, Flat> flats = getCollLink().getFlats();
                switch (lineArr[0]) {
                    case "exit": sc.close(); return;
                    case "help": help(); break;
                    case "info": info(); break;
                    case "show": show(); break;
                    case "insert":
                        key = lineArr[1];
                        if (flats.containsKey(key)) {
                            System.out.println("Ключ \"" + key +
                                "\" уже добавлен.");
```

```
                            continue;
                        }
                        line = sc.nextLine();
                        ++i;
                        flat = Flat.parseLine(line);
                        insert(key, flat);
                        break;
                    case "update":
                        id = Integer.valueOf(lineArr[1]);
                        line = sc.nextLine();
                        ++i;
                        flat = Flat.parseLine(line);
                        update(id, flat);
                        break;
                    case "remove_key":
                        key = lineArr[1];
                        remove_key(key);
                        break;
                    case "clear": clear(); break;
                    case "save": save(); break;
                    case "execute_script":
                        throw new BadInputException(
                            "Во избежание бесконечной рекурсии, " +
                            "вызов \"execute_script\" из скрипта запрещен."
                        );
                    case "remove_lower":
                        id = Integer.valueOf(lineArr[1]);
                        remove_lower(id, (a, b) -> {
                            return Float.compare(a.getArea(), b.getArea());
                        });
                        break;
                    case "replace_if_greater":
                        key = lineArr[1];
                        if (!flats.containsKey(key)) {
                            System.out.println("Ключа \"" +
                                key + "\" нет в коллекции.");
                            continue;
                        }
                        line = sc.nextLine();
                        ++i;
                        flat = Flat.parseLine(line);
                        replace_if_greater(key, flat, (a, b) -> {
                            return Float.compare(a.getArea(), b.getArea());
                        });
                        break;
                    case "replace_if_lower":
                        key = lineArr[1];
                        if (!flats.containsKey(key)) {
                            System.out.println("Ключа \"" +
                                key + "\" нет в коллекции.");
                            continue;
                        }
                        line = sc.nextLine();
                        ++i;
                        flat = Flat.parseLine(line);
                        replace_if_lower(key, flat, (a, b) -> {
                            return Float.compare(a.getArea(), b.getArea());
                        });
                        break;
                    case "min_by_id": min_by_id(); break;
                    case "filter_starts_with_name":
                        subName = lineArr[1];
                        filter_starts_with_name(subName);
                        break;
                    case "print_descending":
                        print_descending();
```

```
                        break;

                    default: throw new BadInputException(
                        "Нет команды \"" + lineArr[0] + "\"."
                    );
                }
            } catch (BadInputException |
                     NullException e) {
                e.printStackTrace();
                System.out.println(
                    "\tфайл: " + filePath + ";\n" +
                    "\tстрока: " + (i + 1) + ".");
                continue;
            } catch (RuntimeException e) {
                System.out.println("Плохой ввод.");
                e.printStackTrace();
                System.out.println(
                    "\tфайл: " + filePath + ";\n" +
                    "\tстрока: " + (i + 1) + ".");
                continue;
            }
        }
        if (i >= READ_LIMIT) {
            System.out.println("Достигнут предел ввода.");
        }
        sc.close();
    }
}
```

ReadFileFlats.java

```
package lab5.iopac;
import java.io.*;
import java.util.*;
import lab5.datapac.*;
import lab5.enums.*;
import lab5.exceptions.*;
import lab5.interf.ReadableFlats;

public class ReadFileFlats
extends RWXbase
implements ReadableFlats {
    public static void printFlatDataArr(int amount) {
        for (int i = 1; i <= amount; ++i) {
            Transport[] tArr = Transport.values();
            View[] vArr = View.values();
            System.out.println(
                "key" + i + " " +
                "FlatName" + i + " " +
                ((20 + i % 10) + 0.625) + " " +
                (2 + i % 4) + " " +
                (i % 2 == 0 ? true : false) + " " +
                tArr[i % tArr.length] + " " +
                vArr[i % vArr.length] + " " +
                ((i % 164) + 0.625) + " " + (i % 164) + " " +
                "HouseName" + i + " " + (1900 + i) + " " + (2 + i % 2)
            );
        }
    }

    protected ReadFileFlats() {}
    public ReadFileFlats(FlatsCollection toLink)
    throws NullException {
        super(toLink);
        envVarName = "READ_FLATS";
        initializeEnvVarPath();
```

```
36.      }
37.
38.      @Override
39.      public void readFrom(String filePath)
40.      throws IOException, NullException {
41.          Scanner sc = openScanner(filePath);
42.          ArrayList<String> text = new ArrayList<>();
43.          while (sc.hasNextLine()) { text.add(sc.nextLine()); }
44.          int counter = 1;
45.          for (String line : text) {
46.              if (line.length() == 0) { ++counter; continue; }
47.              try {
48.                  String[] keyVal = makeKeyValFrom(line);
49.                  Flat flat = Flat.parseLine(keyVal[1]);
50.                  collLink.getFlats().put(keyVal[0], flat);
51.              } catch (BadInputException e) {
52.                  e.printStackTrace();
53.                  System.out.println(
54.                      // e.getMessage() +
55.                      "\tфайл: " + filePath +
56.                      "\n\tстрока: " + counter
57.                  );
58.              }
59.              ++counter;
60.          }
61.          sc.close();
62.      }
63.      @Override
64.      public void read() throws IOException, NullException {
65.          readFrom(envVarPath);
66.      }
67.  }
```

RWXbase.java

```
1.   package lab5.iopac;
2.   import java.io.*;
3.   import java.util.*;
4.   import lab5.datapac.*;
5.   import lab5.exceptions.*;
6.
7.   public class RWXbase {
8.       protected String envVarName = null;
9.       public String getEnvName() { return envVarName; }
10.      protected String envVarPath = null;
11.      public String getEnvVarPath() { return envVarPath; }
12.      protected FlatsCollection collLink;
13.      public FlatsCollection getCollLink() { return collLink; }
14.      protected void initializeEnvVarPath()
15.      throws NullException {
16.          if (envVarName == null) {
17.              throw new NullException("envVarName == null");
18.          }
19.          Map<String, String> envVars = System.getenv();
20.          Set<Map.Entry<String, String>> envSet = envVars.entrySet();
21.          for (Map.Entry<String, String> entry : envSet) {
22.              if (entry.getKey().equals(envVarName)) {
23.                  envVarPath = entry.getValue();
24.                  break;
25.              }
26.          }
27.          if (envVarPath == null) {
28.              throw new NullException("envVarPath == null");
29.          }
30.      }
31.
```

```
32.      protected RWXbase() {}
33.      public RWXbase(FlatsCollection toLink)
34.      throws NullException {
35.          if (toLink == null) { throw new NullException(
36.          "FlatsCollection toLink == null"
37.          ); }
38.          collLink = toLink;
39.      }
40.
41.      public static String[] makeKeyValFrom(String s)
42.      throws NullException, BadInputException {
43.          if (s == null) { throw new NullException(
44.              "Переданная строка s: s == null"
45.          ); }
46.          s = s.trim();
47.          int i = 0;
48.          while (i < s.length() &&
49.                 !Character.isWhitespace(s.charAt(i))) { ++i; }
50.          if (i == s.length()) { throw new BadInputException(
51.              "В переданной строке \"" + s +
52.              "\" нет пробельных символов"
53.          ); }
54.          String[] sArr = new String[] {
55.              s.substring(0, i),
56.              s.substring(i)
57.          };
58.          return sArr;
59.      }
60.      public static String[] makeArrFrom(String s)
61.      throws NullException {
62.          if (s == null) { throw new NullException(
63.              "Переданная строка s: s == null"
64.          ); }
65.          s = s.trim();
66.          if (s.length() == 0) { return new String[0]; }
67.          return s.split("[\\h]+");
68.      }
69.
70.      public static FileWriter openFileWriter(String filePath)
71.      throws IOException, NullException {
72.          if (filePath == null) { throw new NullException(
73.              "Переданная строка filePath: filePath == null"
74.          ); }
75.          File file = new File(filePath);
76.          if (!file.canRead()) { throw new IOException(
77.              "\"" + filePath + "\" не может быть прочитан"
78.          ); }
79.          return new FileWriter(file, false);
80.      }
81.      public static Scanner openScanner(String filePath)
82.      throws IOException, NullException {
83.          if (filePath == null) { throw new NullException(
84.              "Переданная строка filePath: filePath == null"
85.          ); }
86.          File file = new File(filePath);
87.          if (!file.canRead()) { throw new IOException(
88.              "\"" + filePath + "\" не может быть прочитан"
89.          ); }
90.          return new Scanner(file);
91.      }
92.  }
```

WriteFileFlats.java

```
93.  package lab5.iopac;
94.  import java.io.*;
```

```java
import java.util.*;
import lab5.datapac.*;
import lab5.exceptions.*;
import lab5.interf.WritableFlats;

public class WriteFileFlats
extends RWXbase
implements WritableFlats {
    protected WriteFileFlats() {}
    public WriteFileFlats(FlatsCollection toLink)
    throws NullException {
        super(toLink);
        envVarName = "WRITE_FLATS";
        initializeEnvVarPath();
    }

    @Override
    public void writeTextTo(String filePath, ArrayList<String> text)
    throws IOException, NullException {
        FileWriter fw = openFileWriter(filePath);
        for (String line : text) { fw.write(line + "\n"); }
        fw.close();
    }
    @Override
    public void write(ArrayList<String> text)
    throws IOException, NullException {
        writeTextTo(envVarPath, text);
    }
}
```

mainpac

Main.java

```java
package lab5.mainpac;
import java.io.IOException;
import lab5.datapac.*;
import lab5.exceptions.NullException;
import lab5.iopac.*;

public class Main {
    public static void main(String[] args) {
        // ReadFileFlats.printFlatDataArr(16);
        FlatsCollection coll = new FlatsCollection();
        stage1DataInputFromFile(coll);
        stage2InteractiveConcole(coll);
    }
    public static void stage1DataInputFromFile(FlatsCollection coll) {
        try {
            ReadFileFlats readFileFlats = new ReadFileFlats(coll);
            readFileFlats.read();
        }
        catch (IOException | NullException e) {
            e.printStackTrace();
            return;
        }
    }
    public static void stage2InteractiveConcole(FlatsCollection coll) {
        try {
            Executer executer = new Executer(coll);
            executer.executeFromInput();
        } catch (NullException e) { e.printStackTrace(); }
    }
}
```

Выводы по работе

Реализовал консольное приложение, которое реализует управление коллекцией объектов в интерактивном режиме. При выполнении лабораторной работы было изучено следующее: использование переменных окружения (**`System.getenv()`**); чтение / запись файлов и потоков ввода / вывода (**`java.util.Scanner`**, **`java.io.FileWriter`**); документация в формате **`javadoc`**; отображения **`java.util.HashMap`** и **`java.util.Map`**.