**Санкт-Петербургский национальный исследовательский университет ИТМО**

**Факультет программной инженерии и компьютерных технологий**

Отчет по лабораторной работе №5

Вариант 313123

**Выполнили:**

Савон Г.К.

Курашов О.И.

Студенты гр. P3131

**Проверил:**

Усков И.В.

Санкт-Петербург

2020

**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.TreeMap
* При запуске приложения коллекция должна автоматически заполняться значениями из файла.
* Имя файла должно передаваться программе с помощью: **аргумент командной строки**.
* Данные должны храниться в файле в формате xml
* Чтение данных из файла необходимо реализовать с помощью класса java.io.InputStreamReader
* Запись данных в файл необходимо реализовать с помощью класса java.io.BufferedOutputStream
* 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
* insert null {element} : добавить новый элемент с заданным ключом
* update id {element} : обновить значение элемента коллекции, id которого равен заданному
* remove\_key null : удалить элемент из коллекции по его ключу
* 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\_greater {element} : удалить из коллекции все элементы, превышающие заданный
* history : вывести последние 9 команд (без их аргументов)
* remove\_greater\_key null : удалить из коллекции все элементы, ключ которых превышает заданный
* average\_of\_number\_of\_rooms : вывести среднее значение поля numberOfRooms для всех элементов коллекции
* group\_counting\_by\_creation\_date : сгруппировать элементы коллекции по значению поля creationDate, вывести количество элементов в каждой группе
* count\_by\_transport transport : вывести количество элементов, значение поля transport которых равно заданному

**Диаграмма классов**

**A close up of a map

Description automatically generated**

**Данные для запуска программы:**

Login: s286536

Имя архива с программой: lab5.jar

Имя файла, в который сохраняется коллекция: Collection.xml

Для начала выполнения: java18 -jar lab5.jar Collection.xml

**Код программы**

**package** SaleOfApartments;  
  
**import** java.io.IOException;  
  
*/\*\*  
 \** ***@autors*** *Kurashov Oleg, Savon Galina  
 \** ***@version*** *19.0  
 \** ***@year*** *2020  
 \*/***public class** Main {  
 **public static void** main(String[] args) **throws** IOException{  
 **try** {  
 String s = args[0];  
 Runtime.*getRuntime*().addShutdownHook(**new** Thread(() -> System.***out***.println(**"Работа программы завершена!"**)));  
 *//String s = "C:\\Users\\Hannah\\Desktop\\proga\\lab5\_3\\Collection.xml";* Commander commander = **new** Commander(**new** CollectionManager(s));  
 commander.interactiveMod();  
 } **catch**(ArrayIndexOutOfBoundsException ex) {  
 System.***out***.println(**"Путь до файла xml нужно передать через аргумент командной строки."**);  
 System.*exit*(1);  
 }  
 }  
}

**package** SaleOfApartments;  
  
**public class** InputException **extends** Exception {  
  
}

**package** SaleOfApartments;  
  
**public class** RecursiveException **extends** Throwable {  
  
}

**package** SaleOfApartments;  
  
**public enum** Furnish {  
 ***DESIGNER***,  
 ***FINE***,  
 ***LITTLE***,  
 ***BAD***,  
 ***NONE***;  
}

**package** SaleOfApartments;  
  
**public enum** View {  
 ***PARK***,  
 ***STREET***,  
 ***BAD***;  
}

**package** SaleOfApartments;  
  
**public enum** Transport {  
 ***NONE***,  
 ***LITTLE***,  
 ***FEW***,  
 ***NORMAL***,  
 ***ENOUGH***;  
}

**package** SaleOfApartments;  
  
**import** java.util.Objects;  
  
**public class** Coordinates **implements** Comparable<Coordinates> {  
 **private float x**; *//Значение поля должно быть больше -227* **private** Long **y**; *//Максимальное значение поля: 769, Поле не может быть null* Coordinates(**float** x, Long y){  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
  
 **public float** getX() {  
 **return x**;  
 }  
 **public** Long getY() {  
 **return y**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Coordinates {"** +  
 **"x:"** + **x** + **", y:"** + **y** + **"} "**;  
 }  
  
 @Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (!(o **instanceof** Coordinates)) **return false**;  
 Coordinates coordinates = (Coordinates) o;  
 **return x** == coordinates.getX() &&  
 **y**.equals(coordinates.getY());  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**x**,**y**);  
 }  
  
 @Override  
 **public int** compareTo(Coordinates o) {  
 **return** (**int**) (**x**\***x** + **y**\***y** - o.getX()\*o.getX() - o.getY()\*o.getY());  
 }  
}

**package** SaleOfApartments;  
  
**import** java.util.Objects;  
  
**public class** House **implements** Comparable<House> {  
 **private** String **name**; *//Поле не может быть null* **private int year**; *//Значение поля должно быть больше 0* **private int numberOfFloors**; *//Значение поля должно быть больше 0* **private long numberOfFlatsOnFloor**; *//Значение поля должно быть больше 0* House(String name, **int** year, **int** numberOfFloors, **long** numberOfFlatsOnFloor){  
 **this**.**name** = name;  
 **this**.**year** = year;  
 **this**.**numberOfFloors** = numberOfFloors;  
 **this**.**numberOfFlatsOnFloor** = numberOfFlatsOnFloor;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
 **public int** getYear() {  
 **return year**;  
 }  
 **public int** getNumberOfFloors() {  
 **return numberOfFloors**;  
 }  
 **public long** getNumberOfFlatsOnFloor() {  
 **return numberOfFlatsOnFloor**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "House {"** +  
 **"name:"** + **name** + **", year:"** + **year** + **", number of floors:"** + **numberOfFloors** + **", number of flats on floor:"** + **numberOfFlatsOnFloor** + **"}"**;  
 }  
  
 @Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (!(o **instanceof** House)) **return false**;  
 House house = (House) o;  
 **return year** == house.getYear() &&  
 **numberOfFloors** == house.getNumberOfFloors() &&  
 **numberOfFlatsOnFloor** == house.getNumberOfFlatsOnFloor() &&  
 **name**.equals(house.getName());  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**name**, **year**, **numberOfFloors**, **numberOfFlatsOnFloor**);  
 }  
  
 @Override  
 **public int** compareTo(House o) {  
 **if** (**year** - o.getYear() != 0) **return year** - o.getYear();  
 **if** (**numberOfFloors** - o.getNumberOfFloors() != 0) **return numberOfFloors** - o.getNumberOfFloors();  
 **if** (**numberOfFlatsOnFloor** - o.getNumberOfFlatsOnFloor() != 0) **return** (**int**) (**numberOfFlatsOnFloor** - o.getNumberOfFlatsOnFloor());  
 **return name**.compareTo(o.getName());  
 }  
}

**package** SaleOfApartments;  
  
**import** java.io.\*;  
**import** java.util.\*;  
  
  
**public class** Commander {  
  
 **private** CollectionManager **manager**;  
 **private** String **userCommand**;  
 **private** String[] **finalUserCommand**;  
 **private** Scanner **commandReader** = **new** Scanner(System.***in***);  
 **private** ArrayList<String> **commands\_of\_script** =**new** ArrayList<>();  
 **protected static** ArrayList<String> *last\_commands* = **new** ArrayList<>();  
  
 {  
 **userCommand** = **""**;  
 **for** (**int** i = 0; i < 9; i++) {  
 *last\_commands*.add(**"."**);  
 }  
 }  
  
 **public** Commander(CollectionManager manager) {  
 **this**.**manager** = manager;  
 }  
  
 **public void** interactiveMod() {  
 **while** (!**userCommand**.equals(**"exit"**)) {  
 System.***out***.print(**">> "**);  
 **userCommand** = **commandReader**.nextLine();  
 **finalUserCommand** = **userCommand**.trim().split(**" "**, 2);  
 choseCommand();  
 }  
 }  
 **public void** choseCommand() {  
 **try** {  
 *last\_commands*.add(**finalUserCommand**[0]);  
 **switch** (**finalUserCommand**[0]) {  
 **case ""**:  
 **case "exit"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (!**finalUserCommand**[1].equals(**""**)) **throw new** InputException();  
 **else** System.*exit*(1);  
 }  
 **break**;  
 **case "help"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) **manager**.help();  
 **else throw new** InputException();  
 } **else manager**.help();  
 **break**;  
 **case "info"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) System.***out***.println(**manager**.toString());  
 **else throw new** InputException();  
 } **else** System.***out***.println(**manager**.toString());  
 **break**;  
 **case "show"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) **manager**.show();  
 **else throw new** InputException();  
 } **else manager**.show();  
 **break**;  
 **case "insert"**:  
 **manager**.insert(**finalUserCommand**[1]);  
 **break**;  
 **case "execute\_script"**:  
 ArrayList<String> commands\_of\_script =**new** ArrayList<>();  
 **manager**.execute\_script(**finalUserCommand**[1], commands\_of\_script);  
 **if** (commands\_of\_script.size() != 0) {  
 **for** (String command : commands\_of\_script) {  
 System.***out***.println(**">> "** + command);  
 **finalUserCommand** = command.trim().split(**" "**, 2);  
 choseCommand();  
 }  
 CollectionManager.*scripts*.remove(CollectionManager.*scripts*.size()-1);  
 }  
 **break**;  
 **case "update"**:  
 **manager**.update(**finalUserCommand**[1]);  
 **break**;  
 **case "remove\_key"**:  
 **manager**.remove\_key(**finalUserCommand**[1]);  
 **break**;  
 **case "clear"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) **manager**.clear();  
 **else throw new** InputException();  
 } **else manager**.clear();  
 **break**;  
 **case "save"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) **manager**.save();  
 **else throw new** InputException();  
 } **else manager**.save();  
 **break**;  
 **case "remove\_greater"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) **manager**.remove\_greater();  
 **else throw new** InputException();  
 } **else manager**.remove\_greater();  
 **break**;  
 **case "history"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) {  
 *last\_commands*.remove(*last\_commands*.size()-1);  
 **for**(**int** i = 0; i < 9; i++){  
 **if**(!*last\_commands*.get(i).equals(**"."**)) System.***out***.println(*last\_commands*.get(i));  
 }  
 *last\_commands*.add(**"history"**);  
 } **else throw new** InputException();  
 } **else** {  
 *last\_commands*.remove(*last\_commands*.size()-1);  
 **for**(**int** i = 0; i < *last\_commands*.size(); i++){  
 **if**(!*last\_commands*.get(i).equals(**"."**)) System.***out***.println(*last\_commands*.get(i));  
 }  
 *last\_commands*.add(**"history"**);  
 }  
 **break**;  
 **case "remove\_greater\_key"**:  
 **manager**.remove\_greater\_key(**finalUserCommand**[1]);  
 **break**;  
 **case "average\_of\_number\_of\_rooms"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) **manager**.average\_of\_number\_of\_rooms();  
 **else throw new** InputException();  
 } **else manager**.average\_of\_number\_of\_rooms();  
 **break**;  
 **case "group\_counting\_by\_creation\_date"**:  
 **if** (**finalUserCommand**.**length** > 1) {  
 **if** (**finalUserCommand**[1].equals(**""**)) **manager**.group\_counting\_by\_creation\_date();  
 **else throw new** InputException();  
 } **else manager**.group\_counting\_by\_creation\_date();  
 **break**;  
 **case "count\_by\_transport"**:  
 **manager**.count\_by\_transport(**finalUserCommand**[1]);  
 **break**;  
 **default**:  
 **throw new** InputException();  
 }  
 *last\_commands*.remove(0);  
 } **catch** (InputException e) {  
 System.***out***.println(**"Неопознанная команда. Наберите 'help' для справки."**);  
 *last\_commands*.remove(*last\_commands*.size()-1);  
 } **catch** (ArrayIndexOutOfBoundsException ex) {  
 System.***out***.println(**"Отсутствует аргумент"**);  
 } **catch** (IOException ex) {  
 System.***out***.println(**"Невозможно считать данные из файла."**);  
 }  
 }  
  
 @Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (!(o **instanceof** Commander)) **return false**;  
 Commander commander = (Commander) o;  
 **return manager**.equals(commander.**manager**);  
 }  
  
 @Override  
 **public int** hashCode() {  
 **int** result = Objects.*hash*(**manager**, **userCommand**);  
 result = 31 \* result + Arrays.*hashCode*(**finalUserCommand**);  
 **return** result;  
 }  
}

**package** SaleOfApartments;  
  
  
  
**import** org.jdom2.Document;  
**import** org.jdom2.Element;  
**import** org.jdom2.JDOMException;  
**import** org.jdom2.input.SAXBuilder;  
**import** org.jdom2.output.Format;  
**import** org.jdom2.output.XMLOutputter;  
  
**import** javax.management.modelmbean.XMLParseException;  
**import** java.io.\*;  
**import** java.time.\*;  
**import** java.util.\*;  
  
**public class** CollectionManager {  
 **private** TreeMap<String, Flat> **houses**;  
 **protected static** ArrayList<String> *scripts* = **new** ArrayList<>();  
 **private** String **collectionPath**;  
 **private** File **xmlCollection**;  
 **private** Date **initDate**;  
 **private boolean wasStart**;  
 **private** Iterator **iterator**;  
 **protected static** HashMap<String, String> *manual*;  
  
 {  
 **houses** = **new** TreeMap<>();  
 **iterator** = **houses**.entrySet().iterator();  
 *manual* = **new** HashMap<>();  
 *manual*.put(**"help"**,**"вывести справку по доступным командам"**);  
 *manual*.put(**"info"**,**"вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.)"**);  
 *manual*.put(**"show"**,**"вывести в стандартный поток вывода все элементы коллекции в строковом представлении"**);  
 *manual*.put(**"insert null {element}"**,**"добавить новый элемент с заданным ключом"**);  
 *manual*.put(**"update id {element}"**,**"обновить значение элемента коллекции, id которого равен заданному"**);  
 *manual*.put(**"remove\_key null"**,**"удалить элемент из коллекции по его ключу"**);  
 *manual*.put(**"clear"**,**"очистить коллекцию"**);  
 *manual*.put(**"save"**,**"сохранить коллекцию в файл"**);  
 *manual*.put(**"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"**);  
 *manual*.put(**"exit"**,**"завершить программу (без сохранения в файл)"**);  
 *manual*.put(**"remove\_greater {element}"**,**" удалить из коллекции все элементы, превышающие заданный"**);  
 *manual*.put(**"history"**,**"вывести последние 9 команд (без их аргументов)"**);  
 *manual*.put(**"remove\_greater\_key null"**,**"удалить из коллекции все элементы, ключ которых превышает заданный"**);  
 *manual*.put(**"average\_of\_number\_of\_rooms"**, **"вывести среднее значение поля numberOfRooms для всех элементов коллекции"**);  
 *manual*.put(**"group\_counting\_by\_creation\_date"**, **"сгруппировать элементы коллекции по значению поля creationDate, вывести количество элементов в каждой группе"**);  
 *manual*.put(**"count\_by\_transport transport"**, **"вывести количество элементов, значение поля transport которых равно заданному"**);  
 }  
  
 **public** CollectionManager(String collectionPath) {  
 **try** {  
 File file = **new** File(collectionPath);  
 **if** (file.exists()) {  
 **this**.**xmlCollection** = file;  
 **this**.**collectionPath** = collectionPath;  
 }  
 **else throw new** FileNotFoundException();  
 } **catch** (FileNotFoundException ex) {  
 System.***out***.println(**"Путь до файла xml нужно передать через аргумент командной строки. Файл по указанному пути не существует."**);  
 System.*exit*(1);  
 }  
 **this**.load();  
 **this**.**initDate** = **new** Date();  
 **wasStart** = **true**;  
 }  
  
  
 */\*\*  
 \* Выводит на экран список доступных для пользователя команд  
 \*/* **public void** help() {  
 System.***out***.println(**"Доступные к использованию команды:"**);  
 *manual*.keySet().forEach(p -> System.***out***.println(p + **" - "** + *manual*.get(p)));  
 }  
  
 */\*\*  
 \* Выводит все элементы коллекции  
 \*/* **public void** show() {  
 **if** (**houses**.size() != 0) {  
 **houses**.forEach((k,p) -> System.***out***.println(**"key: "** + k + **", "** + p));  
 }  
 **else** System.***out***.println(**"В коллекции отсутствуют элементы. Выполнение команды невозможно."**);  
 }  
  
 */\*\*  
 \* Получает значения элемента в коллекции  
 \*/* **public** Flat newFlat() {  
 Scanner reader = **new** Scanner(System.***in***);  
 System.***out***.print(**"Введите name: "**);  
 String name = reader.nextLine();  
 **while** (name.equals(**""**)) {  
 System.***out***.println(**"Поле не может быть null или пустой строкой "**);  
 System.***out***.print(**"Введите name: "**);  
 name = reader.nextLine();  
 }  
 System.***out***.println(**"Введите coordinates: "**);  
 String a;  
 **boolean** p = **false**;  
 **float** x = 0;  
 **while** (!p) {  
 System.***out***.print(**"Введите x: "**);  
 a = reader.nextLine();  
 **try** {  
 x = Float.*parseFloat*(a);  
 **if** (x > -227)  
 p = **true**;  
 **else** {  
 System.***out***.println(**"Значение поля должно быть больше -227"**);  
 }  
 } **catch** (NumberFormatException e) {  
 System.***out***.println(**"Аргумент не является значением типа float"**);  
 }  
 }  
 Long y = **null**;  
 **while** (p) {  
 System.***out***.print(**"Введите y: "**);  
 a = reader.nextLine();  
 **try** {  
 y = Long.*parseLong*(a);  
 **if** (y <= 769)  
 p = **false**;  
 **else** {  
 System.***out***.println(**"Значение поля должно быть меньше 769"**);  
 }  
 } **catch** (NumberFormatException e) {  
 System.***out***.println(**"Аргумент не является значением типа Long"**);  
 }  
 }  
 **long** area = 0;  
 **while** (!p) {  
 System.***out***.print(**"Введите area: "**);  
 a = reader.nextLine();  
 **try** {  
 area = Long.*parseLong*(a);  
 **if** (area > 0)  
 p = **true**;  
 **else** {  
 System.***out***.println(**"Значение поля должно быть больше 0"**);  
 }  
 } **catch** (NumberFormatException e) {  
 System.***out***.println(**"Аргумент не является значением типа long"**);  
 }  
 }  
 Integer numberOfRooms = **null**;  
 **while** (p) {  
 System.***out***.print(**"Введите numberOfRooms: "**);  
 a = reader.nextLine();  
 **try** {  
 numberOfRooms = Integer.*parseInt*(a);  
 **if** (numberOfRooms > 0)  
 p = **false**;  
 **else** {  
 System.***out***.println(**"Значение поля должно быть больше 0"**);  
 }  
 } **catch** (NumberFormatException e) {  
 System.***out***.println(**"Аргумент не является значением типа Integer"**);  
 }  
 }  
 System.***out***.print(**"Введите Furnish (DESIGNER, FINE, LITTLE, BAD, NONE, null): "**);  
 String furnish\_s = reader.nextLine();  
 **while** (!furnish\_s.equals(**""**) && !furnish\_s.equals(**"DESIGNER"**) && !furnish\_s.equals(**"FINE"**) && !furnish\_s.equals(**"LITTLE"**) && !furnish\_s.equals(**"BAD"**) && !furnish\_s.equals(**"NONE"**)) {  
 System.***out***.println(**"Значение поля неверное"**);  
 System.***out***.print(**"Введите Furnish (DESIGNER, FINE, LITTLE, BAD, NONE, null): "**);  
 furnish\_s = reader.nextLine();  
 }  
 Furnish furnish = **null**;  
 **if** (!furnish\_s.equals(**""**)) furnish = Furnish.*valueOf*(furnish\_s);  
 System.***out***.print(**"Введите View (PARK, STREET, BAD, null): "**);  
 String view\_s = reader.nextLine();  
 **while** (!view\_s.equals(**""**) && !view\_s.equals(**"STREET"**) && !view\_s.equals(**"YARD"**) && !view\_s.equals(**"BAD"**) && !view\_s.equals(**"NORMAL"**) && !view\_s.equals(**"GOOD"**)) {  
 System.***out***.println(**"Значение поля неверное"**);  
 System.***out***.print(**"Введите View (PARK, STREET, BAD, null): "**);  
 view\_s = reader.nextLine();  
 }  
 View view = **null**;  
 **if** (!view\_s.equals(**""**)) view = View.*valueOf*(view\_s);  
 System.***out***.print(**"Введите Transport (ENOUGH, NORMAL, FEW, LITTLE, NONE): "**);  
 String transport\_s = reader.nextLine();  
 **while** (!transport\_s .equals(**"FEW"**) && !transport\_s .equals(**"NONE"**) && !transport\_s .equals(**"LITTLE"**) && !transport\_s .equals(**"NORMAL"**) && !transport\_s .equals(**"ENOUGH"**)) {  
 System.***out***.println(**"Значение поля неверное"**);  
 System.***out***.print(**"Введите Transport (ENOUGH, NORMAL, FEW, LITTLE, NONE): "**);  
 transport\_s = reader.nextLine();  
 }  
 Transport transport = Transport.*valueOf*(transport\_s);  
 System.***out***.println(**"Введите House: "**);  
 System.***out***.print(**"Введите name: "**);  
 String nameHouse = reader.nextLine();  
 **while** (nameHouse.equals(**""**)) {  
 System.***out***.println(**"Поле не может быть null"**);  
 System.***out***.print(**"Введите name: "**);  
 nameHouse = reader.nextLine();  
 }  
 **int** year = 0;  
 **while** (!p) {  
 System.***out***.print(**"Введите year: "**);  
 a = reader.nextLine();  
 **try** {  
 year = Integer.*parseInt*(a);  
 **if** (year > 0)  
 p = **true**;  
 **else** {  
 System.***out***.println(**"Значение поля должно быть больше 0"**);  
 }  
 } **catch** (NumberFormatException e) {  
 System.***out***.println(**"Аргумент не является значением типа int"**);  
 }  
 }  
 **int** numberOfFloors = 0;  
 **while** (p) {  
 System.***out***.print(**"Введите numberOfFloors: "**);  
 a = reader.nextLine();  
 **try** {  
 numberOfFloors = Integer.*parseInt*(a);  
 **if** (numberOfFloors> 0)  
 p = **false**;  
 **else** {  
 System.***out***.println(**"Значение поля должно быть больше 0"**);  
 }  
 } **catch** (NumberFormatException e) {  
 System.***out***.println(**"Аргумент не является значением типа int"**);  
 }  
 }  
 **long** numberOfFlatsOnFloor = 0;  
 **while** (!p) {  
 System.***out***.print(**"Введите numberOfFlatsOnFloor: "**);  
 a = reader.nextLine();  
 **try** {  
 numberOfFlatsOnFloor = Long.*parseLong*(a);  
 **if** (numberOfFlatsOnFloor > 0)  
 p = **true**;  
 **else** {  
 System.***out***.println(**"Значение поля должно быть больше 0"**);  
 }  
 } **catch** (NumberFormatException e) {  
 System.***out***.println(**"Аргумент не является значением типа long"**);  
 }  
 }  
  
 **int** id = 0;  
 **while** (p) {  
 p = **false**;  
 Random random = **new** Random();  
 id = random.nextInt(10000) + 1;  
 **for** (Flat h : **houses**.values()) {  
 **if** (h.getId() == id) {  
 p = **true**;  
 **break**;  
 }  
 }  
 }  
 LocalDate d = LocalDate.*now*();  
 LocalTime t = LocalTime.*now*();  
 LocalDateTime creationDate = LocalDateTime.*of*(d,t);  
 System.***out***.println(**"Все значения элемента успешно получены"**);  
 **return new** Flat(id, name, **new** Coordinates(x, y), creationDate, area, numberOfRooms, furnish, view, transport, **new** House(nameHouse, year, numberOfFloors, numberOfFlatsOnFloor));  
 }  
  
 **public void** insert(String key) {  
 **if** (!**houses**.containsKey(key)) {  
 **houses**.put(key, newFlat());  
 System.***out***.println(**"Элемент успешно добавлен"**);  
 } **else** System.***out***.println(**"Элемент с данным ключом уже существует."**);  
 }  
  
 */\*\*  
 \* Обновляет значение элемента коллекции, id которого равен заданному  
 \** ***@param n*** *: Id элемента, который требуется заменить  
 \*/* **public void** update(String n){  
 **if** (**houses**.size() != 0) {  
 Integer id = Integer.*valueOf*(n);  
 **boolean** b = **false**;  
 **iterator** = **houses**.keySet().iterator();  
 **while** (**iterator**.hasNext()) {  
 String key = (String) **iterator**.next();  
 **if** (**houses**.get(key).getId().equals(id)) {  
 **houses**.replace(key,newFlat());  
 System.***out***.println(**"Элемент коллекции успешно обновлен."**);  
 b = **true**;  
 **break**;  
 }  
 }  
 **if** (!b) System.***out***.println(**"В коллекции не найдено элемента с указанным id."**);  
 }  
 **else** System.***out***.println(**"В коллекции отсутствуют элементы. Выполнение команды не возможно."**);  
 }  
  
 */\*\*  
 \* Удаляет элемент из коллекции по его ключу  
 \** ***@param n*** *: ключ соответствующего элемента, который требуется удалить  
 \*/* **public void** remove\_key(String n){  
 **if** (**houses**.size() != 0) {  
 **if** (!n.equals(**""**)) {  
 **boolean** b = **false**;  
 **iterator** = **houses**.keySet().iterator();  
 **while** (**iterator**.hasNext()) {  
 String key = (String) **iterator**.next();  
 **if** (key.equals(n)) {  
 **houses**.remove(key);  
 System.***out***.println(**"Элемент коллекции успешно удален."**);  
 b = **true**;  
 **break**;  
 }  
 }  
 **if** (!b) System.***out***.println(**"В коллекции не найдено элемента с указанным ключом."**);  
 } **else** System.***out***.println(**"Отсутсвует аргумент"**);  
 }  
 **else** System.***out***.println(**"В коллекции отсутствуют элементы. Выполнение команды не возможно."**);  
 }  
  
 */\*\*  
 \* Удаляет все элементы коллекции.  
 \*/* **public void** clear() {  
 **houses**.clear();  
 System.***out***.println(**"Коллекция очищена."**);  
 }  
  
 */\*\*  
 \* Сериализует коллекцию в файл json.  
 \*/* **public void** save() {  
 **try** {  
 Document doc = **new** Document();  
 *// создаем корневой элемент с пространством имен* doc.setRootElement(**new** Element(**"Flats"**));  
 *// формируем JDOM документ из объектов Student* **for** (String key : **houses**.keySet()) {  
 Element element = **new** Element(**"Flat"**);  
 element.setAttribute(**"key"**, key);  
 element.addContent(**new** Element(**"id"**).setText( String.*valueOf*(**houses**.get(key).getId())));  
 element.addContent(**new** Element(**"name"**).setText(**houses**.get(key).getName()));  
 Element element\_c = **new** Element(**"Coordinates"**);  
 element\_c.addContent(**new** Element(**"x"**).setText(String.*valueOf*(**houses**.get(key).getCoordinates().getX())));  
 element\_c.addContent(**new** Element(**"y"**).setText(String.*valueOf*(**houses**.get(key).getCoordinates().getY())));  
 element.addContent(element\_c);  
 element.addContent(**new** Element(**"creationDate"**).setText(String.*valueOf*(**houses**.get(key).getCreationDate())));  
 element.addContent(**new** Element(**"area"**).setText(String.*valueOf*(**houses**.get(key).getArea())));  
 element.addContent(**new** Element(**"numberOfRooms"**).setText(String.*valueOf*(**houses**.get(key).getNumberOfRooms())));  
 element.addContent(**new** Element(**"view"**).setText(String.*valueOf*(**houses**.get(key).getView())));  
 element.addContent(**new** Element(**"furnish"**).setText(String.*valueOf*(**houses**.get(key).getFurnish())));  
 element.addContent(**new** Element(**"view"**).setText(String.*valueOf*(**houses**.get(key).getView())));  
 element.addContent(**new** Element(**"transport"**).setText(String.*valueOf*(**houses**.get(key).getTransport())));  
 Element element\_d = **new** Element(**"House"**);  
 element\_d.addContent(**new** Element(**"name"**).setText(**houses**.get(key).getHouse().getName()));  
 element\_d.addContent(**new** Element(**"year"**).setText(String.*valueOf*(**houses**.get(key).getHouse().getYear())));  
 element\_d.addContent(**new** Element(**"numberOfFloors"**).setText(String.*valueOf*(**houses**.get(key).getHouse().getNumberOfFloors())));  
 element\_d.addContent(**new** Element(**"numberOfFlatsOnFloor"**).setText(String.*valueOf*(**houses**.get(key).getHouse().getNumberOfFlatsOnFloor())));  
 element.addContent(element\_d);  
 doc.getRootElement().addContent(element);  
 }  
 *// Документ JDOM сформирован и готов к записи в файл* XMLOutputter xmlWriter = **new** XMLOutputter(Format.*getPrettyFormat*());  
 *// сохнаряем в файл* xmlWriter.output(doc, **new** FileOutputStream(**xmlCollection**));  
 System.***out***.println(**"Коллекция успешно сохранена в файл."**);  
 } **catch** (IOException ex) {  
 System.***out***.println(**"Возникла непредвиденная ошибка. Коллекция не может быть записана в файл"**);  
 }  
 }  
  
 */\*\*  
 \* Считывает и исполняет скрипт из указанного файла.  
 \* В скрипте содержатся команды в таком же виде, в котором их вводит пользователь в интерактивном режиме  
 \*/* **public void** execute\_script(String file, ArrayList<String> commands\_of\_script) **throws** IOException {  
 **try** {  
 **if** (file.charAt(1) != **'/'**) file = System.*getenv*(**"PWD"**) + **"/"**+ file;  
 **if** (*scripts*.contains(file)) **throw new** RecursiveException();  
 *scripts*.add(file);  
 **try** (InputStreamReader commandReader = **new** InputStreamReader(**new** FileInputStream(**new** File(file)))) {  
 StringBuilder s = **new** StringBuilder();  
 **while** (commandReader.ready()) s.append((**char**)commandReader.read());  
 String[] s1 = s.toString().split(**"\n"**);  
 commands\_of\_script.addAll(Arrays.*asList*(s1));  
 }  
 } **catch** (FileNotFoundException ex) {  
 System.***out***.println(**"Скрипт по указанному пути не существует"**);  
 *scripts*.remove(*scripts*.size()-1);  
 } **catch** (RecursiveException ex) {  
 Commander.*last\_commands*.remove(Commander.*last\_commands*.size()-1);  
 System.***out***.println(**"Могло произойти зацикливание при исполнении скрипта: "** + file + **"\nКоманда не будет выполнена. Переход к следующей команде"**);  
 }  
 }  
  
 */\*\*  
 \* Удаляет из коллекции все элементы, превышающие заданный  
 \*/* **public void** remove\_greater() {  
 **if** (**houses**.size() != 0) {  
 Flat o = newFlat();  
 **iterator** = **houses**.keySet().iterator();  
 **while** (**iterator**.hasNext()) {  
 **iterator**.next();  
 **if** (**houses**.get(**iterator**).compareTo(o) > 0) {  
 **houses**.remove(**iterator**);  
 }  
 }  
 } **else** System.***out***.println(**"В коллекции отсутствуют элементы. Выполнение команды не возможно."**);  
 }  
  
 */\*\*  
 \* Удаляет из коллекции все элементы, ключ которых превышает заданный  
 \** ***@param n*** *: ключ, относительно которого удалются все элементы с ключом большим чем этот  
 \*/* **public void** remove\_greater\_key(String n) {  
 **if** (**houses**.size() != 0) {  
 **iterator** = **houses**.keySet().iterator();  
 **while** (**iterator**.hasNext()) {  
 String s = (String) **iterator**.next();;  
 **if** (s.compareTo(n) > 0) {  
 **houses**.remove(s);  
 }  
 }  
 }  
 **else** System.***out***.println(**"В коллекции отсутствуют элементы. Выполнение команды не возможно."**);  
 }  
  
 */\*\*  
 \* Выводит среднее значение поля numberOfRooms для всех элементов коллекции  
 \*/* **public void** average\_of\_number\_of\_rooms() {  
 **if** (**houses**.size() != 0) {  
 **int** sum\_number\_of\_rooms = 0;  
 **for** (String key: **houses**.keySet()) {  
 sum\_number\_of\_rooms += **houses**.get(key).getNumberOfRooms();  
 }  
 System.***out***.println(**"Cреднее значение поля numberOfRooms для всех элементов коллекции: "** + sum\_number\_of\_rooms/**houses**.size());  
 }  
 **else** System.***out***.println(**"В коллекции отсутствуют элементы. Выполнение команды не возможно."**);  
 }  
  
 */\*\*  
 \* Группирует элементы коллекции по значению поля creationDate, вывести количество элементов в каждой группе  
 \*/* **public void** group\_counting\_by\_creation\_date(){  
 **if** (**houses**.size() != 0) {  
 HashMap<LocalDateTime,Integer> creationDates = **new** HashMap<>();  
 **for** (String key: **houses**.keySet()) {  
 **if** (creationDates.containsKey(**houses**.get(key).getCreationDate())) {  
 creationDates.replace(**houses**.get(key).getCreationDate(),creationDates.get(**houses**.get(key).getCreationDate()) + 1);  
 } **else** creationDates.put(**houses**.get(key).getCreationDate(),1);  
 }  
 creationDates.forEach((k,p) -> System.***out***.println(k + **": "** + p));  
 } **else** System.***out***.println(**"В коллекции отсутствуют элементы. Выполнение команды не возможно."**);  
 }  
  
 */\*\*  
 \* Выводит количество элементов, значение поля transport которых равно заданному  
 \** ***@param transport\_s*** *: значение поля transport  
 \*/* **public void** count\_by\_transport(String transport\_s){  
 **if** (**houses**.size() != 0) {  
 **try** {  
 **int** count\_by\_transport = 0;  
 Transport transport = Transport.*valueOf*(transport\_s);  
 **for** (String key : **houses**.keySet()) {  
 **if** (**houses**.get(key).getTransport().equals(transport))  
 count\_by\_transport++;  
 }  
 System.***out***.println(**"Количество элементов, значение поля transport которых равно "** + transport\_s + **": "** + count\_by\_transport);  
 } **catch** (IllegalArgumentException | NullPointerException ex) {  
 System.***out***.println(**"Значение поля Transport некорректно. Возможные значения: FEW, NONE, LITTLE, NORMAL, ENOUGH."**);  
 }  
 } **else** System.***out***.println(**"В коллекции отсутствуют элементы. Выполнение команды не возможно."**);  
 }  
  
 */\*\*  
 \* Десериализует коллекцию из файла json.  
 \*/* **public void** load() {  
 **int** beginSize = **houses**.size();  
 **try** {  
 **if** (!**xmlCollection**.exists()) **throw new** FileNotFoundException();  
 } **catch** (FileNotFoundException ex) {  
 System.***out***.println(**"Файла по указанному пути не существует."**);  
 **if** (!**wasStart**) System.*exit*(1);  
 **else return**;  
 }  
 **try** {  
 **if** (!**xmlCollection**.canRead() || !**xmlCollection**.canWrite()) **throw new** SecurityException();  
 } **catch** (SecurityException ex) {  
 System.***out***.println(**"Файл защищён от чтения и/или записи. Для работы программы нужны оба разрешения."**);  
 **if** (!**wasStart**) System.*exit*(1);  
 **else return**;  
 }  
 **try** {  
 **if** (**xmlCollection**.length() == 0) **throw new** XMLParseException(**""**);  
 } **catch** (XMLParseException ex) {  
 System.***out***.println(**"Файл пуст."**);  
 **if** (!**wasStart**) System.*exit*(1);  
 **else return**;  
 }  
 **try** {  
 System.***out***.println(**"Идёт загрузка коллекции "** + **xmlCollection**.getAbsolutePath());  
 *// мы можем создать экземпляр JDOM Document из классов DOM, SAX и STAX Builder* org.jdom2.Document jdomDocument = *createJDOMusingSAXParser*(**collectionPath**);  
 Element root = jdomDocument.getRootElement();  
 *// получаем список всех элементов* List<Element> labWorkListElements = root.getChildren(**"Flat"**);  
 *// список объектов Student, в которых будем хранить  
 // считанные данные по каждому элементу* **for** (Element lab : labWorkListElements) {  
  
 String key = lab.getAttributeValue(**"key"**);  
 Integer id = Integer.*parseInt*(lab.getChildText(**"id"**));  
 String name = lab.getChildText(**"name"**);  
 List<Element> lab\_c = lab.getChildren(**"Coordinates"**);  
 **float** x = Float.*parseFloat*(lab\_c.get(0).getChildText(**"x"**));  
 Long y = Long.*parseLong*(lab\_c.get(0).getChildText(**"y"**));  
 LocalDateTime creationDate = LocalDateTime.*parse*(lab.getChildText(**"creationDate"**));  
 **long** area = Long.*parseLong*(lab.getChildText(**"area"**));  
 Integer numberOfRooms = Integer.*parseInt*(lab.getChildText(**"numberOfRooms"**));  
 Furnish furnish = **null**;  
 String furnish\_s = lab.getChildText(**"furnish"**);  
 **if** (!furnish\_s.equals(**"null"**)) furnish = Furnish.*valueOf*(furnish\_s);  
 View view = **null**;  
 String view\_s = lab.getChildText(**"view"**);  
 **if** (!view\_s.equals(**"null"**)) view = View.*valueOf*(view\_s);  
 Transport transport = Transport.*valueOf*(lab.getChildText(**"transport"**));  
 List<Element> lab\_d = lab.getChildren(**"House"**);  
 String nameHouse = lab\_d.get(0).getChildText(**"name"**);  
 **int** year = Integer.*parseInt*(lab\_d.get(0).getChildText(**"year"**));  
 **int** numberOfFloors = Integer.*parseInt*(lab\_d.get(0).getChildText(**"numberOfFloors"**));  
 **long** numberOfFlatsOnFloor = Long.*parseLong*(lab\_d.get(0).getChildText(**"numberOfFlatsOnFloor"**));  
 **houses**.put(key, **new** Flat(id, name, **new** Coordinates(x, y), creationDate, area, numberOfRooms, furnish, view, transport, **new** House(nameHouse, year, numberOfFloors, numberOfFlatsOnFloor)));  
 }  
 }**catch** (Exception e) {  
 System.***out***.println(**"Не удалось загрузить коллекцию. Всё очеь-очень плохо!"**);  
 }  
 System.***out***.println(**"Коллекция успешно загружена. Добавлено "** + (**houses**.size() - beginSize) + **" элементов."**);  
 }  
  
 **private static** org.jdom2.Document createJDOMusingSAXParser(String fileName)  
 **throws** JDOMException, IOException {  
 SAXBuilder saxBuilder = **new** SAXBuilder();  
 **return** saxBuilder.build(**new** File(fileName));  
 }  
  
 */\*\*  
 \* Выводит информацию о коллекции.  
 \*/* @Override  
 **public** String toString() {  
 **return "Тип коллекции: "** + **houses**.getClass() +  
 **"\nДата инициализации: "** + **initDate** +  
 **"\nКоличество элементов: "** + **houses**.size();  
 }  
  
 @Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (!(o **instanceof** CollectionManager)) **return false**;  
 CollectionManager manager = (CollectionManager) o;  
 **return houses**.equals(manager.**houses**) &&  
 **xmlCollection**.equals(manager.**xmlCollection**) &&  
 **initDate**.equals(manager.**initDate**);  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**houses**, **initDate**);  
 }  
}

**package** SaleOfApartments;  
  
**import** java.time.LocalDateTime;  
**import** java.util.Objects;  
  
**public class** Flat **implements** Comparable<Flat>{  
 **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* **private** String **name**; *//Поле не может быть null, Строка не может быть пустой* **private** Coordinates **coordinates**; *//Поле не может быть null* **private** LocalDateTime **creationDate**; *//Поле не может быть null, Значение этого поля должно генерироваться автоматически* **private long area**; *//Значение поля должно быть больше 0* **private** Integer **numberOfRooms**; *//Значение поля должно быть больше 0* **private** Furnish **furnish**; *//Поле может быть null* **private** View **view**; *//Поле может быть null* **private** Transport **transport**; *//Поле не может быть null* **private** House **house**; *//Поле не может быть null* **public** Flat(Integer id, String name, Coordinates coordinates, LocalDateTime creationDate, **long** area, Integer numberOfRooms, Furnish furnish, View view, Transport transport, House house){  
 **this**.**id** = id;  
 **this**.**name** = name;  
 **this**.**coordinates** = coordinates;  
 **this**.**creationDate** = creationDate;  
 **this**.**area** = area;  
 **this**.**numberOfRooms** = numberOfRooms;  
 **this**.**furnish** = furnish;  
 **this**.**view** = view;  
 **this**.**transport** = transport;  
 **this**.**house** = house;  
 }  
  
 @Override  
 **public int** compareTo(Flat e) {  
 **if** (**area** != e.getArea()) **return** (**int**) (**area** - e.getArea());  
 **if** (!**numberOfRooms**.equals(e.**numberOfRooms**)) **return numberOfRooms** - e.getNumberOfRooms();  
 **if** (**transport**.compareTo(e.getTransport()) != 0) **return** -(**transport**.compareTo(e.getTransport()));  
 **if** (**furnish**.compareTo(e.getFurnish()) != 0) **return furnish**.compareTo(e.getFurnish());  
 **if** (**view**.compareTo(e.getView()) != 0) **return view**.compareTo(e.getView());  
 **if** (**coordinates**.compareTo(e.getCoordinates()) != 0) **return coordinates**.compareTo(e.getCoordinates());  
 **if** (**house**.compareTo(e.getHouse()) != 0) **return house**.compareTo(e.getHouse());  
 **if** (**name**.compareTo(e.getName()) != 0) **return name**.compareTo(e.getName());  
 **return id** - e.getId();  
 }  
  
 **public** Integer getId(){  
 **return id**;  
 }  
 **public** String getName(){  
 **return name**;  
 }  
 **public** Coordinates getCoordinates(){  
 **return coordinates**;  
 }  
 **public long** getArea(){  
 **return area**;  
 }  
 **public** Integer getNumberOfRooms(){  
 **return numberOfRooms**;  
 }  
 **public** View getView(){  
 **return view**;  
 }  
 **public** Furnish getFurnish(){  
 **return furnish**;  
 }  
 **public** Transport getTransport() {  
 **return transport**;  
 }  
 **public** House getHouse(){  
 **return house**;  
 }  
 **public** LocalDateTime getCreationDate(){  
 **return creationDate**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Flat {"** +  
 **"id:"** + **id** + **", name:"** + **name** + **", "** + **coordinates**.toString() + **", creation date:"** + **creationDate** + **", area:"** + **area** + **", number of rooms:"** + **numberOfRooms** + **", view:"** + **view** +  
 **", furnish:"** + **furnish** + **", transport:"** + **transport** + **", "** + **house**.toString() + **"}"**;  
 }  
  
 @Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (!(o **instanceof** Flat)) **return false**;  
 Flat flat = (Flat) o;  
 **return area** == flat.**area** &&  
 **name**.equals(flat.**name**) &&  
 **coordinates**.equals(flat.**coordinates**) &&  
 **creationDate**.equals(flat.**creationDate**) &&  
 **id**.equals(flat.**id**) &&  
 **numberOfRooms**.equals(flat.**numberOfRooms**) &&  
 **view**.equals(flat.**view**) &&  
 **furnish**.equals(flat.**furnish**) &&  
 **transport**.equals(flat.**transport**) &&  
 **house**.equals(flat.**house**);  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**id**, **name**, **coordinates**, **creationDate**, **area**, **numberOfRooms**, **view**, **transport**, **house**);  
 }  
}

**Результат работы программы:**

Программа работает

**Вывод:**

