Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет ПИиКТ

Программирование

Лабораторная работа №5

Вариант 10122

Выполнила: Наумова Надежда

Группа P3101

Преподаватель: Гаврилов Антон Валерьевич

Санкт-Петербург

2019 г.

# Задание :

Реализовать на базе программы из лабораторной работы №4 консольное приложение, которое реализует управление коллекцией объектов в интерактивном режиме. Номенклатуру сохраняемых в коллекции объектов необходимо заранее согласовать с преподавателем.

**Разработанная программа должна удовлетворять следующим требованиям:**

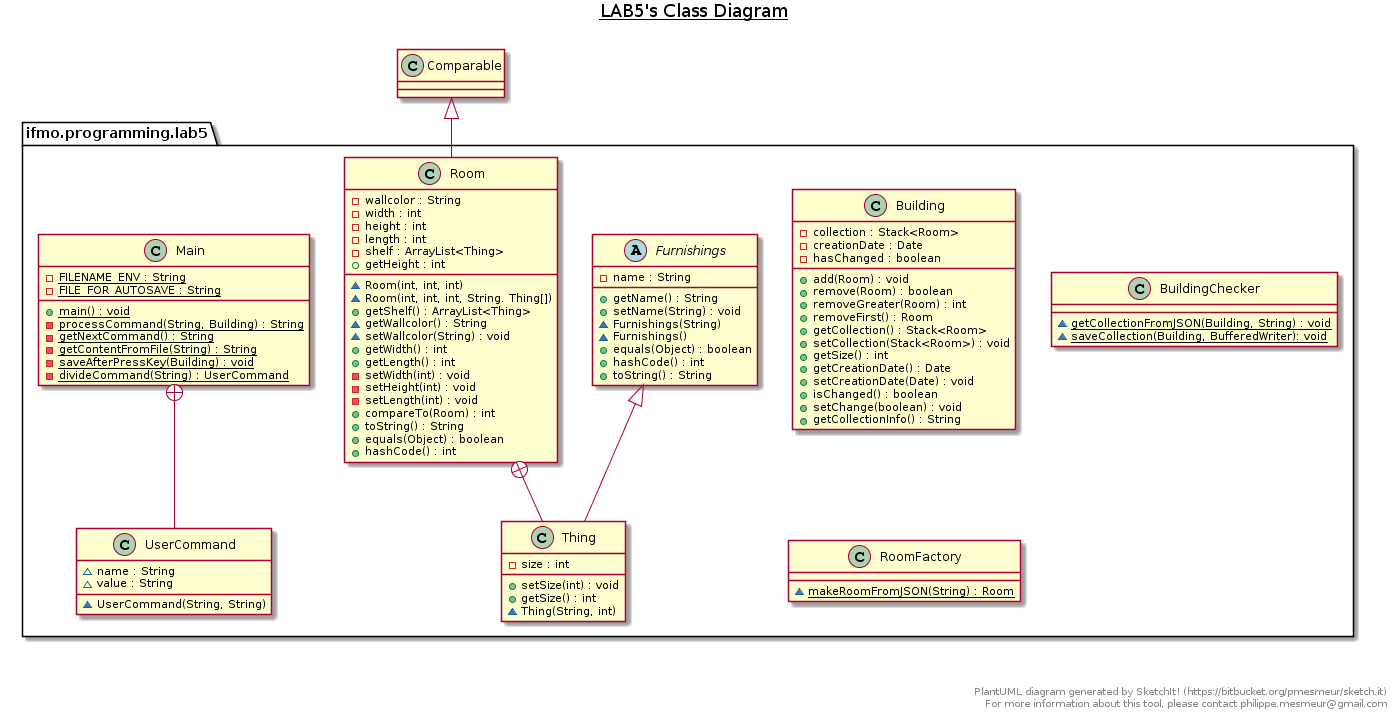
* Класс, коллекцией экземпляров которого управляет программа, должен реализовывать сортировку по умолчанию.
* Для хранения необходимо использовать коллекцию типа java.util.Stack.
* При запуске приложения коллекция должна автоматически заполняться значениями из файла.
* Имя файла должно передаваться программе с помощью переменной окружения.
* Данные должны храниться в файле в формате json.
* При остановке приложения текущее состояние коллекции должно автоматически сохраняться в файл.
* Чтение данных из файла необходимо реализовать с помощью класса java.io.FileReader.
* Запись данных в файл необходимо реализовать с помощью класса java.io.BufferedWriter.
* Все реализованные команды (см. ниже) должны быть задокументированы в формате javadoc.
* Формат задания объектов в командах - json.

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

* add {element}: добавить новый элемент в коллекцию
* remove\_first: удалить первый элемент из коллекции
* remove\_greater {element}: удалить из коллекции все элементы, превышающие заданный
* show: вывести в стандартный поток вывода все элементы коллекции в строковом представлении
* info: вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.)
* load: перечитать коллекцию из файла
* remove {element}: удалить элемент из коллекции по его значению

**Отчёт по работе должен содержать:**

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

Исходный код программы

**Building.java**

package ifmo.programming.lab5;  
  
import java.io.FileNotFoundException;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.Map;  
import java.util.Stack;  
  
import static ifmo.programming.lab5.BuildingChecker.*getCollectionFromJSON*;  
import static ifmo.programming.lab5.Main.*getContentFromFile*;  
  
*/\*\*  
 \* Хрущёвка  
 \*/*public class Building {  
 private static final String *FILENAME\_ENV* = "loadfile";  
 private Stack<Room> collection = new Stack<>();  
 private Date creationDate = new Date();  
 private boolean hasChanged = false;  
  
 */\*\*  
 \* Добавляет команту в здание  
 \** ***@param*** *room комната, которую нужно добавить  
 \*/* public void add(Room room) {  
 collection.push(room);  
 hasChanged = true;  
 }  
  
 */\*\*  
 \* Перечитывает коллекцию из файла  
 \** ***@return*** *Количество загруженных элементов  
 \** ***@throws*** *Exception когда что-то идёт не так  
 \*/* public int load () throws Exception {  
 Map<String, String> env = System.*getenv*();  
 String filename = env.get(*FILENAME\_ENV*);  
 int initialSize = collection.size();  
  
 if (filename != null && !filename.isEmpty()) {  
 String content = *getContentFromFile*(filename);  
// collection.clear();  
 *getCollectionFromJSON*(this, content);  
 }  
 return collection.size() - initialSize;  
 }  
  
 */\*\*  
 \* Удаляет комнату из здания  
 \** ***@param*** *room комната, которую нужно удалить  
 \** ***@return*** *true, если комната удалена  
 \*/* public boolean remove(Room room) {  
 if (!collection.contains(room))  
 return false;  
 collection.remove(room);  
 hasChanged = true;  
 return true;  
 }  
  
 */\*\*  
 \* Удаляет каждую комнату, превыщающую указанную  
 \** ***@param*** *room Комната, с которой происходит сравнение  
 \** ***@return*** *Количество удалённых комнат  
 \*/* public int remove\_greater(Room room) {  
 ArrayList<Room> removingRooms = new ArrayList<>();  
 for (Room current : collection) {  
 if (current.compareTo(room) > 0)  
 removingRooms.add(current);  
 }  
 for (Room removing : removingRooms)  
 collection.remove(removing);  
  
 if (removingRooms.size() > 0)  
 hasChanged = true;  
  
 return removingRooms.size();  
 }  
  
 */\*\*  
 \* Удаляет первый элемент стека  
 \** ***@return*** *удалённый элемент (вернет null, если стек пуст)  
 \*/* public Room removeFirst() {  
 Stack<Room> snew = new Stack<>();  
 while(collection.size() > 1)  
 snew.push(collection.pop());  
  
 if (collection.size() == 1) {  
 Room removed = collection.pop();  
 collection.clear();  
 while (snew.size() > 0)  
 collection.push(snew.pop());  
  
 hasChanged = true;  
  
 return removed;  
 }  
  
 return null;  
 }  
  
 public Stack<Room> getCollection() {  
 return collection;  
 }  
  
 public void setCollection(Stack<Room> collection) {  
 this.collection = collection;  
 hasChanged = true;  
 }  
  
 public int getSize() {  
 return collection.size();  
 }  
  
 public Date getCreationDate() {  
 return creationDate;  
 }  
  
 public void setCreationDate(Date createdDate) {  
 this.creationDate = createdDate;  
 hasChanged = true;  
 }  
  
 */\*\*  
 \** ***@return*** *true, если содержимое коллекции было изменено  
 \*/* public boolean isChanged() { return hasChanged; }  
  
 */\*\*  
 \* Устанавливает состояние изменённости коллекции. Если передатть true, коллекция  
 \* будет обозначена как изменённая. Рекомендуется вызывать функцию с переданным false  
 \* после сохранения состояния коллекции в файл  
 \** ***@param*** *hasChanged Если true, коллекция изменена  
 \*/* public void setChange(boolean hasChanged) { this.hasChanged = hasChanged; }  
  
 */\*\*  
 \*  
 \** ***@return*** *читабельное строковое представление коллекции  
 \*/* public String getCollectionInfo() {  
 return "Коллекция типа " + collection.getClass().getName() + ",\n" +  
 "дата создания: " + creationDate + ",\n" +  
 "содержит " + collection.size() + " элементов";  
 }  
  
 */\*\*  
 \* справка по командам, реализуемым приложением  
 \** ***@return*** *справка по командам, реализуемым приложением  
 \*/* public String getHelp() {  
 return "Приложение поддерживает выполнение следующих команд:" +  
 "\n\t• add {element}: добавить новый элемент в коллекцию;" +  
 "\n\t• remove\_first: удалить первый элемент из коллекции;" +  
 "\n\t• remove\_greater {element}: удалить из коллекции все элементы, превышающие заданный;" +  
 "\n\t• show: вывести в стандартный поток вывода все элементы коллекции в строковом представлении;" +  
 "\n\t• info: вывести в стандартный поток вывода информацию о коллекции;" +  
 "\n\t• load: перечитать коллекцию из файла;" +  
 "\n\t• remove {element}: удалить элемент из коллекции по его значению;" +  
 "\n\t• help: вызов справки." ;  
  
  
 }  
}

# Furnishings.java

package ifmo.programming.lab5;  
  
import java.util.Objects;  
  
public abstract class Furnishings {  
 private String name;  
  
 public String getName(){  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 Furnishings (String name) {setName(name);}  
 Furnishings () {}  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Furnishings that = (Furnishings) o;  
 return Objects.*equals*(name, that.name);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(name);  
 }  
  
 @Override  
 public String toString() {  
 return "Furnishings{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
}

**Room.java**

package ifmo.programming.lab5;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Objects;  
  
public class Room implements Comparable<Room> {  
 private String wallcolor;  
 private int width, height, length;  
 private ArrayList<Thing> shelf;  
  
 Room(int width, int height, int length) {  
 this(width, height, length, "");  
 }  
  
 Room(int width, int height, int length, String wallcolor, Thing... things) {  
 setWidth(width);  
 setHeight(height);  
 setLength(length);  
 setWallcolor(wallcolor);  
 shelf = new ArrayList<Thing>(Arrays.*asList*(things));  
  
 }  
  
 public ArrayList<Thing> getShelf() { return shelf; }  
  
 String getWallcolor() { return wallcolor; }  
 void setWallcolor(String wallcolor) {  
 if (wallcolor == null) { throw new IllegalArgumentException("Устанавливаемый цвет не должен быть null"); }  
 this.wallcolor = wallcolor;  
 }  
  
 public int getWidth() { return width; }  
 public int getHeight() { return height; }  
 public int getLength() { return length; }  
  
 private void setWidth(int width) { this.width = width; }  
 private void setHeight(int height) { this.height = height; }  
 private void setLength(int length) { this.length = length; }  
  
  
 @Override  
 public int compareTo(Room o) {  
 return getWidth()\*getHeight()\*getLength() - o.getWidth()\*o.getHeight()\*getLength();  
 }  
  
 @Override  
 public String toString() {  
 String roominfo = new String("Комната ");  
 if (wallcolor.isEmpty()) {  
 roominfo = roominfo.concat("с прозрачными стенами, ");  
 }  
 else {  
 String color = "цвета " + wallcolor + ", ";  
 roominfo = roominfo.concat(color);  
 }  
 roominfo = roominfo.concat("имеющая размеры: " + width + " x " + height + " x " + length + ", ");  
 if (shelf.size() == 0) {roominfo = roominfo.concat("пустая.");}  
 else {  
 roominfo = roominfo.concat("содержащая " + shelf.size() + " предметов.");  
 }  
 return roominfo;  
  
 }  
  
 public static class Thing extends Furnishings {  
 private int size;  
 public void setSize (int size) {this.size = size;}  
 public int getSize () {return size;}  
  
 Thing(String name, int size) {  
 this.setName(name);  
 this.setSize(size);  
 }  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Room room = (Room) o;  
 return width == room.width &&  
 height == room.height &&  
 length == room.length &&  
 Objects.*equals*(wallcolor, room.wallcolor) &&  
 Objects.*equals*(shelf, room.shelf);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(wallcolor, width, height, length, shelf);  
 }  
}

**Main.java**

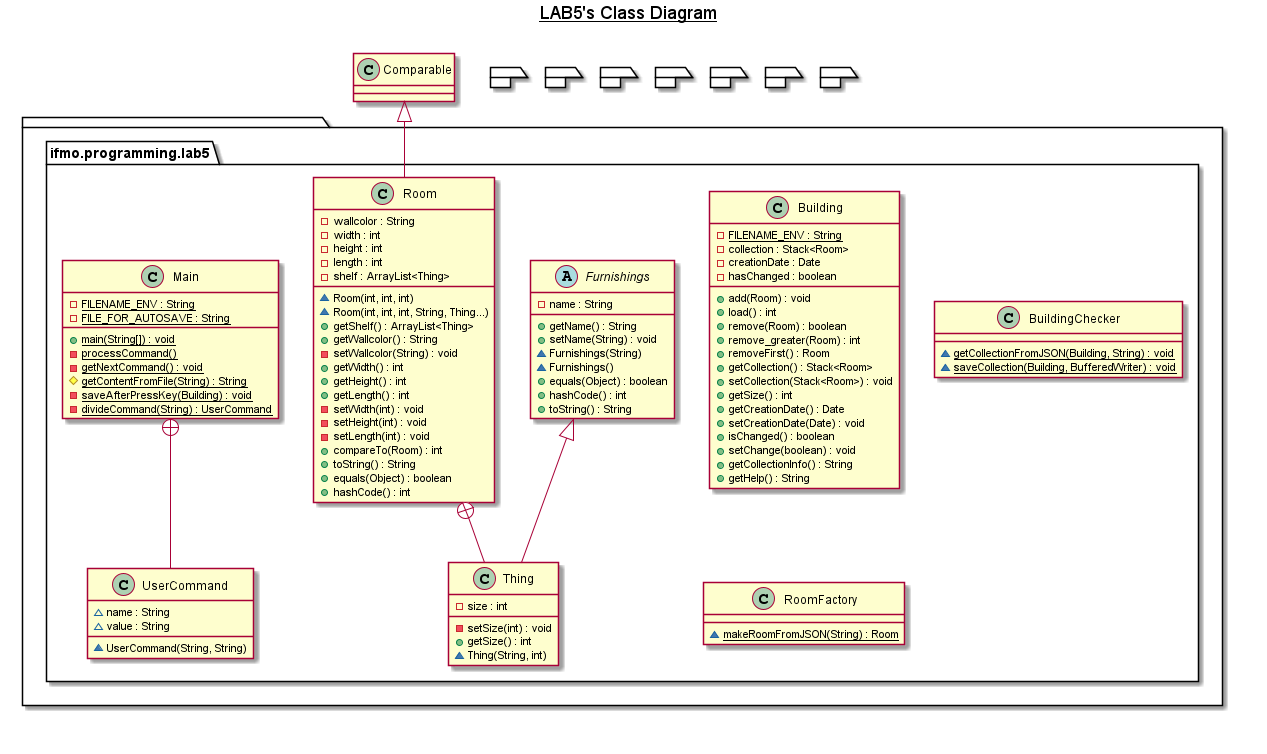
package ifmo.programming.lab5;  
  
import com.company.lib.\*;  
  
import java.io.\*;  
import java.nio.file.AccessDeniedException;  
import java.util.Map;  
  
import static ifmo.programming.lab5.BuildingChecker.*getCollectionFromJSON*;  
import static ifmo.programming.lab5.RoomFactory.*makeRoomFromJSON*;  
  
public class Main {  
  
 private static final String *FILENAME\_ENV* = "LOADFILE";  
 private static final String *FILE\_FOR\_AUTOSAVE* = "building.json";  
  
 public static void main(String[] args) {

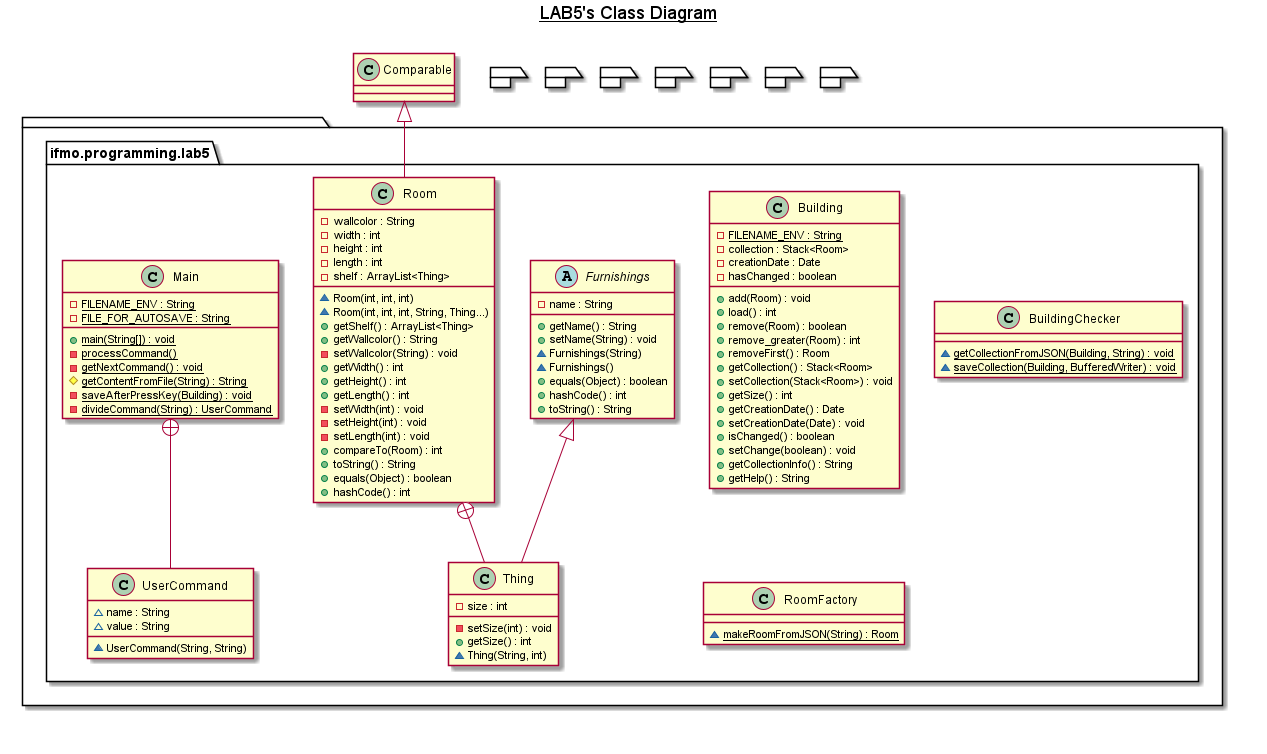
System.*out*.println("Добро пожаловать! \nИ помните, что в конце каждой команды должен стоять символ \";\"." +  
 "\nЧтобы получить справку по командам, введите команду help." );  
 Building building = new Building();  
  
 Map<String, String> env = System.*getenv*();  
 String filename = env.get(*FILENAME\_ENV*);  
  
 if (filename != null && !filename.isEmpty()) {  
 try {  
 String content = *getContentFromFile*(filename);  
 *getCollectionFromJSON*(building, content);  
 System.*out*.println("Состояние загружено из файла");  
 System.*out*.println("Загружено " + building.getSize() + " комнат");  
  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Файл не найден");  
 } catch (AccessDeniedException e) {  
 System.*out*.println("Отсутствует соответствующий доступ к файлу");  
 } catch (IOException e) {  
 System.*out*.println("Ошибка чтения/записи");  
 } catch (Exception e) {  
 System.*out*.println("Не получилось загрузить состояние: " + e.getMessage());  
 }  
 }  
  
 *saveAfterPressKey*(building);  
  
 try {  
 while (true) {  
 System.*out*.print("Введите команду >>> ");  
 String response = *processCommand*(*getNextCommand*(), building);  
 System.*out*.println(response);  
 }  
 } catch (IOException e) {  
 System.*out*.println("Совершенно неожиданная ввода/вывода");  
 }  
 }  
  
 private static String processCommand(String request, Building building) {  
 System.*out*.println(request);  
 if (request == null) {  
 System.*exit*(0);  
 }  
 request = request.trim();  
 UserCommand command = *divideCommand*(request);  
 String result = new String("");  
  
 if (command.name.isEmpty())  
 return "Следует ввести команду";  
  
 switch (command.name) {  
 case "exit":  
 System.*exit*(0);  
  
  
 case "add":  
 try {  
 if (command.value == null)  
 return "Введите данные в формате json после команды add";  
 building.add(*makeRoomFromJSON*(command.value));  
 return "Комната добавлена";  
 } catch (JSONParseException | IllegalArgumentException e) {  
 return e.getMessage();  
 }  
  
 case "remove\_first":  
 if (building.getSize() == 0) {  
 return "В здании нет ни одной комнаты";  
 }  
 return "Удалено комнат: " + building.removeFirst();  
  
 case "remove\_greater":  
 try {  
 Room room = *makeRoomFromJSON*(command.value);  
 return "Удалено " + building.remove\_greater(room) + " комнат";  
 } catch (JSONParseException | IllegalArgumentException e) {  
 return e.getMessage();  
 }  
  
 case "show":  
 if (building.getSize() == 0) {  
 return "В здании нет ни одной комнаты";  
 }  
 result = result.concat("Комнаты: \n");  
 for (Room room : building.getCollection()) {  
 result = result.concat(room.toString() + "\n");  
 }  
 return result;  
  
 case "info":  
 return building.getCollectionInfo();  
  
 case "save":  
 if (command.value == null)  
 return "Укажите имя файла";  
 try {  
 BuildingChecker.*saveCollection*(building, new BufferedWriter(new FileWriter(command.value)));  
 return "Сохранено";  
 } catch (IOException e) {  
 return "Ошибка чтения/записи";  
 }  
  
 case "load":  
 try {  
 int loaded = building.load();  
 return "Загружено " + loaded + " элементов";  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Файл не найден");  
 } catch (IOException e) {  
 System.*out*.println("Ошибка чтения/записи");  
 } catch (Exception e) {  
 System.*out*.println("Не получилось загрузить состояние: " + e.getMessage());  
 }  
  
 case "remove":  
 try {  
 boolean removed = building.remove(*makeRoomFromJSON*(command.value));  
 if (removed) {  
 return "Комната удалена";  
 }  
 return "Нет такой комнаты";  
 } catch (JSONParseException | IllegalArgumentException e) {  
 return e.getMessage();  
 }  
  
 case "help":  
 return building.getHelp();  
  
  
 default:  
 return "Неизвестная команда: " + command.name;  
 }  
 }  
  
 protected static String getContentFromFile(String filename) throws IOException {  
 try (FileReader reader = new FileReader(filename)) {  
 String filecontent = new String("");  
 int running;  
 do {  
 running = reader.read();  
 if (running != -1) {  
 filecontent = filecontent.concat(String.*valueOf*((char)running));  
  
 }  
 } while (running != -1);  
 return filecontent;  
 }  
 }  
  
 private static String getNextCommand() throws IOException {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));

char running;  
 boolean inString = false;  
 StringBuilder builder = new StringBuilder();  
 do {  
 running = (char)reader.read();  
 if (running != ';' || inString) { builder.append(running); }  
 if (running == '"') { inString = !inString;}  
 } while (running != ';' || inString);  
 return builder.toString();  
 }  
  
 private static UserCommand divideCommand(String request) {  
 int spacePosition = request.indexOf(' ');  
 if (spacePosition == -1) {return new UserCommand(request, null);}  
 else {return new UserCommand(request.substring(0, spacePosition), request.substring(spacePosition+1));}  
 }  
  
  
 private static void saveAfterPressKey(Building building) {  
 Runnable saveCode = new Runnable() {  
 @Override  
 public void run() {  
 try {  
 if (building.isChanged()) {  
 System.*out*.println("\nСохраняю...");  
 BuildingChecker.*saveCollection*(building, new BufferedWriter(new FileWriter(*FILE\_FOR\_AUTOSAVE*)));  
 System.*out*.println("Сохранено");  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 };  
 Thread savingThread = new Thread(saveCode);  
 Runtime.*getRuntime*().addShutdownHook(savingThread);  
 }  
  
 private static class UserCommand {  
 String name;  
 String value;  
  
 UserCommand (String name, String value) {  
 this.name = name;  
 this.value = value;  
 }  
 }  
}

**RoomFactory.java**

package ifmo.programming.lab5;  
  
import com.company.lib.\*;  
import java.util.ArrayList;  
  
*/\*\*  
 \* Советский завод по производству комнат для хрущёвок  
 \*/*public class RoomFactory {  
 static Room makeRoomFromJSON(String json) throws JSONParseException, IllegalArgumentException {  
 JSONEntity entity = JSONParser.*parse*(json);  
  
 if (!entity.isObject()) {  
 throw new IllegalArgumentException("Данный json должен быть объектом, но имеет тип " + entity.getType().toString().toLowerCase());  
 }  
  
 JSONObject object = (JSONObject)entity;  
  
 JSONEntity widthEntity = object.getItem("width");  
 JSONEntity heightEntity = object.getItem("height");  
 JSONEntity lengthEntity = object.getItem("length");  
  
 if (widthEntity == null || heightEntity == null || lengthEntity == null) {  
 throw new IllegalArgumentException("width, height и length должны быть обязательно указаны");  
 }  
  
 if (!(widthEntity.isNumber() && heightEntity.isNumber() && lengthEntity.isNumber())) {  
 throw new IllegalArgumentException("width, height и length должны быть числами");  
 }  
  
 int width = (int)((JSONNumber)widthEntity).getValue();  
 int height = (int)((JSONNumber)heightEntity).getValue();  
 int length = (int)((JSONNumber)lengthEntity).getValue();  
  
 String wallcolor = "";  
 JSONEntity wallcolorEntity = object.getItem("wallcolor");  
 if (wallcolorEntity != null) {  
 if (wallcolorEntity.isString()) {  
 wallcolor = ((JSONString)wallcolorEntity).getContent();  
 }  
 else {  
 throw new IllegalArgumentException("wallcolor должен быть строкой, но имеет тип " + wallcolorEntity.getType().toString().toLowerCase());  
 }  
 }  
  
 Room.Thing[] shelfArray = new Room.Thing[0];  
 JSONEntity thingsEntity = object.getItem("shelf");  
  
 if (thingsEntity != null) {  
 if (!thingsEntity.isArray()) {  
 throw new IllegalArgumentException("shelf должен быть массивом, но имеет тип" + thingsEntity.getType().toString().toLowerCase());  
 }  
  
 ArrayList<JSONEntity> entities = ((JSONArray)thingsEntity).getItems();  
 shelfArray = new Room.Thing[entities.size()];  
  
 for (int i = 0; i < entities.size(); i++) {  
 if (!entities.get(i).isObject()) {  
 throw new IllegalArgumentException("Все элементы массива shelf должны быть объектами");  
 }  
 JSONObject thingObject = (JSONObject) entities.get(i);  
 String name = "";  
 int size;  
  
 JSONEntity nameEntity = thingObject.getItem("name");  
 if (nameEntity != null) {  
 if (nameEntity.isString()) {  
 name = ((JSONString) nameEntity).getContent();  
 }  
 else {  
 throw new IllegalArgumentException("Поля name в элементах массива shelf должны быть строками, но одно из них имеет тип " + nameEntity.getType().toString().toLowerCase());  
 }  
 }  
  
 JSONEntity sizeEntity = thingObject.getItem("size");  
 if (sizeEntity == null)  
 throw new IllegalArgumentException("Поле size в элементах массива shelf являются обязательными");  
  
 if (sizeEntity.isNumber()) { size = (int)((JSONNumber) sizeEntity).getValue();}  
 else {  
 throw new IllegalArgumentException("Поля size в элементах массива shelf должны быть числами, но одно из них имеет тип " + sizeEntity.getType().toString().toLowerCase());  
 }  
  
 shelfArray[i] = new Room.Thing(name, size);  
 }  
 }  
  
 return new Room(width, height, length, wallcolor, shelfArray);  
 }  
}

{  
 "created": 1550477201162,  
 "collection":  
 [  
 {  
 "width": 5,  
 "length": 11,  
 "wallcolor": "yellow",  
 "height": 10,  
 "shelf":  
 [  
 {  
 "size": 1,  
 "name": "flower"  
 },  
 {  
 "size": 1,  
 "name": "hat"  
 },  
 {  
 "size": 1,  
 "name": "flower"  
 }  
 ]  
 },  
 {  
 "width": 5,  
 "length": 11,  
 "wallcolor": "white",  
 "height": 10,  
 "shelf":  
 [  
 {  
 "size": 2,  
 "name": "flower"  
 },  
 {  
 "size": 1,  
 "name": "hat"  
 },  
 {  
 "size": 3,  
 "name": "book"  
 }  
 ]  
 },  
 {  
 "width": 5,  
 "length": 1,  
 "wallcolor": "yellow",  
 "height": 10,  
 "shelf":  
 [  
 {  
 "size": 1,  
 "name": "flower"  
 },  
 {  
 "size": 1,  
 "name": "hat"  
 },  
 {  
 "size": 1,  
 "name": "flower"  
 }  
 ]  
 },  
 {  
 "width": 5,  
 "length": 5,  
 "wallcolor": "red",  
 "height": 10,  
 "shelf":  
 [  
 {  
 "size": 1,  
 "name": "book"  
 }  
 ]  
 }  
  
 ]  
}

****

# Вывод:

в ходе многочисленных и продолжительных страданий, связанных с выполнением данной лабораторной работы, я смогла (надеюсь) сделать нечто, отдаленно напоминающее консольное приложение, на всю оставшуюся жизнь запомнила, что представляет из себя формат json, научилась читать и записывать данные в файл, а также узнала некоторые коллекции.