;Министерство транспорта Российской Федерации

Федеральное агентство железнодорожного транспорта

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Дальневосточный государственный университет путей сообщения»

Естественно-научный институт

Кафедра «Вычислительная техника и компьютерная графика»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ JAVA

Расчетно-графическая работа

КР.09.03.03.WEB.14.01.00-БО941ПРИ - ПЗ

Выполнил

Студент группы БО941ПРИ Финк Д.В.

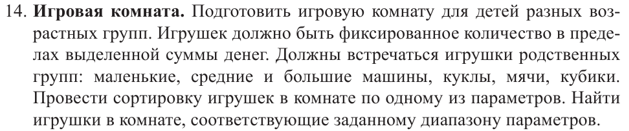
Проверил

преподаватель Сухобок Ю.А.

(подпись, дата)

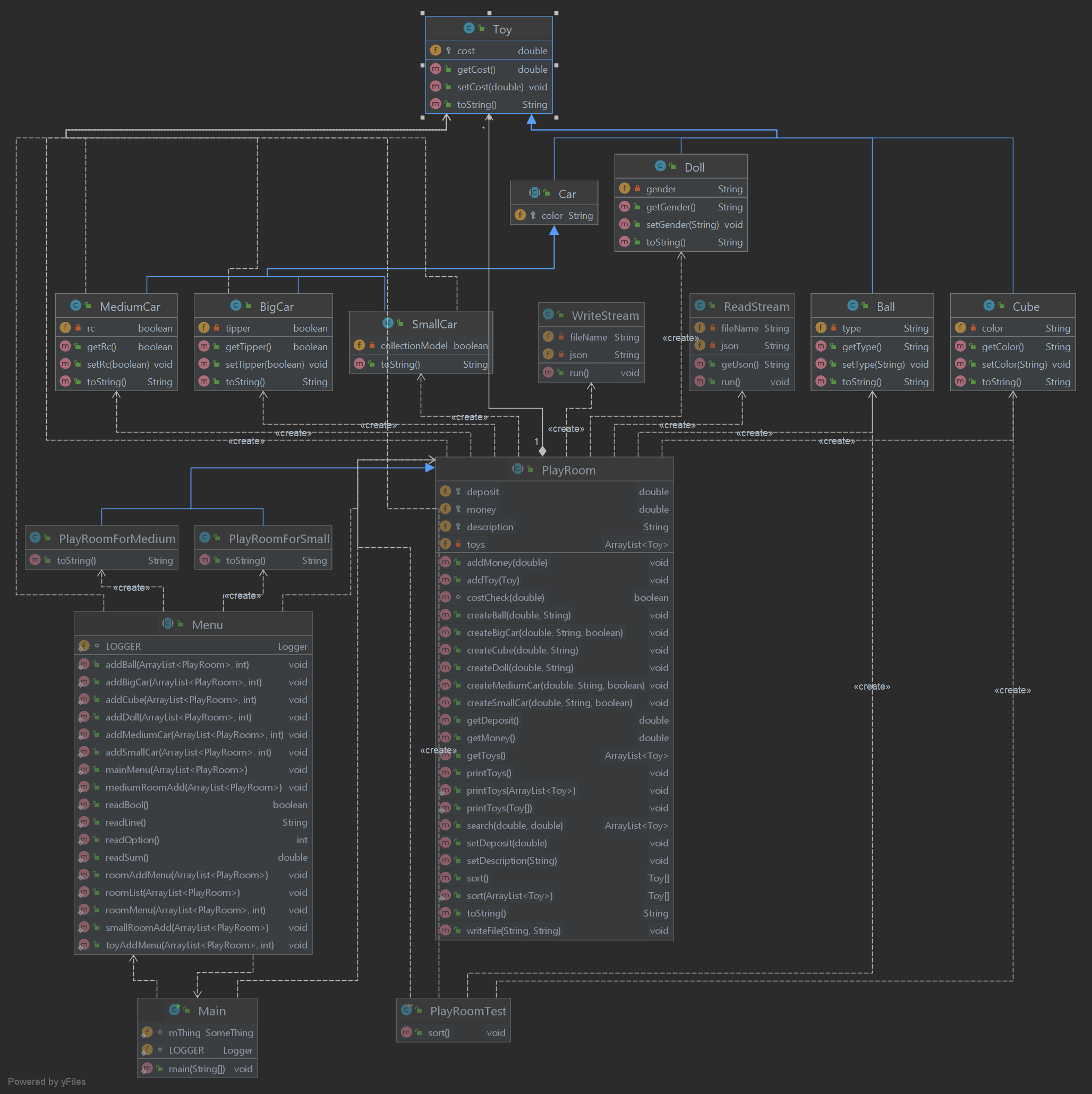
Хабаровск 2021

1. ЗАДАНИЕ



* + 1. Задание

1. UML – Диаграмма



* + 1. UML Диаграмма

1. Код программы
   1. Package Rooms
      1. PlayRoom

package Rooms;  
  
import Toys.\*;  
import stream\_file.\*;  
  
import java.util.\*;  
import java.util.logging.Level;  
  
*/\*\*  
 \* Фабричный паттерн игровой комнаты  
 \*/*public abstract class PlayRoom{  
  
 PlayRoom(double deposit){  
 this.deposit = deposit;  
 }  
  
 protected double deposit; //на какую сумму можно закупать игрушки  
 protected double money; //на какую сумму игрушки уже закуплены  
 protected String description = "/нет описания/";  
  
 private ArrayList<Toy> toys = new ArrayList<>(); //все игрушки хранятся тут  
  
 */\*\*  
 \* сеттеры и геттеры  
 \*/* public ArrayList<Toy> getToys() {  
 return toys;  
 }  
  
 public void setDeposit(double deposit) {  
 this.deposit = deposit;  
 }  
  
 public double getDeposit() {  
 return deposit;  
 }  
  
 public void addToy(Toy toy){  
 toys.add(toy);  
 }  
  
 public double getMoney() {  
 return money;  
 }  
  
 public void addMoney(double cost){  
 this.money += cost;  
 }  
  
 */\*\*  
 \* проверка хватает ли денег на добавляемую игрушку  
 \** ***@param*** *cost на вход получает стоимость игрушки  
 \** ***@return*** *возвращает булево хватает ли средств купить игрушку  
 \*/* boolean costCheck(double cost){  
 return (getMoney() + cost <= getDeposit());  
 }  
  
 */\*\*  
 \* сортировка игрушек по стоимости  
 \** ***@return*** *возвращает отсортированный массив игрушек  
 \*/* public Toy[] sort(){  
  
 Toy[] sorted = new Toy[getToys().size()];  
 getToys().toArray(sorted);  
 Comparator<Toy> comp = Comparator.comparingDouble(Toy::getCost);  
 Arrays.sort(sorted, comp );  
 return sorted;  
 }  
  
 public static Toy[] sort(ArrayList<Toy> ts){  
  
 Toy[] sorted = new Toy[ts.size()];  
 ts.toArray(sorted);  
 Comparator<Toy> comp = Comparator.comparingDouble(Toy::getCost);  
 Arrays.sort(sorted, comp);  
 return sorted;  
 }  
  
 */\*\*  
 \* поиск по диапазону стоимости  
 \** ***@param*** *min  
 \** ***@param*** *max  
 \** ***@return*** *возвращает ArrayList с игрушками, находящимися в диапазоне  
 \*/* public ArrayList<Toy> search(double min, double max){  
  
 ArrayList<Toy> result = new ArrayList<>();  
 ArrayList<Toy> arr = getToys();  
 for (int i=0; i < arr.size(); i++){  
 if ((arr.get(i).getCost() <= max) && (arr.get(i).getCost() >= min)){  
 result.add(arr.get(i));  
 }  
 }  
 return result;  
 }  
  
 */\*\*  
 \* методы создания экземпляров игрушек и добавления их в список  
 \*/* public void createBall(double cost, String type){  
 if (costCheck(cost)) {  
 addMoney(cost);  
 Toy toy = new Ball(cost, type);  
 addToy(toy);  
 }else System.out.println("Недостаточно средств");  
 }  
  
 public void createDoll(double cost, String gender){  
 if (costCheck(cost)){  
 addMoney(cost);  
 Toy toy = new Doll(cost, gender);  
 addToy(toy);  
 }else System.out.println("Недостаточно средств");  
 }  
  
 public void createCube(double cost, String color){  
 if (costCheck(cost)) {  
 addMoney(cost);  
 Toy toy = new Cube(cost, color);  
 addToy(toy);  
 }else System.out.println("Недостаточно средств");  
 }  
  
 public void createSmallCar(double cost, String color, boolean collectionModel){  
 if (costCheck(cost)) {  
 addMoney(cost);  
 Toy toy = new SmallCar(cost, color, collectionModel);  
 addToy(toy);  
 }else System.out.println("Недостаточно средств");  
 }  
  
 public void createMediumCar(double cost, String color, boolean rc){  
 if (costCheck(cost)) {  
 addMoney(cost);  
 Toy toy = new MediumCar(cost, color, rc);  
 addToy(toy);  
 }else System.out.println("Недостаточно средств");  
 }  
  
 public void createBigCar(double cost, String color, boolean tipper){  
 if (costCheck(cost)) {  
 addMoney(cost);  
 Toy toy = new BigCar(cost, color, tipper);  
 addToy(toy);  
 }else System.out.println("Недостаточно средств");  
 }  
  
 */\*\*  
 \* Считать описание комнаты через файл  
 \*/* public void setDescription(String filename){  
 ReadStream reader = new ReadStream(filename);  
 reader.start();  
  
 try {  
 reader.join();  
 }catch (InterruptedException e){  
 System.out.println(e.getMessage());  
 }  
  
 System.out.println("Файл " + filename + " считан");  
 this.description = reader.getJson();  
 }  
  
 */\*\*  
 \* Запись данных в файл  
 \*/* public void writeFile(String filename, String data){  
 WriteStream writer = new WriteStream(filename, data);  
 writer.start();  
 }  
  
 */\*\*  
 \* функции для вывода  
 \*/* public void printToys(){  
 for(Toy r:toys){  
 System.out.println(r.toString());  
 }  
 }  
  
 public static void printToys(Toy[] ts){  
 for(Toy r:ts){  
 System.out.println(r.toString());  
 }  
 }  
  
 public static void printToys(ArrayList<Toy> ts){  
 for(Toy r:ts){  
 System.out.println(r.toString());  
 }  
 }  
  
 @Override  
 public String toString() {  
 return "PlayRoom: " +  
 "deposit " + deposit +  
 ", money " + money +  
 " ,description " + description;  
 }  
}

* + 1. PlayRoomForSmall

package Rooms;  
  
*/\*\*  
 \* Игровая комната для маленьких детей  
 \*/*public class PlayRoomForSmall extends PlayRoom {  
 public PlayRoomForSmall(double deposit) {  
 super(deposit);  
 }  
  
 @Override  
 public String toString() {  
 return "PlayRoomForSmall: " +  
 "deposit=" + deposit +  
 ", money=" + getMoney()+  
 " ,description " + description;  
 }  
}

* + 1. PlayRoomForMedium

package Rooms;

*/\*\*  
 \* Игровая комната для детей средрего возраста  
 \*/*public class PlayRoomForMedium extends PlayRoom{  
 public PlayRoomForMedium(double deposit) {  
 super(deposit);  
 }  
  
 @Override  
 public String toString() {  
 return "PlayRoomForMedium: " +  
 "deposit=" + deposit +  
 ", money=" + getMoney()+  
 " ,description " + description;  
 }  
}

* 1. Package Toys
     1. Ball

package Toys;  
  
public class Ball extends Toy{  
  
 private String type;  
  
  
 public Ball(double cost, String type) {  
 super(cost);  
 this.type = type;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 @Override  
 public String toString() {  
 return "Ball: " +  
 "cost " + cost +  
 ", type " + type;  
 }  
}

* + 1. Doll

package Toys;  
  
public class Doll extends Toy{  
  
 String gender;  
  
 public Doll(double cost, String gender) {  
 super(cost);  
 this.gender = gender;  
 }  
  
 public Doll() {  
  
 }  
  
 public String getGender() {  
 return gender;  
 }  
  
 public void setGender(String gender) {  
 this.gender = gender;  
 }  
  
 @Override  
 public String toString() {  
 return "Doll: " +  
 "cost " + cost +  
 ", gender " + gender;  
 }  
}

* + 1. Cube

package Toys;  
  
public class Cube extends Toy{  
  
 private String color;  
  
 public Cube(double cost, String color) {  
 super(cost);  
 this.color = color;  
 }  
  
  
 public String getColor() {  
 return color;  
 }  
  
 public void setColor(String color) {  
 this.color = color;  
 }  
  
 @Override  
 public String toString() {  
 return "Cube: " +  
 "cost " + cost +  
 ", color " + color;  
 }  
}

* + 1. Car

package Toys;  
  
public abstract class Car extends Toy{  
  
 protected String color;  
  
 public Car(double cost, String color) {  
 super(cost);  
 this.color = color;  
 }  
}

* + 1. SmallCar

package Toys;  
  
public class SmallCar extends Car{  
 private boolean collectionModel; // это коллекционная модель?  
 public SmallCar(double cost, String color, boolean collectionModel) {  
 super(cost, color);  
 this.collectionModel = collectionModel;  
 }  
  
 @Override  
 public String toString() {  
 return "SmallCar: " +  
 "cost " + cost +  
 ", color '" + color + '\'' +  
 ", collectionModel " + collectionModel;  
 }  
}

* + 1. MediumCar

package Toys;  
  
public class MediumCar extends Car{  
  
 private boolean rc; //маишна на радиоуправлении?  
  
 public MediumCar(double cost, String color, boolean rc) {  
 super(cost, color);  
 this.rc = rc;  
 }  
  
 public void setRc(boolean rc) {  
 this.rc = rc;  
 }  
  
 public boolean getRc(){  
 return rc;  
 }  
  
 @Override  
 public String toString() {  
 return "MediumCar: " +  
 "cost " + getCost() +  
 ", color " + color +  
 ", rc " + rc ;  
 }  
}

* + 1. BigCar

package Toys;  
  
public class BigCar extends Car{  
 private boolean tipper; //это самосвал?  
  
 public BigCar(double cost, String color, boolean tipper) {  
 super(cost, color);  
 this.tipper = tipper;  
 }  
  
 public void setTipper(boolean tipper) {  
 this.tipper = tipper;  
 }  
  
 public boolean getTipper(){  
 return tipper;  
 }  
  
 @Override  
 public String toString() {  
 return "BigCar: " +  
 "cost " + cost +  
 ", color " + color +  
 ", tipper " + tipper;  
 }  
}

* + 1. Toy

package Toys;  
  
public class Toy {  
 protected double cost;  
  
 public Toy(){}  
  
 public Toy(double cost){  
 this.cost = cost;  
 }  
  
 public double getCost() {  
 return cost;  
 }  
  
 public void setCost(double cost){  
 this.cost = cost;  
 }  
  
 @Override  
 public String toString() {  
 return "Toy: " +  
 "cost=" + cost +  
 '}';  
 }  
}

* 1. Main

import Rooms.\*;  
  
import java.io.FileInputStream;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
import java.util.ArrayList;  
import java.util.logging.LogManager;  
  
public class Main {  
 */\*\*  
 \* Подключение логгера, который считывает свои параметры из файла (в файле указаны параметры записи в файл)  
 \*/* static Logger *LOGGER*;  
 static {  
 try(FileInputStream ins = new FileInputStream("log.config")){ //полный путь до файла с конфигами  
 LogManager.*getLogManager*().readConfiguration(ins);  
 *LOGGER* = Logger.*getLogger*(Main.class.getName());  
 }catch (Exception ignore){  
 ignore.printStackTrace();  
 }  
 }  
  
 public static void main(String[] args) {  
 *LOGGER*.log(Level.*INFO*,"Начало main, запуск программы]");  
  
 try{  
 ArrayList<PlayRoom> rooms = new ArrayList<>();  
 Menu.*mainMenu*(rooms);  
 }  
 catch (Exception e){  
 *LOGGER*.log(Level.*WARNING*,"Аварийное завершение программы, ошибка " + e);  
 }  
 }  
}

* 1. Menu

import Rooms.\*;  
import Toys.\*;  
  
import java.io.FileInputStream;  
import java.util.ArrayList;  
import java.util.Scanner;  
import java.util.logging.\*;  
  
*/\*\*  
 \* Класс, в котором реализованы методы управления программой через консоль  
 \*/*public abstract class Menu {  
 */\*\*  
 \* Объявление логгера и подрузка его параметров из файла  
 \*/* static Logger LOGGER;  
 static {  
 try(FileInputStream ins = new FileInputStream("log.config")){ //полный путь до файла с конфигами  
 LogManager.getLogManager().readConfiguration(ins);  
 LOGGER = Logger.getLogger(Main.class.getName());  
 }catch (Exception ignore){  
 ignore.printStackTrace();  
 }  
 }  
  
 //главное меню  
 public static void mainMenu(ArrayList<PlayRoom> rooms){  
 System.out.println("Главное меню");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("1. Просмотреть список комнат");  
 System.out.println("2. Выход");  
  
 //переход  
 int option = readOption();  
 switch (option){  
 case 1:  
 roomList(rooms);  
 case 2:  
 LOGGER.log(Level.INFO,"Завершение программы");  
 System.exit(0);  
 default:{  
 System.out.println("Неверное значение. Повторите ввод");  
 mainMenu(rooms);  
 }  
 }  
 }  
  
 //меню со списком комнат  
 public static void roomList(ArrayList<PlayRoom> rooms){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("1. Создать комнату");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("Ваши комнаты:");  
 int n = 1;  
 for(PlayRoom r:rooms){  
 n++;  
 System.out.print(n + ". ");  
 System.out.println(r.toString());  
 }  
 System.out.println("Чтобы войти в комнату введите её номер");  
  
 //переход  
 int option = readOption();  
 switch (option){  
 case 0:  
 mainMenu(rooms);  
 case 1:  
 roomAddMenu(rooms);  
 default:{  
 if (option > 1 && option <= n){  
 roomMenu(rooms, option-2);  
 }  
 }  
 }  
  
  
  
 }  
  
 //меню со списком игрушек  
 public static void roomMenu(ArrayList<PlayRoom> rooms, int n){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("1. Добавить игрушку");  
 System.out.println("2. Изменить сумму на комнату");  
 System.out.println("3. Сортировать по стоимости игрушек");  
 System.out.println("4. Вывести игрушки с интервале стоимости");  
 System.out.println("5. Добавить описание комнаты из файла");  
 System.out.println("6. Записать информацию о классе в файл");  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.out.println("Игрушки в комнате:");  
 rooms.get(n).printToys();  
  
 int option = readOption();  
 switch (option){  
  
 case 0:  
 mainMenu(rooms);  
 break;  
  
 case 1:  
 toyAddMenu(rooms, n);  
 break;  
  
 case 2:  
 System.out.println("Введите новую ссумму");  
 double sum = readSum();  
 rooms.get(n).setDeposit(sum);  
 LOGGER.log(Level.INFO,"Изменение суммы в комнате");  
 mainMenu(rooms);  
 break;  
  
 case 3:  
 Toy[] toysArr = rooms.get(n).sort();  
 PlayRoom.printToys(toysArr);  
 mainMenu(rooms);  
 break;  
  
 case 4:  
 System.out.println("Введите через 'ENTER' минимальное и максимальное значения диапазона");  
 double min = readSum();  
 double max = readSum();  
 ArrayList<Toy> toysArrList = rooms.get(n).search(min, max);  
 PlayRoom.printToys(toysArrList);  
 mainMenu(rooms);  
 break;  
  
 case 5:  
 System.out.println("Введите название файла");  
 String fn = readLine();  
 rooms.get(n).setDescription(fn);  
 LOGGER.log(Level.INFO,"Обновление описания комнаты из файла " + fn);  
 mainMenu(rooms);  
 break;  
  
 case 6:  
 System.out.println("Введите название файла");  
 String fn1 = readLine();  
 rooms.get(n).writeFile(fn1, rooms.get(n).toString());  
 LOGGER.log(Level.INFO,"Информация о комнате была записана в файл " + fn1);  
 mainMenu(rooms);  
 break;  
  
 default:  
 System.out.println("Неверное значение. Повторите ввод");  
 mainMenu(rooms);  
 }  
 }  
  
 //меню выбора типа игрушки  
 public static void toyAddMenu(ArrayList<PlayRoom> rooms, int n){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("1. Мяч");  
 System.out.println("2. Кукла");  
 System.out.println("3. Куб");  
 System.out.println("4. Маленькая машина");  
 System.out.println("5. Средняя машина");  
 System.out.println("6. Большая машина");  
  
 int option = readOption();  
 switch (option){  
 case 0:  
 mainMenu(rooms);  
 break;  
 case 1:  
 addBall(rooms, n);  
 break;  
 case 2:  
 addDoll(rooms, n);  
 break;  
 case 3:  
 addCube(rooms, n);  
 break;  
 case 4:  
 addSmallCar(rooms, n);  
 break;  
 case 5:  
 addMediumCar(rooms, n);  
 break;  
 case 6:  
 addBigCar(rooms, n);  
 break;  
 default:{  
 System.out.println("Неверное значение. Повторите ввод");  
 mainMenu(rooms);  
 }  
 }  
 }  
  
 //меню выбора типа комнаты  
 public static void roomAddMenu(ArrayList<PlayRoom> rooms){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("1. Комната для маленьких");  
 System.out.println("2. Комната для средних");  
  
 int option = readOption();  
 switch (option){  
 case 0:  
 mainMenu(rooms);  
 break;  
 case 1:  
 smallRoomAdd(rooms);  
 break;  
 case 2:  
 mediumRoomAdd(rooms);  
 break;  
 default:{  
 System.out.println("Неверное значение. Повторите ввод");  
 mainMenu(rooms);  
 }  
 }  
 }  
  
 // Меню получения данных для создания новой малой комнаты  
 public static void smallRoomAdd(ArrayList<PlayRoom> rooms){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("Введите сумму, на которую будут закупаться игрушки в комнате");  
  
 double sum = readSum();  
 if (sum == 0) {  
 }  
 else {  
 PlayRoomForSmall room = new PlayRoomForSmall(sum);  
 rooms.add(room);  
 LOGGER.log(Level.INFO,"Создание маленькой комнаты на сумму " + sum);  
 }  
 mainMenu(rooms);  
 }  
  
 // Меню получения данных для создания новой средней комнаты  
 public static void mediumRoomAdd(ArrayList<PlayRoom> rooms){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("Введите сумму, на которую будут закупаться игрушки в комнате");  
  
 double sum = readSum();  
 if (sum == 0) {  
 }  
 else {  
 PlayRoomForMedium room = new PlayRoomForMedium(sum);  
 rooms.add(room);  
 LOGGER.log(Level.INFO,"Создание средней комнаты на сумму " + sum);  
 }  
 mainMenu(rooms);  
 }  
  
 //далее представлены методы, запрашивающие информацию для создания той или иной игрушки и вызывающие соотв. методы из класса PlayRoom  
 public static void addBall(ArrayList<PlayRoom> rooms, int n){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("Введите через 'ENTER' такие параметры как:");  
 System.out.println("1) Стоимость\n2) Тип (прим: баскетбольный)");  
  
 double sum = readSum();  
 if (sum != 0) {  
 String type = readLine();  
 rooms.get(n).createBall(sum, type);  
 LOGGER.log(Level.INFO,"Создание игрушки 'МЯЧ' с параметрами " + sum + ", " + type);  
 }  
 mainMenu(rooms);  
 }  
  
 public static void addDoll(ArrayList<PlayRoom> rooms, int n){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("Введите через 'ENTER' такие параметры как:");  
 System.out.println("1) Стоимость\n2) Пол (прим: женский)");  
  
 double sum = readSum();  
 if (sum != 0) {  
 String gender = readLine();  
 rooms.get(n).createDoll(sum, gender);  
 LOGGER.log(Level.INFO,"Создание игрушки 'КУКЛА' с параметрами " + sum + ", " + gender);  
 }  
 mainMenu(rooms);  
 }  
  
 public static void addCube(ArrayList<PlayRoom> rooms, int n){System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("Введите через 'ENTER' такие параметры как:");  
 System.out.println("1) Стоимость\n2) Цвет");  
  
 double sum = readSum();  
 if (sum != 0) {  
 String color = readLine();  
 rooms.get(n).createCube(sum, color);  
 LOGGER.log(Level.INFO,"Создание игрушки 'КУБ' с параметрами " + sum + ", " + color);  
 }  
 mainMenu(rooms);  
 }  
  
 public static void addSmallCar(ArrayList<PlayRoom> rooms, int n){System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("Введите через 'ENTER' такие параметры как:");  
 System.out.println("1) Стоимость\n2) Цвет\n3) Является ли эта модель коллекционной? (true/false)");  
  
 double sum = readSum();  
 if (sum != 0) {  
 String color = readLine();  
 boolean bool = readBool();  
 rooms.get(n).createSmallCar(sum, color, bool);  
 LOGGER.log(Level.INFO,"Создание игрушки 'МАЛЕНЬКАЯ МАШИНА' с параметрами " + sum + ", " + color + ", " + bool);  
 }  
 mainMenu(rooms);  
 }  
  
 public static void addMediumCar(ArrayList<PlayRoom> rooms, int n){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("Введите через 'ENTER' такие параметры как:");  
 System.out.println("1)Стоимость\n2)Цвет\n3) Является ли эта модель радиоуправляемой? (true/false)");  
  
 double sum = readSum();  
 if (sum != 0) {  
 String color = readLine();  
 boolean bool = readBool();  
 rooms.get(n).createMediumCar(sum, color, bool);  
 LOGGER.log(Level.INFO,"Создание игрушки 'СРЕДНЯЯ МАШИНА' с параметрами " + sum + ", " + color + ", " + bool);  
 }  
 mainMenu(rooms);  
 }  
  
 public static void addBigCar(ArrayList<PlayRoom> rooms, int n){  
 System.out.println("----------------------------------------------------------------");  
 System.out.println("0. Главное меню");  
 System.out.println("Введите через 'ENTER' такие параметры как:");  
 System.out.println("1)Стоимость\n2)Цвет\n3) Является ли эта модель самосвалом? (true/false)");  
  
 double sum = readSum();  
 if (sum != 0) {  
 String color = readLine();  
 boolean bool = readBool();  
 rooms.get(n).createBigCar(sum, color, bool);  
 LOGGER.log(Level.INFO,"Создание игрушки 'БОЛЬШАЯ МАШИНА' с параметрами " + sum + ", " + color + ", " + bool);  
 }  
 mainMenu(rooms);  
 }  
  
 //Считыватели  
 public static int readOption(){  
 Scanner scan = new Scanner(System.in);  
 int option = -1;  
 option = scan.nextInt();  
 return option;  
 }  
  
 public static double readSum(){  
 Scanner scan = new Scanner(System.in);  
 double option = -1;  
 option = scan.nextDouble();  
 return option;  
 }  
  
 public static String readLine(){  
 Scanner scan = new Scanner(System.in);  
 String option = "";  
 option = scan.nextLine();  
 return option;  
 }  
  
 public static boolean readBool(){  
 Scanner scan = new Scanner(System.in);  
 boolean option = false;  
 option = scan.nextBoolean();  
 return option;  
 }  
}

2.5.1 ReadStream

package stream\_file;  
import java.io.\*;  
import java.util.Scanner;  
import java.util.stream.Stream;  
  
public class ReadStream extends Thread {  
  
 private String fileName;  
 private String json;  
 */\*\*  
 \** ***@param*** *fn - имя файла для чтения  
 \*/* public ReadStream(String fn){  
 super("write stream");  
 this.fileName = fn;  
 }  
  
 public void run(){  
 try {  
 FileReader reader = new FileReader(this.fileName);  
 Scanner scan = new Scanner(reader);  
 this.json = scan.nextLine();  
 System.out.println(this.fileName +" was read");  
 reader.close();  
 scan.close();  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
  
 */\*\*  
 \** ***@return*** *json - результат чтения в формате String  
 \*/* public String getJson(){  
 return this.json;  
 }  
}

2.5.2 WriteStream

package stream\_file;  
  
  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class WriteStream extends Thread{  
  
 private String fileName;  
 private String json;  
 */\*\*  
 \** ***@param*** *fn - имя файла для чтения  
 \*/* public WriteStream(String fn, String j){  
 super("write stream");  
 this.fileName = fn;  
 this.json = j;  
 }  
  
 public void run(){  
  
 try {  
 FileWriter writer = new FileWriter(this.fileName, false);  
 writer.write(this.json);  
 writer.close();  
 System.out.println(this.fileName + " has written");  
  
 } catch (IOException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
}

1. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ
2. Выпускная квалификационная работа. Общие требования и правила оформления: методическое пособие / В. Н. Гопкало, О. А. Графский – Хабаровск: Изд-во ДВГУПС, 2014. – 46 с.
3. ГОСТ 2.105-1995 «Общие требования к текстовым документам»