Буреев Фёдор Григорьевич P3107

**Лабораторная работа №3**

**вар. 335998**

1. **Текст задания**

**Описание предметной области, по которой должна быть построена объектная модель:**

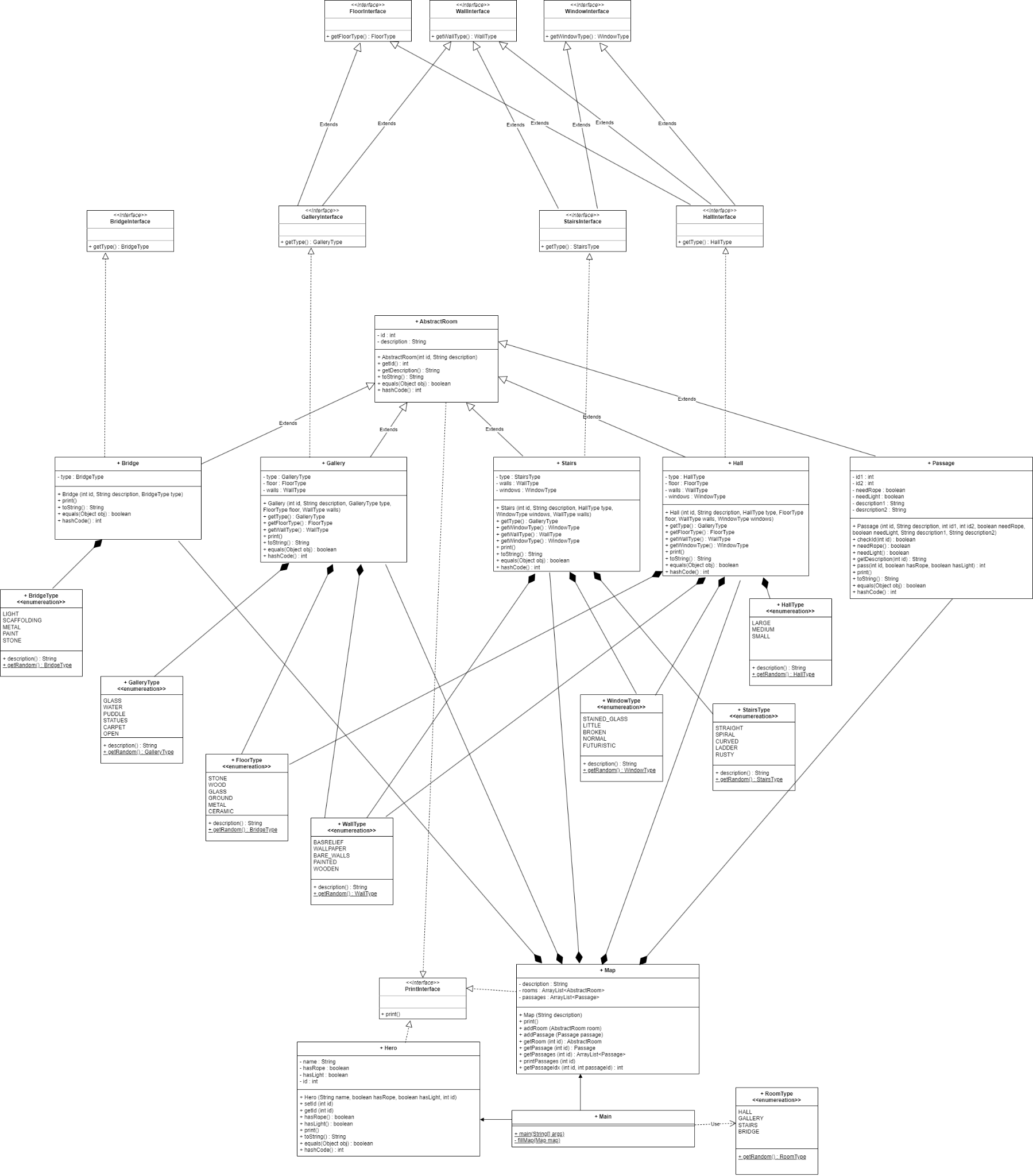
“Наконец мы напали на целый ряд окон -- в венчавшем 'здание громадном пятиугольнике; сквозь них просматривалась просторная, хорошо сохранившаяся комната с каменным полом, однако спуститься туда без веревки не представлялось возможным. Веревка лежала у нас в рюкзаке, но не хотелось возиться без крайней необходимости с двадцатифутовой связкой, особенно в такой разряженной атмосфере, где сердечно-сосудистая система испытывала большие перегрузки. Огромная комната была, скорее всего, главным вестибюлем или залом, и наши электрические фонарики высветили четкие барельефы с поражавшими воображение резными портретами, идущими широкой полосой по стенам зала и отделенными друг от друга традиционным точечным орнаментом. Постаравшись получше запомнить это место, мы решили вернуться сюда в том случае, если не найдем ничего более доступного. В результате мы отыскали проем в стене с арочным перекрытием, шириной шесть и длиной десять футов -- прежде сюда подходил воздушный мостик, соединявший между собой здания. Не знаю, как раньше, но теперь бы он располагался всего в пяти футах над ледяным покровом. Эти сводчатые проходы соответствовали верхним этажам; сохранился здесь, к счастью, и пол. Фасадом это доступное для нас строение было обращено на запад, спускаясь ко . льду террасами. Напротив него, там, где зиял другой арочный проем, возвышалась обшарпанная глухая постройка цилиндрической формы с венчающим ее округлым утолщением -- футах в десяти над единственным отверстием.”

**Программа должна удовлетворять следующим требованиям:**

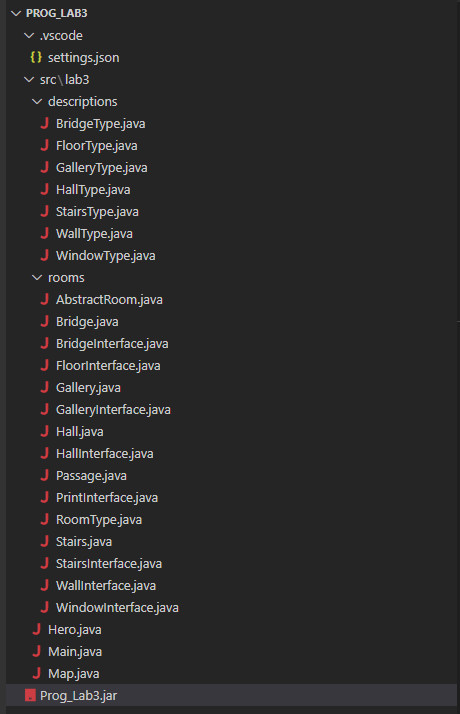
1. Доработанная модель должна соответствовать [принципам SOLID](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)).
2. Программа должна содержать как минимум два интерфейса и один абстрактный класс (номенклатура должна быть согласована с преподавателем).
3. В разработанных классах должны быть переопределены методы equals(), toString() и hashCode().
4. Программа должна содержать как минимум один перечисляемый тип (enum).

**Порядок выполнения работы:**

1. Доработать объектную модель приложения.
2. Перерисовать диаграмму классов в соответствии с внесёнными в модель изменениями.
3. Согласовать с преподавателем изменения, внесённые в модель.
4. Модифицировать программу в соответствии с внесёнными в модель изменениями.
5. **Диаграмма классов объектной модели**



1. **Исходный код программы**



# BridgeType.java

package lab3.descriptions;

import java.util.Random;

public enum BridgeType {

LIGHT,

SCAFFOLDING,

METAL,

PLANT,

STONE;

public String description() {

switch(this) {

case LIGHT:

return "Лёгкий воздушный мостик, казалось, парил в воздухе.";

case SCAFFOLDING:

return "Прочность старых строительных лесов не вызывала никакого доверия.";

case METAL:

return "Тяжёлая металлическая конструкция железнодорожного моста тихо гудела.";

case PLANT:

return "Здесь лианы столь густо переплетали ветви и стволы деревьев, что по ним можно было перебраться на другую сторону провала, как по мосту.";

case STONE:

return "Под небольшим каменном мостиком с четырьмя статуями тихо журчал ручей.";

default:

return "";

}

}

public static BridgeType getRandom() {

Random ran = new Random();

return BridgeType.values()[ran.nextInt(BridgeType.values().length)];

}

}

# FloorType.java

package lab3.descriptions;

import java.util.Random;

public enum FloorType {

STONE,

WOOD,

GLASS,

GROUND,

METAL,

CERAMIC;

public String description() {

switch(this) {

case STONE:

return "Каменный пол был усыпан кусками штукатурки, отвалившейся от потолка.";

case WOOD:

return "Старые пыльные половицы отчаянно скрипели под ногами.";

case GLASS:

return "Сквозь мутный стеклянный пол было очень трудно разглядеть, что находилось вниз.";

case GROUND:

return "Сквозь земляной пол, местами покрытый мхом, пробивались бледные ростки.";

case METAL:

return "Металлический настил гулко звенел под ногами.";

case CERAMIC:

return "Плитки, покрывавшие пол, были в удручающем состоянии.";

default:

return "";

}

}

public static FloorType getRandom() {

Random ran = new Random();

return FloorType.values()[ran.nextInt(FloorType.values().length)];

}

}

# GalleryType.java

package lab3.descriptions;

import java.util.Random;

public enum GalleryType {

GLASS,

WATER,

PUDDLE,

STATUES,

CARPET,

OPEN;

public String description() {

switch(this) {

case GLASS:

return "Стеклянная галерея неплохо сохранилась.";

case WATER:

return "Со сводчатого потолка галереи капала ледяная вода.";

case PUDDLE:

return "Глубокая ледяная лужа перегородила проход, и, чтобы не намочить ботинки, пришлось обойти её по маленькому карнизу возле стены";

case STATUES:

return "Пыльные статуи в металлических доспехах на одинаковых расстояниях стояли по бокам длинного коридора.";

case CARPET:

return "Остатки роскошных ковров на полу и стенах галереи напоминали о былом величии.";

case OPEN:

return "Свежий ветерок гулял по открытой галерее.";

default:

return "";

}

}

public static GalleryType getRandom() {

Random ran = new Random();

return GalleryType.values()[ran.nextInt(GalleryType.values().length)];

}

}

# HallType.java

package lab3.descriptions;

import java.util.Random;

public enum HallType {

LARGE,

MEDIUM,

SMALL;

public String description() {

switch(this) {

case LARGE:

return "Эта помещение было, скорее всего, главным вестибюлем или залом. От одного его размера захватывало дух.";

case MEDIUM:

return "Это была просторная, но не очень хорошо сохранившаяся комната.";

case SMALL:

return "Крохотная тесная комнатушка выглядела довольно неуютно.";

default:

return "";

}

}

public static HallType getRandom() {

Random ran = new Random();

return HallType.values()[ran.nextInt(HallType.values().length)];

}

}

# StairsType.java

package lab3.descriptions;

import java.util.Random;

public enum StairsType {

STRAIGHT,

SPIRAL,

CURVED,

LADDER,

RUSTY;

public String description() {

switch(this) {

case STRAIGHT:

return "Широкий пролёт лестницы с пыльными металлическими периллами с изображениями змей уходил в бесконечность.";

case SPIRAL:

return "Винтовая лестница без перил круто уходила вверх и вниз.";

case CURVED:

return "Большую часть помещения занимала изогнутая лестница с широкими ступенями.";

case LADDER:

return "Длиннная приставная лестница уходила куда-то вверх. Вниз шла такая же лестница в противоположном углу помещения.";

case RUSTY:

return "Тяжёлая проржавевшая конструкция, напоминающая мост, выглядела очень странно в этом помещении.";

default:

return "";

}

}

public static StairsType getRandom() {

Random ran = new Random();

return StairsType.values()[ran.nextInt(StairsType.values().length)];

}

}

# WallType.java

package lab3.descriptions;

import java.util.Random;

public enum WallType {

BASRELIEF,

WALLPAPER,

BARE\_WALLS,

PAINTED,

WOODEN;

public String description() {

switch(this) {

case BASRELIEF:

return "Четкие барельефы с поражавшими воображение резными портретами, шли широкой полосой по стенам зала, отделённые друг от друга традиционным точечным орнаментом.";

case WALLPAPER:

return "Выцветшие от времени обои скучно свисали пыльными лоскутами с ободранных стен.";

case BARE\_WALLS:

return "На голых стенах не осталось почти ничего кроме практически обвалившейся штукатурки.";

case PAINTED:

return "Узоры практически выцветшей краски, когда-то давно бывшие многоцветным восточным орнаментом, покрывали стены от пола до потолка.";

case WOODEN:

return "Частично прогнившие деревянные панели на стенах местами разошлись, открывая неприглядного вида каменную кладку.";

default:

return "";

}

}

public static WallType getRandom() {

Random ran = new Random();

return WallType.values()[ran.nextInt(WallType.values().length)];

}

}

# 

# WindowType.java

package lab3.descriptions;

import java.util.Random;

public enum WindowType {

STAINED\_GLASS,

LITTLE,

BROKEN,

NORMAL,

FUTURISTIC;

public String description() {

switch(this) {

case STAINED\_GLASS:

return "Прекрасный витраж с изображениями морской жизни, когда-то украшавший большое окно, изрядно выцвел и покрылся многовековой пылью.";

case LITTLE:

return "Крошечное оконце на потолке практически не пропускало света.";

case BROKEN:

return "Остатки оконной рамы, выбитые, казалось бы, ударом ужасающей силы, сиротливо торчали в оконном проёме.";

case NORMAL:

return "Оконное стекло здесь неплохо сохранилось.";

case FUTURISTIC:

return "Полупрозрачное перламутровое окно овальной формы испускало загадочный потусторонний свет.";

default:

return "";

}

}

public static WindowType getRandom() {

Random ran = new Random();

return WindowType.values()[ran.nextInt(WindowType.values().length)];

}

}

# AbstractRoom.java

package lab3.rooms;

public abstract class AbstractRoom implements PrintInterface {

private int id;

private String description;

public AbstractRoom(int id, String description) {

this.id = id;

this.description = description;

}

public int getId() {

return id;

}

public String getDescription() {

return description;

}

@Override

public String toString() {

return description;

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if ((obj == null) || (getClass() != obj.getClass())) {

return false;

}

AbstractRoom room = (AbstractRoom) obj;

return id == room.getId();

}

@Override

public int hashCode() {

return id;

}

}

# Bridge.java

package lab3.rooms;

import lab3.descriptions.BridgeType;

public class Bridge extends AbstractRoom implements BridgeInterface{

private BridgeType type;

public Bridge(int id, String description, BridgeType type) {

super(id, description);

this.type = type;

}

public BridgeType getType() {

return type;

}

@Override

public void print() {

System.out.println(getDescription());

System.out.println(type.description());

}

@Override

public String toString() {

return getDescription() + " " + type.description();

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if ((obj == null) || (getClass() != obj.getClass())) {

return false;

}

Bridge room = (Bridge) obj;

return getId() == room.getId();

}

@Override

public int hashCode() {

return getId();

}

}

# BridgeInterface.java

package lab3.rooms;

import lab3.descriptions.BridgeType;

public interface BridgeInterface {

public BridgeType getType();

}

# FloorInterface.java

package lab3.rooms;

import lab3.descriptions.FloorType;

public interface FloorInterface {

public FloorType getFloorType();

}

# Gallery.java

package lab3.rooms;

import lab3.descriptions.GalleryType;

import lab3.descriptions.WallType;

import lab3.descriptions.FloorType;

public class Gallery extends AbstractRoom implements GalleryInterface {

private GalleryType type;

private FloorType floor;

private WallType walls;

public Gallery(int id, String description, GalleryType type, FloorType floor, WallType walls) {

super(id, description);

this.type = type;

this.floor = floor;

this.walls = walls;

}

public GalleryType getType() {

return type;

}

public FloorType getFloorType() {

return floor;

}

public WallType getWallType() {

return walls;

}

@Override

public void print() {

System.out.println(getDescription());

System.out.println(type.description());

System.out.println(floor.description());

System.out.println(walls.description());

}

@Override

public String toString() {

return getDescription() + " " + type.description() + " " + floor.description() + " " + walls.description();

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if ((obj == null) || (getClass() != obj.getClass())) {

return false;

}

Gallery room = (Gallery) obj;

return getId() == room.getId();

}

@Override

public int hashCode() {

return getId();

}

}

# GalleryInterface.java

package lab3.rooms;

import lab3.descriptions.GalleryType;

public interface GalleryInterface extends FloorInterface, WallInterface {

public GalleryType getType();

}

# Hall.java

package lab3.rooms;

import lab3.descriptions.HallType;

import lab3.descriptions.FloorType;

import lab3.descriptions.WallType;

import lab3.descriptions.WindowType;

public class Hall extends AbstractRoom implements HallInterface{

private HallType type;

private FloorType floor;

private WallType walls;

private WindowType windows;

public Hall(int id, String description, HallType type, FloorType floor, WallType walls, WindowType windows) {

super(id, description);

this.type = type;

this.floor = floor;

this.walls = walls;

this.windows = windows;

}

public HallType getType() {

return type;

}

public FloorType getFloorType() {

return floor;

}

public WallType getWallType() {

return walls;

}

public WindowType getWindowType() {

return windows;

}

@Override

public void print() {

System.out.println(getDescription());

System.out.println(type.description());

System.out.println(floor.description());

System.out.println(walls.description());

System.out.println(windows.description());

}

@Override

public String toString() {

return getDescription() + " " + type.description() + " " + floor.description() + " " + walls.description() + " " + windows.description();

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if ((obj == null) || (getClass() != obj.getClass())) {

return false;

}

Hall room = (Hall) obj;

return getId() == room.getId();

}

@Override

public int hashCode() {

return getId();

}

}

# HallInterface.java

package lab3.rooms;

import lab3.descriptions.HallType;

public interface HallInterface extends FloorInterface, WallInterface, WindowInterface {

public HallType getType();

}

# Passage.java

package lab3.rooms;

public class Passage extends AbstractRoom {

private int id1;

private int id2;

private boolean needRope;

private boolean needLight;

private String description1;

private String description2;

public Passage(int id, String description, int id1, int id2, boolean needRope, boolean needLight, String description1, String description2) {

super(id, description);

this.id1 = id1;

this.id2 = id2;

this.needRope = needRope;

this.needLight = needLight;

this.description1 = description1;

this.description2 = description2;

}

public boolean checkId(int id) {

return ((id == id1) || (id == id2));

}

public boolean needRope() {

return needRope;

}

public boolean needLight() {

return needLight;

}

public String getDescription(int id) {

if(id == id1) {

return description1;

} else if(id == id2) {

return description2;

}

return "";

}

public int pass(int id, boolean hasRope, boolean hasLight) {

if(((needRope && hasRope) || !needRope) && ((needLight && hasLight) || !needLight)) {

if(id == id1) {

return id2;

} else if (id == id2) {

return id1;

}

} else {

System.out.println("Невозможно пройти : " + ((needRope && !hasRope) ? " Нужна верёвка! " : "") + ((needLight && !hasLight) ? " Нужен фонарик! " : ""));

}

return id;

}

public void print() {

System.out.println(getDescription());

System.out.println(id1 + ": " + getDescription(id1));

System.out.println(id2 + ": " + getDescription(id2));

System.out.println((needRope ? "Нужна верёвка." : "") + (needLight ? "Нужен фонарик." : ""));

}

@Override

public String toString() {

return getDescription() + " " + id1 + ": [" + getDescription(id1) + "] " + id2 + ": [" + getDescription(id2) + "] " + (needRope ? "Нужна верёвка." : "") + (needLight ? "Нужен фонарик." : "");

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if ((obj == null) || (getClass() != obj.getClass())) {

return false;

}

Passage passage = (Passage) obj;

return getId() == passage.getId();

}

@Override

public int hashCode() {

return getId();

}

}

# PrintInterface.java

package lab3.rooms;

public interface PrintInterface {

public void print();

}

# RoomType.java

package lab3.rooms;

import java.util.Random;

public enum RoomType {

HALL,

GALLERY,

STAIRS,

BRIDGE;

public static RoomType getRandom() {

Random ran = new Random();

return RoomType.values()[ran.nextInt(RoomType.values().length)];

}

}

# Stairs.java

package lab3.rooms;

import lab3.descriptions.WallType;

import lab3.descriptions.WindowType;

import lab3.descriptions.StairsType;

public class Stairs extends AbstractRoom implements StairsInterface {

private StairsType type;

private WindowType windows;

private WallType walls;

public Stairs(int id, String description, StairsType type, WindowType windows, WallType walls) {

super(id, description);

this.type = type;

this.windows = windows;

this.walls = walls;

}

public StairsType getType() {

return type;

}

public WindowType getWindowType() {

return windows;

}

public WallType getWallType() {

return walls;

}

@Override

public void print() {

System.out.println(getDescription());

System.out.println(type.description());

System.out.println(walls.description());

System.out.println(windows.description());

}

@Override

public String toString() {

return getDescription() + " " + type.description() + " " + walls.description() + " " + windows.description();

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if ((obj == null) || (getClass() != obj.getClass())) {

return false;

}

Stairs room = (Stairs) obj;

return getId() == room.getId();

}

@Override

public int hashCode() {

return getId();

}

}

# StairsInterface.java

package lab3.rooms;

import lab3.descriptions.StairsType;

public interface StairsInterface extends WallInterface, WindowInterface {

public StairsType getType();

}

# WallInterface.java

package lab3.rooms;

import lab3.descriptions.WallType;

public interface WallInterface {

public WallType getWallType();

}

# WindowInterface.java

package lab3.rooms;

import lab3.descriptions.WindowType;

public interface WindowInterface {

public WindowType getWindowType();

}

# Hero.java

package lab3;

import lab3.rooms.PrintInterface;

public class Hero implements PrintInterface {

private String name;

private boolean hasRope;

private boolean hasLight;

private int id;

public Hero(String name, boolean hasRope, boolean hasLight, int id) {

this.name = name;

this.hasRope = hasRope;

this.hasLight = hasLight;

this.id = id;

}

public void setId(int id) {

this.id = id;

}

public int getId() {

return id;

}

public boolean hasRope() {

return hasRope;

}

public boolean hasLight() {

return hasLight;

}

public void print() {

System.out.println("Имя: " + name);

System.out.println("Вещи:" + (hasRope ? " Верёвка" : "") + (hasLight ? " Фонарик" : ""));

System.out.println("Положение: " + id);

}

@Override

public String toString() {

return name + " [" + (hasRope ? " Верёвка" : "") + (hasLight ? " Фонарик" : "") + "]" + "Положение: " + id;

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if ((obj == null) || (getClass() != obj.getClass())) {

return false;

}

Hero hero = (Hero) obj;

return name.equals(hero.name);

}

@Override

public int hashCode() {

return name.hashCode();

}

}

# Main.java

package lab3;

import lab3.descriptions.\*;

import lab3.rooms.\*;

public class Main {

public static void main(String[] args) {

Map map = new Map("Старый город.");

fillMap(map);

Hero hero = new Hero("A", true, true, 1);

hero.print();

System.out.println();

//Проходы, по которым нужно пройти

int moves[] = {101, 102, 102, 103, 104};

for(int i = 0; i < moves.length; i++) {

int id = hero.getId();

map.getRoom(id).print();

map.printPassages(id);

System.out.println();

System.out.println("### Делаем переход из комнаты " + id + " по проходу " + map.getPassageIdx(id, moves[i]));

System.out.println();

hero.setId(map.getPassage(moves[i]).pass(id, hero.hasRope(), hero.hasLight()));

hero.print();

System.out.println();

}

map.getRoom(hero.getId()).print();

System.out.println();

System.out.println("И, наконец, мы поняли, что пришли, куда хотели. Мы увидели ларёк с пирожками, и радости нашей не было предела!");

}

private static void fillMap(Map map) {

String descr, descr1, descr2;

descr = "Наконец мы напали на целый ряд окон -- в венчавшем 'здание громадном пятиугольнике.";

map.addRoom(new Gallery(1, descr, GalleryType.GLASS, FloorType.GROUND, WallType.WOODEN));

descr = "Мы перешли в гигантский зал.";

map.addRoom(new Hall(2, descr, HallType.LARGE, FloorType.STONE, WallType.BASRELIEF, WindowType.STAINED\_GLASS));

descr = "Затем мы дошли до странного моста. Однако, мы решили не идти по нему, и вернулись обратно в зал.";

map.addRoom(new Bridge(3, descr, BridgeType.getRandom()));

descr = "Пойдя по этому проходу, мы пришли к лестнице.";

map.addRoom(new Stairs(4, descr, StairsType.getRandom(), WindowType.getRandom(), WallType.getRandom()));

descr = "С большим трудом мы добрались до конца прохода и вышли в следующее помещение. Наши ноги начинали уставать.";

switch(RoomType.getRandom()) {

case HALL :

map.addRoom(new Hall(5, descr, HallType.getRandom(), FloorType.getRandom(), WallType.getRandom(), WindowType.getRandom()));

case GALLERY :

map.addRoom(new Gallery(5, descr, GalleryType.getRandom(), FloorType.getRandom(), WallType.getRandom()));

case STAIRS :

map.addRoom(new Stairs(5, descr, StairsType.getRandom(), WindowType.getRandom(), WallType.getRandom()));

case BRIDGE :

map.addRoom(new Bridge(5, descr, BridgeType.getRandom()));

}

descr1 = "Cквозь ряд окон просматривалась просторная, хорошо сохранившаяся комната, однако спуститься туда без веревки не представлялось возможным. Постаравшись получше запомнить это место, мы решили вернуться сюда в том случае, если не найдем ничего более доступного.";

descr2 = "Возле самого потолка мы увидели целый ряд окон в венчавшем здание гигантском пятиугольнике, но подняться туда без приставной лестницы или верёвки невозможно.";

map.addPassage(new Passage(100, "Проход №100", 1, 2, true, false, descr1, descr2));

descr1 = "В результате мы отыскали проем в стене с арочным перекрытием, шириной шесть и длиной десять футов -- прежде сюда подходил воздушный мостик, соединявший между собой здания. Не знаю, как раньше, но теперь бы он располагался всего в пяти футах над ледяным покровом. Эти сводчатые проходы соответствовали верхним этажам; сохранился здесь, к счастью, и пол.";

descr2 = "Тёмное отверстие прохода между двумя деревянными панелями вело обратно к стеклянной галерее, откуда мы только что пришли.";

map.addPassage(new Passage(101, "Проход №101", 1, 2, false, false, descr1, descr2));

descr1 = "Узкая тёмная нора в северо-западном углу тоже куда-то вела.";

descr2 = "Узкая тёмная нора вела обратно в гигантский зал.";

map.addPassage(new Passage(102, "Проход №102", 2, 3, false, false, descr1, descr2));

descr1 = "Небольшая неприметная дверца в углу зала была чуть приоткрыта. За ней октрывался тёмный прямой коридор.";

descr2 = "Тёмный прямой коридор вёл обратно к гигантскому залу.";

map.addPassage(new Passage(103, "Проход №103", 2, 4, false, false, descr1, descr2));

descr1 = "Запах горячих пирожков доносился из провала в полу. Мы очень не хотели развязывать верёвку и включать фонарики с едва работающими батареями, но запах был слишком силён.";

descr2 = "Наевшись пирожков, мы поняли, что нам совсем не хочется возвращаться.";

map.addPassage(new Passage(104, "Проход №104", 4, 5, true, true, descr1, descr2));

}

}

# Map.java

package lab3;

import java.util.ArrayList;

import lab3.rooms.AbstractRoom;

import lab3.rooms.Passage;

import lab3.rooms.PrintInterface;

public class Map implements PrintInterface{

private String description;

private ArrayList<AbstractRoom> rooms = new ArrayList<AbstractRoom>();

private ArrayList<Passage> passages = new ArrayList<Passage>();

public Map(String description) {

this.description = description;

}

public void print() {

System.out.println(description);

}

public void addRoom(AbstractRoom room) {

rooms.add(room);

}

public void addPassage(Passage passage) {

passages.add(passage);

}

public AbstractRoom getRoom(int id) {

for(int i = 0; i < rooms.size(); i++) {

if(id == rooms.get(i).getId()) {

return rooms.get(i);

}

}

return null;

}

public Passage getPassage(int id) {

for(int i = 0; i < passages.size(); i++) {

if(id == passages.get(i).getId()) {

return passages.get(i);

}

}

return null;

}

public ArrayList<Passage> getPassages(int id) {

ArrayList<Passage> result = new ArrayList<Passage>();

for(int i = 0; i < passages.size(); i++) {

if(passages.get(i).checkId(id)) {

result.add(passages.get(i));

}

}

return result;

}

public void printPassages(int id) {

int n = 1;

System.out.println();

for(int i = 0; i < passages.size(); i++) {

if(passages.get(i).checkId(id)) {

System.out.println("Проход " + n + ": " + passages.get(i).getDescription(id));

n++;

}

}

}

public int getPassageIdx(int id, int passageId) {

int n = 1;

for(int i = 0; i < passages.size(); i++) {

if(passages.get(i).checkId(id)) {

if(passages.get(i).getId() == passageId) {

return n;

}

n++;

}

}

return 0;

}

}

1. **Результат работы программы**

Имя: A

Вещи: Верёвка Фонарик

Положение: 1

Наконец мы напали на целый ряд окон -- в венчавшем 'здание громадном пятиугольнике.

Стеклянная галерея неплохо сохранилась.

Сквозь земляной пол, местами покрытый мхом, пробивались бледные ростки.

Частично прогнившие деревянные панели на стенах местами разошлись, открывая неприглядного вида каменную кладку.

Проход 1: Cквозь ряд окон просматривалась просторная, хорошо сохранившаяся комната, однако спуститься туда без веревки не представлялось возможным. Постаравшись получше запомнить это место, мы решили вернуться сюда в том случае, если не найдем ничего более доступного.

Проход 2: В результате мы отыскали проем в стене с арочным перекрытием, шириной шесть и длиной десять футов -- прежде сюда подходил воздушный мостик, соединявший между собой здания. Не знаю, как раньше, но теперь бы он располагался всего в пяти футах над ледяным покровом. Эти сводчатые проходы соответствовали верхним этажам; сохранился здесь, к счастью, и пол.

### Делаем переход из комнаты 1 по проходу 2

Имя: A

Вещи: Верёвка Фонарик

Положение: 2

Мы перешли в гигантский зал.

Эта помещение было, скорее всего, главным вестибюлем или залом. От одного его размера захватывало дух.

Каменный пол был усыпан кусками штукатурки, отвалившейся от потолка.

Четкие барельефы с поражавшими воображение резными портретами, шли широкой полосой по стенам зала, отделённые друг от друга традиционным точечным орнаментом.

Прекрасный витраж с изображениями морской жизни, когда-то украшавший большое окно, изрядно выцвел и покрылся многовековой пылью.

Проход 1: Возле самого потолка мы увидели целый ряд окон в венчавшем здание гигантском пятиугольнике, но подняться туда без приставной лестницы или верёвки невозможно.

Проход 2: Тёмное отверстие прохода между двумя деревянными панелями вело обратно к стеклянной галерее, откуда мы только что пришли.

Проход 3: Узкая тёмная нора в северо-западном углу тоже куда-то вела.

Проход 4: Небольшая неприметная дверца в углу зала была чуть приоткрыта. За ней октрывался тёмный прямой коридор.

### Делаем переход из комнаты 2 по проходу 3

Имя: A

Вещи: Верёвка Фонарик

Положение: 3

Затем мы дошли до странного моста. Однако, мы решили не идти по нему, и вернулись обратно в зал.

Здесь лианы столь густо переплетали ветви и стволы деревьев, что по ним можно было перебраться на другую сторону провала, как по мосту.

Проход 1: Узкая тёмная нора вела обратно в гигантский зал.

### Делаем переход из комнаты 3 по проходу 1

Имя: A

Вещи: Верёвка Фонарик

Положение: 2

Мы перешли в гигантский зал.

Эта помещение было, скорее всего, главным вестибюлем или залом. От одного его размера захватывало дух.

Каменный пол был усыпан кусками штукатурки, отвалившейся от потолка.

Четкие барельефы с поражавшими воображение резными портретами, шли широкой полосой по стенам зала, отделённые друг от друга традиционным точечным орнаментом.

Прекрасный витраж с изображениями морской жизни, когда-то украшавший большое окно, изрядно выцвел и покрылся многовековой пылью.

Проход 1: Возле самого потолка мы увидели целый ряд окон в венчавшем здание гигантском пятиугольнике, но подняться туда без приставной лестницы или верёвки невозможно.

Проход 2: Тёмное отверстие прохода между двумя деревянными панелями вело обратно к стеклянной галерее, откуда мы только что пришли.

Проход 3: Узкая тёмная нора в северо-западном углу тоже куда-то вела.

Проход 4: Небольшая неприметная дверца в углу зала была чуть приоткрыта. За ней октрывался тёмный прямой коридор.

### Делаем переход из комнаты 2 по проходу 4

Имя: A

Вещи: Верёвка Фонарик

Положение: 4

Пойдя по этому проходу, мы пришли к лестнице.

Большую часть помещения занимала изогнутая лестница с широкими ступенями.

Узоры практически выцветшей краски, когда-то давно бывшие многоцветным восточным орнаментом, покрывали стены от пола до потолка.

Полупрозрачное перламутровое окно овальной формы испускало загадочный потусторонний свет.

Проход 1: Тёмный прямой коридор вёл обратно к гигантскому залу.

Проход 2: Запах горячих пирожков доносился из провала в полу. Мы очень не хотели развязывать верёвку и включать фонарики с едва работающими батареями, но запах был слишком силён.

### Делаем переход из комнаты 4 по проходу 2

Имя: A

Вещи: Верёвка Фонарик

Положение: 5

С большим трудом мы добрались до конца прохода и вышли в следующее помещение. Наши ноги начинали уставать.

Это была просторная, но не очень хорошо сохранившаяся комната.

Плитки, покрывавшие пол, были в удручающем состоянии.

Выцветшие от времени обои скучно свисали пыльными лоскутами с ободранных стен.

Оконное стекло здесь неплохо сохранилось.

И, наконец, мы поняли, что пришли, куда хотели. Мы увидели ларёк с пирожками, и радости нашей не было предела!

1. **Выводы по работе**

Во время выполнения данной лабораторной работы я научился составлять объектную модель для предметной области, создавать интерфейсы и абстрактные классы, переопределять методы классов и использовать enum в java, а также генерировать случайные значения в диапазоне значений enum с помощью библиотеки Random. Ещё я научился рисовать uml-диаграмму классов.

Программа, написанная для данной лабораторной работы, создает набор помещений, связанных переходами, используя частично случайные, частично указанные описания, и обеспечивает проход персонажа по заданному маршруту. На каждом переходе или в комнате выводится окружение: типы проходов и описание текущей комнаты.