Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет

им. И.И. Ползунова»

Факультет (институт) Информационных технологий

Кафедра Прикладная математик

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.М. Старолетов

(подпись преподавателя) (инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

Отчет

по лабораторной (практической) работе № 4

по дисциплине Архитектурное проектирование и паттерны программирования

(наименование дисциплины)

ЛР 09.03.04.10.000 ОТ

(обозначение документа)

Студенты группы ПИ-42 Е.А. Колесников

(инициалы, фамилии)

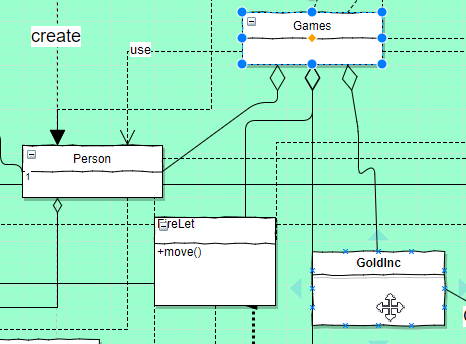
Преподаватель Доцент к. канд. физ.-мат. наук С.М.Старолетов

(должность, ученое звание) (инициалы, фамилия)

Барнаул 2017

**Задание:** реализовать паттерны: Мост, Фасад, Информационный эксперт, Приспособленец

Фасад:



**КОД:**

public interface IGameFasad {  
 public void update();  
}

public class Games implements IGameFasad {  
 public static final int *SIZE\_ROCK* = 50;  
 private Person person;  
 private FireLet let\_fire;  
 private CurrentLog currentLog;  
 private boolean GameOver;  
 private Texture backgroundScreen;  
 private FlyweightFactory flyweightFactory;  
 private int count\_rock;  
 private GoldInc goldInc;  
 private Playing play;  
 private ConcretePerson concretePerson;  
 private float getLastDropTime;  
 private long longtimeGame;  
 private int timeGame;  
 public Texture getBackgroundScreen() {  
 return backgroundScreen;  
 }  
 public void setBackgroundScreen(Texture backgroundScreen) {  
 this.backgroundScreen = backgroundScreen;  
 }  
  
 public Array<IGold> getArray()  
 {  
 return goldInc.getGold();  
 }  
 public Games() {  
 goldInc=new GoldInc();  
 goldInc.createGold();  
 getLastDropTime=0;  
 timeGame=0;  
 longtimeGame=TimeUtils.*nanoTime*();  
 concretePerson=new ConcretePerson();  
 count\_rock=0;  
 GameOver=false;  
 flyweightFactory= new FlyweightFactory();  
 Rectangle bucket = new Rectangle();  
 //центрируем персонажа по горизонтали  
 bucket.x = *VIEWPORT\_WIDTH* / 2 - *WIDTH\_PLAYER* / 2;  
 bucket.y = *STARTER\_POSITION*;  
 bucket.width = *WIDTH\_PLAYER*;  
 bucket.height = *HEIGHT\_PLAYER*;  
 Texture PlayerImage = new Texture(Gdx.*files*.internal("player2/1.png"));  
 backgroundScreen=new Texture(Gdx.*files*.internal("background/back.jpg"));  
  
 Vector3 touchPos = new Vector3(bucket.x, bucket.y, 0);  
  
 //Инициализируем игрока  
 person = new Person(touchPos, PlayerImage, bucket, "player2");  
  
 person.setjumpAction( new CanJump());  
 person.setHEIGHT\_JUMP(person.getHEIGHT\_JUMP() + 10);  
 currentLog=new CurrentLog(this.person);  
 CurrentToFileLog currentToFileLog = new CurrentToFileLog(this.person);  
 // инициализируем горизонтальное препятствие  
 let\_fire = concretePerson.getFireLet();  
 //инициализируем вертикальное препятствие+  
 Shooting shooting=new Shooting(this.getPerson());  
 play=new Playing();  
 }  
  
 public int getCount\_rock() {  
 return count\_rock;  
 }  
  
 public FlyweightFactory getFlyweightFactory() {  
  
 return flyweightFactory;  
 }  
  
 public HashMap<Integer, RockLet> getHashMap() {  
 return flyweightFactory.getRocklet();  
 }  
  
 public int updateTime()  
 {  
 if(TimeUtils.*nanoTime*()-longtimeGame>1000000000) {  
 timeGame++;  
 longtimeGame=TimeUtils.*nanoTime*();  
 }  
 return timeGame;  
 }  
  
 public int getTimeGame() {  
 return timeGame;  
 }  
  
 @Override  
 public void update() {  
  
 let\_fire.Move();  
 person.update(Gdx.*graphics*.getDeltaTime());  
 goldInc.update(person);  
 if(MathUtils.*random*(0,200)==0)  
 {  
 goldInc.createGold();  
 }  
 if (person.getPlayer().overlaps(let\_fire.getHorizontal\_rectangle())|| person.getHP()<=0) {  
// neytronGame.setScreen(new MainMenuScreen(neytronGame));  
 GameOver=true;  
 }  
 if (Gdx.*input*.isKeyPressed(com.badlogic.gdx.Input.Keys.*LEFT*))  
 {  
 person.setOrientation(*ORIENTATION\_LEFT*);  
 person.perfomWalk();  
 }  
 //Метод Gdx.graphics.getDeltaTime() возвращает время, прошедшее между последним и текущим кадром в секундах.  
 if (Gdx.*input*.isKeyPressed(com.badlogic.gdx.Input.Keys.*RIGHT*))  
 {  
 person.setOrientation(*ORIENTATION\_RIGHT*);  
 person.perfomWalk();  
 }  
  
 // play.execute(person);  
  
 //Движение персонажа по карте(вверх-ввниз)  
 if (Gdx.*input*.isKeyPressed(com.badlogic.gdx.Input.Keys.*T*))  
 person.perfomTeleport();  
 if (Gdx.*input*.isKeyPressed(Input.Keys.*CONTROL\_LEFT*))  
 person.doAction();  
 if (Gdx.*input*.isKeyPressed(com.badlogic.gdx.Input.Keys.*UP*))  
 person.PerfomJump(TimeUtils.*nanoTime*());  
  
 if (TimeUtils.*nanoTime*() - getLastDropTime > 1000000000) {  
 if(count\_rock<=*SIZE\_ROCK*) {  
  
 SinglePoolLet.*getInstance*().createNewObject();  
 getLastDropTime = TimeUtils.*nanoTime*();  
 count\_rock++;  
 }  
 }  
 if (Gdx.*input*.isKeyPressed(com.badlogic.gdx.Input.Keys.*RIGHT*) && Gdx.*input*.isKeyPressed(com.badlogic.gdx.Input.Keys.*LEFT*))  
 person.setPlayerImage(new Texture(Gdx.*files*.internal(*PLAYER*+"/1.png")));  
 SinglePoolLet.*getInstance*().Move();  
 ArrayList<Shot> shots=person.getCommand().getShots();  
 for (int i=0;i<shots.size();i++)  
 {  
 shots.get(i).update();  
 if(shots.get(i).getRectangle\_shot().x>*VIEWPORT\_WIDTH*&&shots.get(i).getRectangle\_shot().x<0)  
 shots.remove(i);  
 if(shots.get(i).getRectangle\_shot().overlaps(let\_fire.getHorizontal\_rectangle()))  
 {  
 shots.remove(i);  
 let\_fire = concretePerson.getFireLet();  
 }  
 }  
 }  
  
 public boolean isGameOver() {  
 return GameOver;  
 }  
  
 public void setGameOver(boolean gameOver) {  
 GameOver = gameOver;  
 }  
  
 public Person getPerson() {  
 return person;  
 }  
  
 public void setPerson(Person person) {  
 this.person = person;  
 currentLog=new CurrentLog(this.person);  
 CurrentToFileLog currentToFileLog = new CurrentToFileLog(this.person);  
 Shooting shooting=new Shooting(this.getPerson());  
 }  
  
 public FireLet getLet\_fire() {  
 return let\_fire;  
 }  
  
 public void setLet\_fire(FireLet let\_fire) {  
 this.let\_fire = let\_fire;  
 }  
  
  
  
}

Приспособленец:

public class FlyweightFactory {  
 public static final int *TIME\_DROP* = 10000000;  
 private long lastDropTime;  
 private HashMap<Integer, RockLet> rocklet = new HashMap<Integer, RockLet>();  
  
 public FlyweightFactory() {  
 lastDropTime=0;  
 }  
  
 public long getLastDropTime() {  
 return lastDropTime;  
 }  
  
 public void setLastDropTime(long lastDropTime) {  
 this.lastDropTime = lastDropTime;  
 }  
  
 public RockLet getRockLet(int rockCode) {  
  
 RockLet rockLet = rocklet.get(new Integer(rockCode));  
 if (rockLet == null) {  
 switch (MathUtils.*random*(0, 1)) {  
 case 0:  
 rockLet = new Rock();  
 lastDropTime=TimeUtils.*nanoTime*();  
 break;  
 case 1:  
 rockLet = new IceRock();  
 lastDropTime=TimeUtils.*nanoTime*();  
 break;  
  
 }  
 rocklet.put(rockCode,rockLet);  
 }  
 return rockLet;  
 }  
  
 public HashMap<Integer, RockLet> getRocklet() {  
 return rocklet;  
 }  
  
 public void move(Person e)  
 {  
 for (Map.Entry<Integer, RockLet> pair: rocklet.entrySet())  
 {  
 pair.getValue().fall();  
 if(pair.getValue().getRock\_let().y<0) {  
 if (TimeUtils.*nanoTime*() - lastDropTime > *TIME\_DROP*) {  
 pair.getValue().create\_rock();  
 lastDropTime=TimeUtils.*nanoTime*();  
 }  
 }  
 if(e.getPlayer().overlaps(pair.getValue().getRock\_let()))  
 {  
 e.setHP(e.getHP()-pair.getValue().getDamage());  
 pair.getValue().create\_rock();  
 }  
  
 }  
  
 }  
}

**Тесты:**