|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**Дисциплина «Программирование на языке Джава»**

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОМУ ЗАНЯТИЮ №6**

Выполнил студент группы ИВБО-07-19 Гудаев И.И.

Принял Степанин П.С.

Практические работы выполнены «\_\_\_»\_\_\_\_\_\_\_2020г.

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2020г.

Отметка о выполнении

**Москва – 2020 г.**

## **Задание**

1.Создайте в draw.io UML диаграмму, а затем напишите по ней реализацию.

2.Добавьте в проект ранее разработанный класс прямоугольник (Rectangle). На основе него напишите новый класс MovableRectangle (движущийся прямоугольник). Ваш класс должен реализовывать интерфейс Movable

3.Прямоугольник можно также представить как две движущиеся точки MovablePoints (представляющих верхняя левая и нижняя правая точки) и реализующие интерфейс Movable.

4.Убедитесь, что две точки имеет одну и ту же скорость (нужно добавить метод, который бы это проверял).

## **Ход Работы**

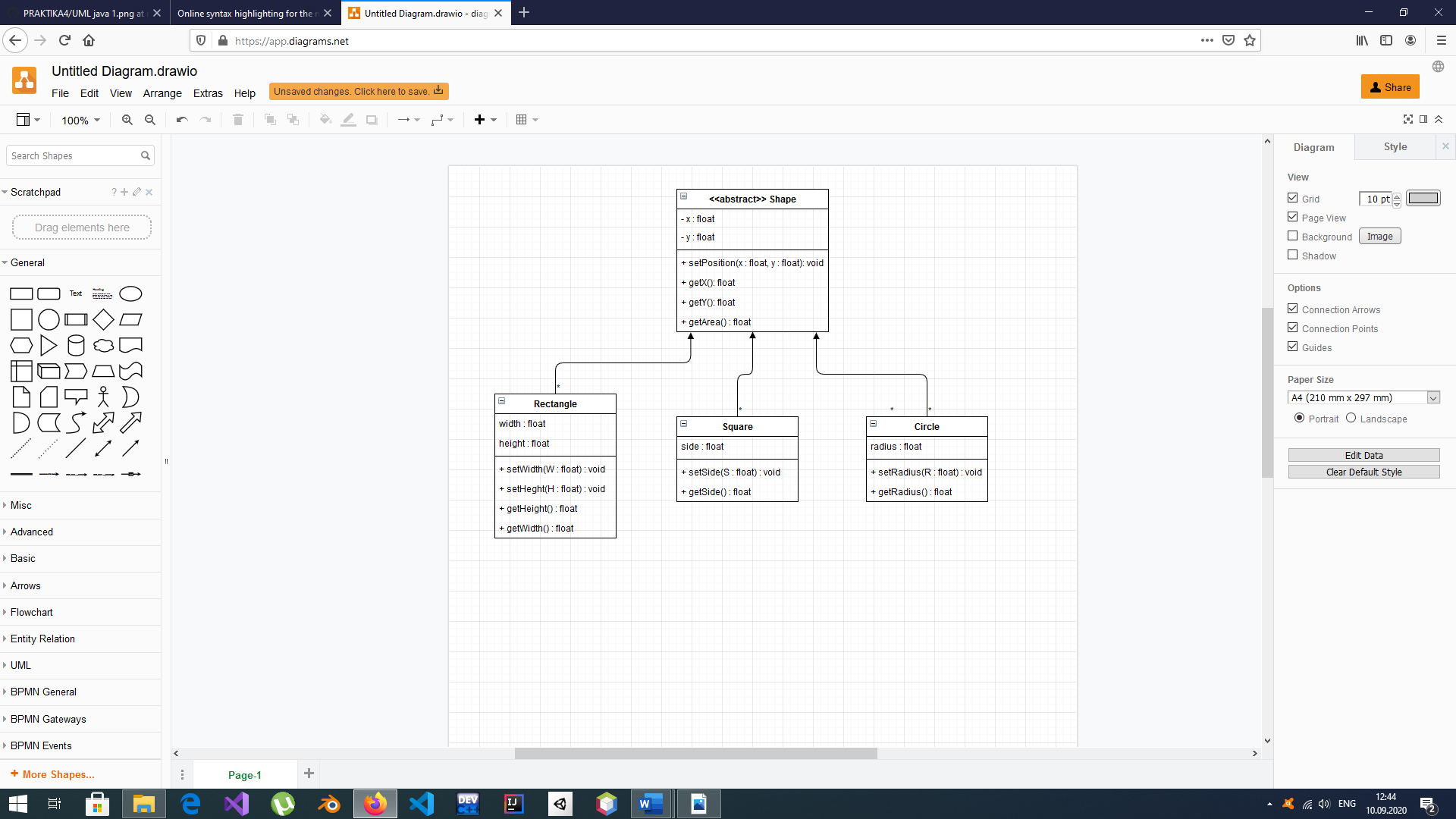


Рисунок 1. UML диаграмма 1 задания

На рисунке 1 представлена диаграмма 1 задания. На ней изображены 4 класса: абстрактный класс Shape и его 3 класса-наследника – Rectangle, Circle и Square. Каждый из них имеет свои методы и поля, но, кроме этого, они реализуют абстрактные методы класса Shape. На листингах 1-5 представлена реализация данной диаграммы.

**package** com.company;

**public** **abstract** **class** Shape {

**private** int x, y;

**public** void setPosition(int X, int Y)

{

x = X;

y = Y;

}

**public** int getX() {

**return** x;

}

**public** int getY() {

**return** y;

}

**public** **abstract** float getArea();

}

Листинг 1. Задание 1 (класс Shape)

**package** com.company;

**public** **class** Rectangle **extends** Shape{

**private** float width, height;

**public** float getWidth() {

**return** width;

}

**public** void setWidth(float width) {

**this**.width = width;

}

**public** float getHeight() {

**return** height;

}

**public** void setHeight(float height) {

**this**.height = height;

}

**@Override**

**public** float getArea() {

**return** **this**.height \* **this**.width;

}

}

return y;

}

**public** **abstract** float getArea();

}

Листинг 2. Задание 1 (класс Rectangle)

**package** com.company;

**public** **class** Square **extends** Shape {

**private** float side;

**public** float getSide() {

**return** side;

}

**public** void setSide(float side) {

**this**.side = side;

}

**@Override**

**public** float getArea() {

**return** side \* side;

}

}

Листинг 3. Задание 1 (класс Square)

**package** com.company;

**public** **class** Circle **extends** Shape {

**private** float radius;

**public** float getRadius() {

**return** radius;

}

**public** void setRadius(float radius) {

**this**.radius = radius;

}

**@Override**

**public** float getArea() {

**return** 3.14f \* **this**.radius \* **this**.radius;

}

}

Листинг 4. Задание 1 (класс Circle)

**package** com.company;

**public** **class** Main {

**public** **static** void main(**String**[] args) {

// write your code here

Square sq = **new** Square();

Circle cr = **new** Circle();

cr.setRadius(3);

**System**.out.print(cr.getArea());

}

}

Листинг 5. Задание 1 (класс Main)

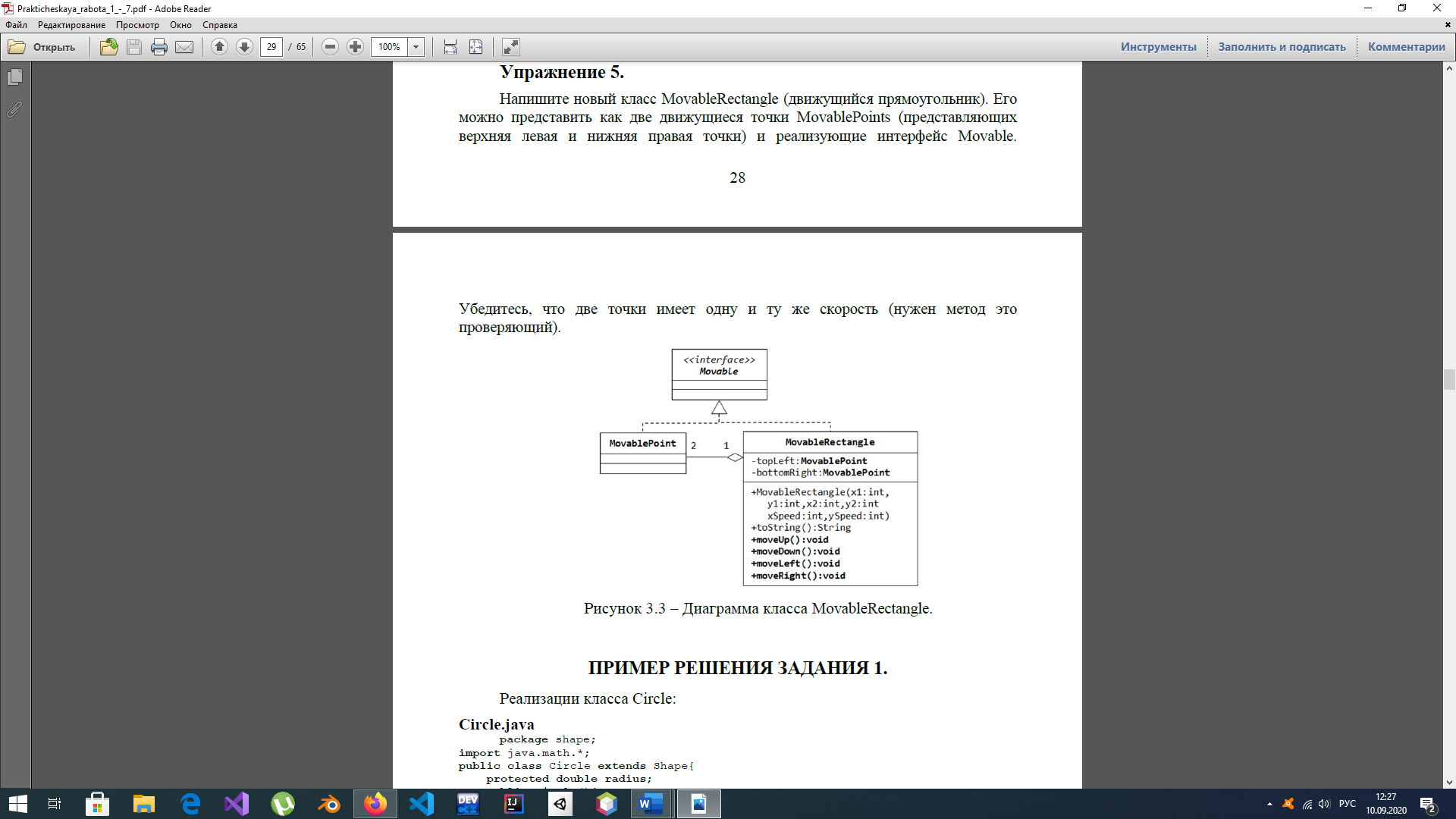


Рисунок 2. UML диаграмма 2 задания

На UML диаграмме из рисунка 2 представлен интерфейс Movable из пр. работы №5, от которого имплементируются классы MovablePoint (из пр. раб. №5) и MovableRectangle. Оба класса связаны агрегацией. Ниже, на листингах 6-9, представлена реализация данной диаграммы.

В ходе выполнения работы были получены следующие исходные коды:

**package** com.company;

**public** **interface** Movable {

void moveUp();

void moveDown();

void moveLeft();

void moveRight();

}

Листинг 6. Задание 2 (интерфейс Movable)

**package** com.company;

**import** java.time.Clock;

**public** **class** MovablePoint **implements** Movable{

int x, y, xSpeed, ySpeed;

**public** MovablePoint(int x, int y, int xSpeed, int ySpeed) {

**this**.x = x;

**this**.y = y;

**this**.xSpeed = xSpeed;

**this**.ySpeed = ySpeed;

}

Листинг 7.1. Задание 2 (класс MovablePoint)

**@Override**

**public** void moveDown() {

**for**(float i = 0; i < 10f; i+= 0.01f)

{

x += 0;

y += ySpeed;

}

}

**@Override**

**public** void moveLeft() {

**for**(float i = 0; i < 10f; i+= 0.01f)

{

x -= xSpeed;

y += 0;

}

}

**@Override**

**public** void moveRight() {

**for**(float i = 0; i < 10f; i+= 0.01f)

{

x += xSpeed;

y += 0;

}

}

**@Override**

**public** void moveUp() {

**for**(float i = 0; i < 10f; i+= 0.01f)

{

x += 0;

y -= ySpeed;

}

}

}

Листинг 7.2. Задание 2 (класс MovablePoint)

**package** com.company;

**public** **class** MovableRectangle **implements** Movable{

**private** MovablePoint topLeft;

**private** MovablePoint bottomRight;

MovableRectangle(int x1, int y1, int x2, int y2, int xSpeed, int ySpeed)

{

topLeft.x = x1; topLeft.y = y1; topLeft.xSpeed = xSpeed; topLeft.ySpeed = ySpeed;

bottomRight.x = x2; bottomRight.y = y2; bottomRight.xSpeed = xSpeed; bottomRight.ySpeed = ySpeed;

}

**public** boolean checkSpeeds()

{

**if**(topLeft.xSpeed == bottomRight.xSpeed && topLeft.ySpeed == bottomRight.ySpeed)

{

**return** **true**;

}

**else**

{

**return** **false**;

}

}

Листинг 8.1. Задание 2 (класс MovableRectangle)

**@Override**

**public** void moveDown() {

**if**(checkSpeeds())

{

**for**(float i = 0; i < 10f; i+= 0.01f)

{

topLeft.x += 0;

topLeft.y += topLeft.ySpeed;

bottomRight.x += 0;

bottomRight.y += topLeft.ySpeed;

}

}

}

**@Override**

**public** void moveLeft() {

**if**(checkSpeeds())

{

**for**(float i = 0; i < 10f; i+= 0.01f)

{

topLeft.x -= topLeft.xSpeed;

topLeft.y += 0;

bottomRight.x += 0;

bottomRight.y += bottomRight.ySpeed;

}

}

}

**@Override**

**public** void moveRight() {

**if**(checkSpeeds())

{

**for**(float i = 0; i < 10f; i+= 0.01f)

{

topLeft.x += topLeft.xSpeed;

topLeft.y += 0;

bottomRight.x += 0;

bottomRight.y += bottomRight.ySpeed;

}

}

}

**@Override**

**public** void moveUp() {

**if**(checkSpeeds())

{

**for**(float i = 0; i < 10f; i+= 0.01f)

{

topLeft.x += 0;

topLeft.y -= topLeft.ySpeed;

bottomRight.x += 0;

bottomRight.y += bottomRight.ySpeed;

}

}

}

}

Листинг 8.2. Задание 2 (класс MovableRectangle)

**package** com.company;

**public** **class** Main {

**public** **static** void main(**String**[] args) {

// write your code here

MovableRectangle mrect = **new** MovableRectangle(1, 1, 2, 2, 1, 1);

**if**(mrect.checkSpeeds())

{

**System**.out.print("OK");

}

**else**

{

**System**.out.print("Not OK");

}

}

}

Листинг 9. Задание 2 (класс Main)

## **Вывод**

В ходе выполнения этой работы мы создали свою UML диаграмму на сайте draw.io и составили по ней программу, а так же написали другую программу, в которой классы реализуют интерфейс Movable аналогично тому, как мы делали в практической работе №5.