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 №3

Выполнил студент группы ИНБО-02-20 Деревянкин Н.А.

Принял Степанов П.В.

Практическая работа выполнена «___»_____2021 г.

«_____» «___»_____2021 г.

Отметка о выполнении

Москва – 2021 г.

СОДЕРЖАНИЕ

Цель работы 3
Задание 3
Выполнение работы 4
Код выполненной работы 7
Вывод 14

Цель работы

Освоить на практике работу с абстрактными классами и наследованием на Java

Задание

1) Абстрактный суперкласс Shape и его подклассы Задание перепишите суперкласс Shape и его подклассы так как это представлено на диаграмме Circle, Rectangle and Square.

В этом задании, класс Shape определяется как абстрактный класс , который содержит:

Два protected (защищенных) переменных color(String) и filled(boolean). Защищенные переменные могут быть доступны в подклассах и классах в одном пакете . Они обозначаются со знаком '#' в диаграмме классов в нотации языка UML. Методы геттеры и сеттеры для всех переменных экземпляра класса, и метод toString () .

Два абстрактных метода getArea() и getPerimeter() выделены курсивом в диаграмме класса).

В подклассах Circle(круг) и Rectangle(прямоугольник) должны переопределяться абстрактные методы getArea() и getPerimeter(), чтобы обеспечить их надлежащее выполнение для конкретных экземпляров типа подкласс.

Также необходимо для каждого подкласса переопределить toString() .

Вам нужно написать тестовый класс, чтобы самостоятельно это проверить, необходимо объяснить полученные результаты и связать их с понятием ООП - полиморфизм. Некоторые объявления могут вызвать ошибки компиляции. Объясните полученные ошибки, если таковые имеются.

2) Вам нужно написать два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable.

Выполнение работы

1) Первоначально я создал абстрактный класс Shape, в котором описал все по диаграмме, в том числе и абстрактные методы. Далее я создал каждый класс, соблюдая наследование, в которых заполнил абстрактные методы в зависимости от фигуры класса. И перешел к написанию теста.
После создания теста (взятого из задания) я увидел, что некоторые тесты имеют ошибки.

```
System.out.println(s1.isFilled());
System.out.println((s1).getRadius());
```

Ошибка связана с тем, что в классе Shape нет метода .getRadius(), а в классе Circle есть. Это и является причиной возникновения ошибки при компиляции. Чтобы избежать ошибки, требуется привести тип(кастануть), s1 -> ((Circle)s1).

2) Второе задание было выполнено по аналогии с первым. Но теперь наши классы связаны другим способом, теперь я пользовался имплементацией и агрегацией.

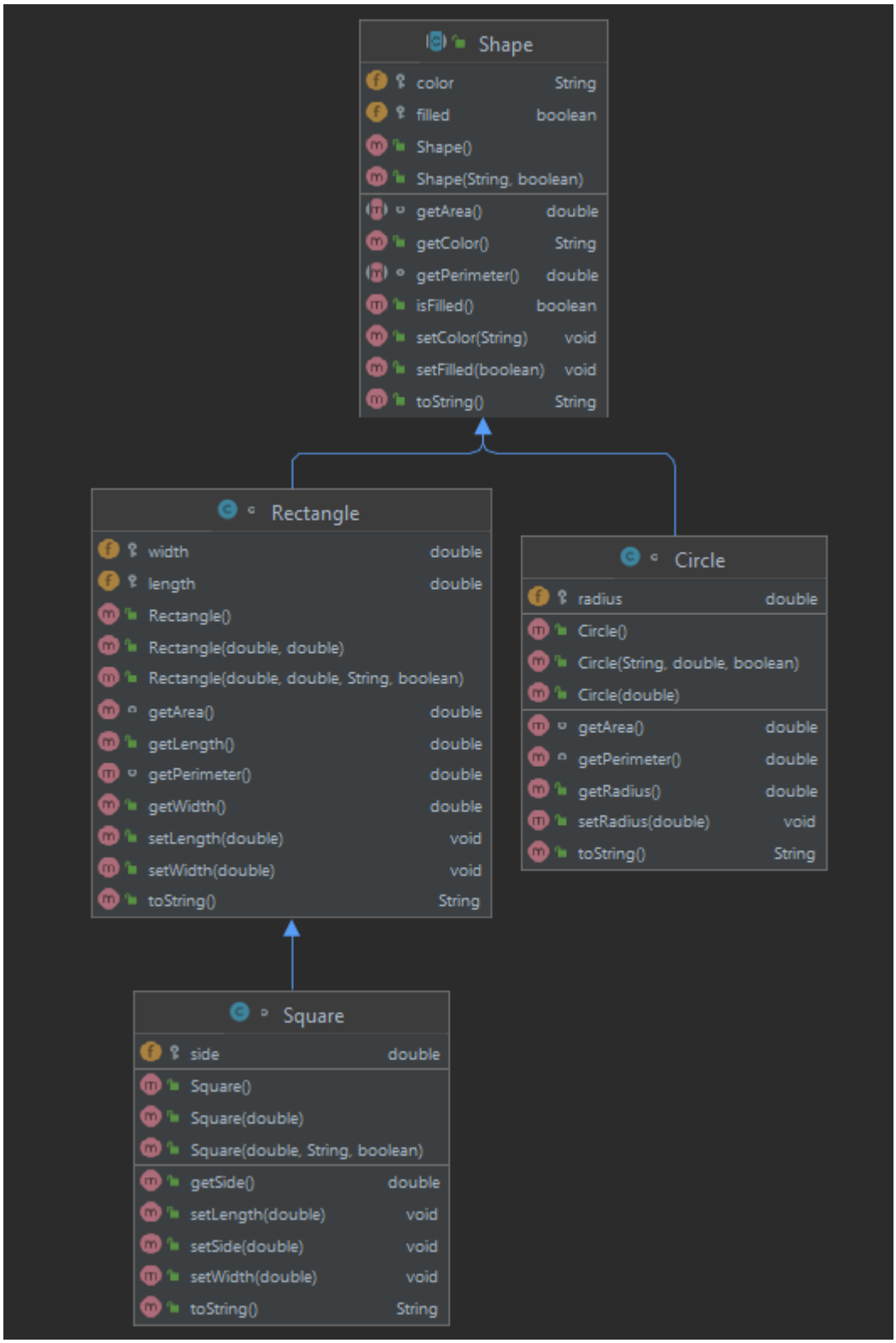


Рисунок 1 – UML Диаграмма (1)

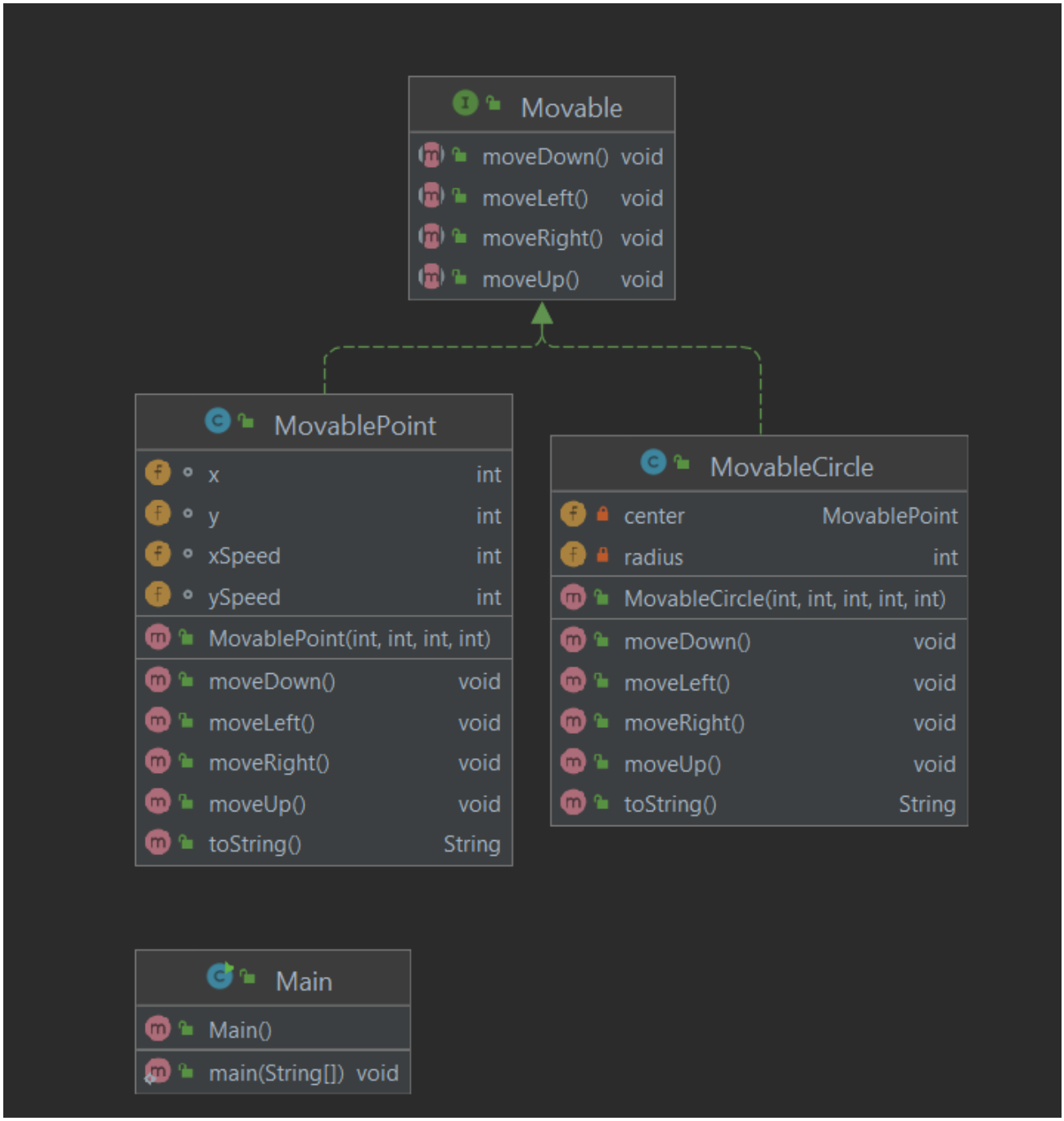


Рисунок 2 – UML Диаграмма (2)

Код выполненной работы

Здесь в нескольких скриншотах можно увидеть, как выглядит код полученного задания и его вывод.

```
public abstract class Shape {
    protected String color;
    protected boolean filled;

    public Shape(){
        color = "Blue";
        filled = false;
    }

    public Shape(String color, boolean filled){
        this.color = "Blue";
        this.filled = false;
    }

    public String getColor() { return color; }

    public void setColor(String color) { this.color = color; }

    public boolean isFilled() { return filled; }

    public void setFilled(boolean filled) { this.filled = filled; }

    abstract double getArea();
    abstract double getPerimeter();
```

Рисунок 3 – Абстрактный класс Shape (1)

```java
class Circle extends Shape {
    protected double radius;

    public Circle(){
        this.color = "Yellow";
        this.filled = false;
        radius = 2;
    }

    public Circle(double radius){
        this.color = "Yellow";
        this.filled = false;
        this.radius = radius;
    }

    public Circle(String color, double radius, boolean filled){
        this.color = color;
        this.filled = filled;
        this.radius = radius;
    }

    public double getRadius() { return radius; }

    public void setRadius(double radius) { this.radius = radius; }
```

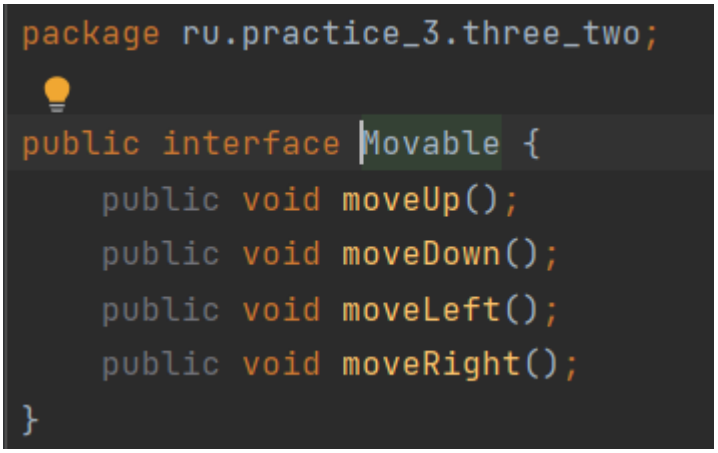
Рисунок 4 – Класс Circle, который наследует Shape (1)

```
class Rectangle extends Shape {
    protected double width;
    protected double length;

    public Rectangle() {
        this.color = "Red";
        this.filled = true;
        this.width = 4;
        this.length = 5;
    }

    public Rectangle(double width, double length) {
        this.color = "Red";
        this.filled = true;
        this.width = width;
        this.length = length;
    }

    public Rectangle(double width, double length, String color, boolean filled) {
        this.length = length;
        this.width = width;
        this.color = color;
        this.filled = filled;
    }

    public double getWidth() { return width; }

    public void setWidth(double width) { this.width = width; }

    public double getLength() { return length; }
```

Рисунок 5 – Класс Rectangle, который наследует Shape (1)

```
class Square extends Rectangle {
    protected double side;

    public Square() {
        this.color = "Red";
        this.filled = true;
        this.side = 5;
    }

    public Square(double side) {
        this.color = "Red";
        this.filled = true;
        this.side = side;
    }

    public Square(double side, String color, boolean filled) {
        this.color = color;
        this.filled = filled;
        this.side = side;
    }

    public double getSide() { return side; }

    public void setSide(double side) {
        this.side = side;
    }

    public void setLength(double length) { this.length = length; }

    public void setWidth(double width) { this.width = width; }
```

Рисунок 6 – Класс Square, который наследует Rectangle (1)



```
package ru.practice_3.three_two;

public interface Movable {
    public void moveUp();
    public void moveDown();
    public void moveLeft();
    public void moveRight();
}
```

Рисунок 7 – Интерфейс Movable (2)

```java
public class MovableCircle implements Movable{
    private MovablePoint center;
    private int radius;


    public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {
        this.center = new MovablePoint(x,y,xSpeed, ySpeed);
        this.radius = radius;
    }

    @Override
    public void moveUp() {
        center.moveUp();
    }

    @Override
    public void moveDown() {
        center.moveDown();
    }

    @Override
    public void moveLeft() {
        center.moveLeft();
    }

    @Override
    public void moveRight() {
        center.moveRight();
    }
```

Рисунок 8 – Класс MovableCirlce, который имплементирует Movable и агрегирует класс MovablePoint (2)

```java
public class MovablePoint implements Movable {
    int x;
    int y;
    int xSpeed;
    int ySpeed;

    public MovablePoint(int x, int y, int xSpeed, int ySpeed) {
        this.x = x;
        this.y = y;
        this.xSpeed = xSpeed;
        this.ySpeed = ySpeed;
    }

    @Override
    public void moveUp() {
        y+=ySpeed;
    }

    @Override
    public void moveDown() {
        y-=ySpeed;
    }

    @Override
    public void moveLeft() {
        x-=xSpeed;
    }

    @Override
    public void moveRight() {
```

Рисунок 9 – Класс MovablePoint, который имплементирует Movable (2)

Вывод

В результате выполнения данной практической работы я разобрался с абстрактными классами и методами, также понял работу наследования, имплементации и агрегации.