|  |
| --- |
|  |
| 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 ПРАКТИЧЕСКОЙ РАБОТЕ №3**

Выполнил студент группы ИНБО-02-20 Бойко К.В.

Принял Степанов П.В.

Практическая работа выполнена «\_\_\_»\_\_\_\_\_\_\_2021г.

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2021г.

Отметка о выполнении

**Москва – 2021 г.**

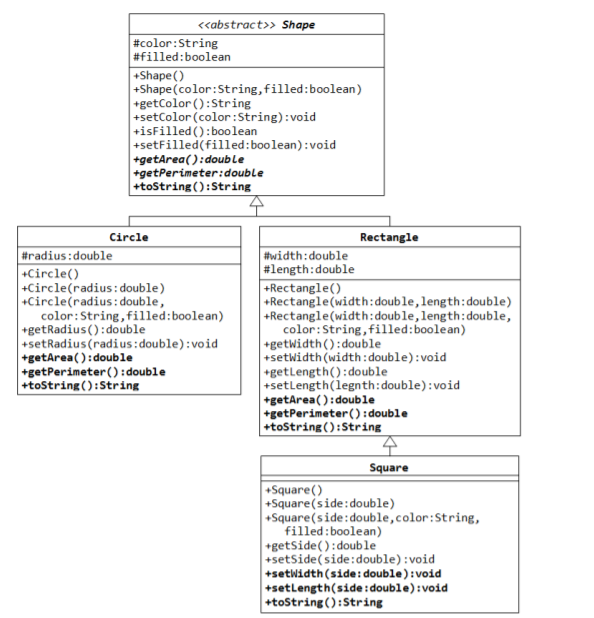
1. **Задание**

**Упражнение 1.**

Создайте абстрактный родительский суперкласс Shape и его дочерние классы (подклассы).

**Упражнение 2.**

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 поля или переменные класса, объявлены с модификатором protected color (тип String) и filled (тип boolean). Такие защищенные 27 переменные могут быть доступны в подклассах и классах в одном пакете. Они обозначаются со знаком “#” на 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).

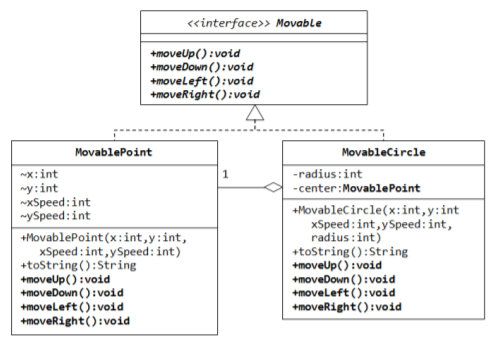
В подклассах Circle (круг) и Rectangle (прямоугольник) должны переопределяться абстрактные методы getArea() и getPerimeter(), чтобы обеспечить их надлежащее выполнение для конкретных экземпляров типа подкласс. Также необходимо для каждого подкласса переопределить toString() .

**Упражнение 3.**

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.

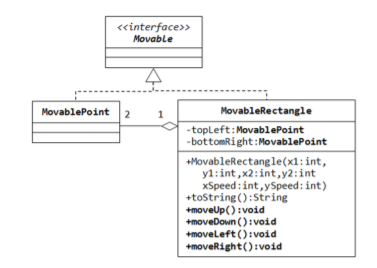
**Упражнение 4.**

Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable.



**Упражнение 5.**

Напишите новый класс MovableRectangle (движущийся прямоугольник). Его можно представить как две движущиеся точки MovablePoints (представляющих верхняя левая и нижняя правая точки) и реализующие интерфейс Movable. Убедитесь, что две точки имеет одну и ту же скорость (нужен метод это проверяющий).



1. **Ход Работы**

(основной код доступен по ссылке <https://github.com/Zyablikitsme/Java_practice>)

В ходе выполнения работы были получены следующие исходные коды:

**Shape.java**

package ru.mirea.java.practice3.ex1;  
  
public abstract class Shape {  
 protected String color;  
 protected boolean filled;  
  
 public Shape() {  
 }  
  
 public Shape(String color, boolean filled) {  
 this.color = color;  
 this.filled = filled;  
 }  
  
 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;  
 }  
  
 public abstract double getArea();  
  
 public abstract double getPerimeter();  
  
 public abstract String toString();  
  
}

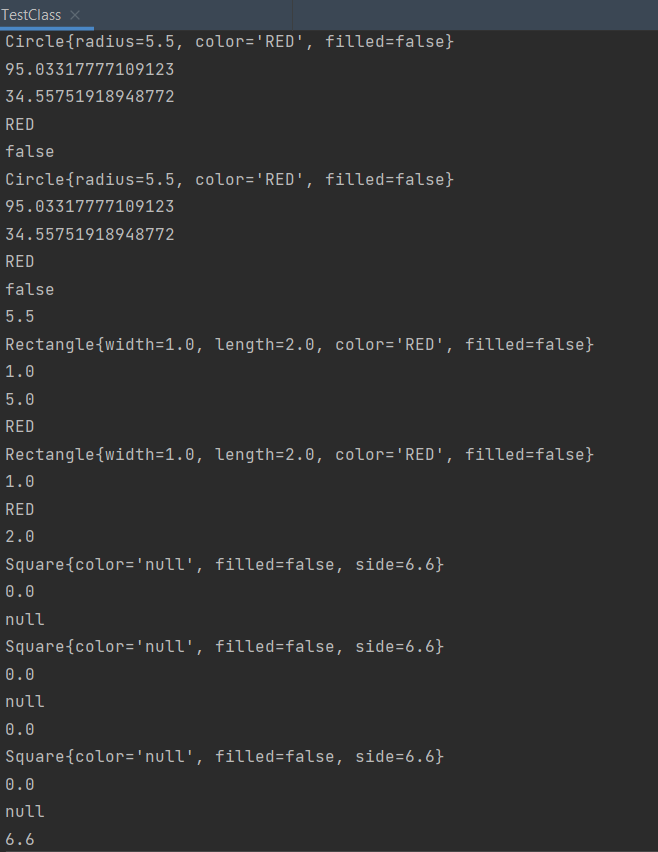
**Square.java**

package ru.mirea.java.practice3.ex1;  
  
public class Square extends Rectangle{  
 private double side; // ???  
  
 public Square() {  
 }  
  
 public Square(double side) { this.side = side; }  
  
 public Square(double side, String color, boolean filled){  
 this.side = side;  
 this.color = color;  
 this.filled = filled;  
 }  
  
 public double getSide() { return side; }  
  
 public void setSide(double side) { this.side = side; }  
  
 @Override  
 public void setWidth(double side){ width = side; } // ????  
  
 @Override  
 public void setLength(double side){ length = side; } // ???  
  
 @Override  
 public String toString() {  
 return "Square{" +  
 "color='" + color + '\'' +  
 ", filled=" + filled +  
 ", side=" + side +  
 '}';  
 }  
}

**MovablePoint.java**

package ru.mirea.java.practice3.ex2;  
  
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 String toString() {  
 return "MovablePoint{" +  
 "x=" + x +  
 ", y=" + y +  
 ", xSpeed=" + xSpeed +  
 ", ySpeed=" + ySpeed +  
 '}';  
 }  
  
 @Override  
 public void moveUp() {  
 y += ySpeed;  
 }  
  
 @Override  
 public void moveDown() {  
 y -= ySpeed;  
 }  
  
 @Override  
 public void moveLeft() {  
 x -= xSpeed;  
 }  
  
 @Override  
 public void moveRight() {  
 x += xSpeed;  
 }  
  
}

**Выходные данные программы** (при запуске TestClass.java)**:**

1. **Вывод**

В ходе данной работы мной были получены знания по работе с абстрактными классами, интерфейсами и наследованием на Java.