ПРАКТИЧЕСКАЯ РАБОТА №3
Абстрактные суперклассы и его подклассы в Java.

ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ:

Цель данной практической работы – освоить на практике работу с абстрактными классами и наследованием на Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы помечаются ключевым словом **abstract**.

Абстрактный метод не завершён. Он состоит только из объявления и не имеет тела:

abstract void yourMethod();

По сути, мы создаём шаблон метода. Например, можно создать абстрактный метод для вычисления площади фигуры в абстрактном классе Фигура. А все другие производные классы от главного класса могут уже реализовать свой код для готового метода. Ведь площадь у прямоугольника и треугольника вычисляется по разным алгоритмам и универсального метода не существует.

Если вы объявляете класс, производный от абстрактного класса, но хотите иметь возможность создания объектов нового типа, вам придётся предоставить определения для всех абстрактных методов базового класса. Если этого не сделать, производный класс тоже останется абстрактным, и компилятор заставит пометить новый класс ключевым словом **abstract**.

Абстрактный класс не может содержать какие-либо объекты, а также абстрактные конструкторы и абстрактные статические методы. Любой подкласс абстрактного класса должен либо реализовать все абстрактные методы суперкласса, либо сам быть объявлен абстрактным

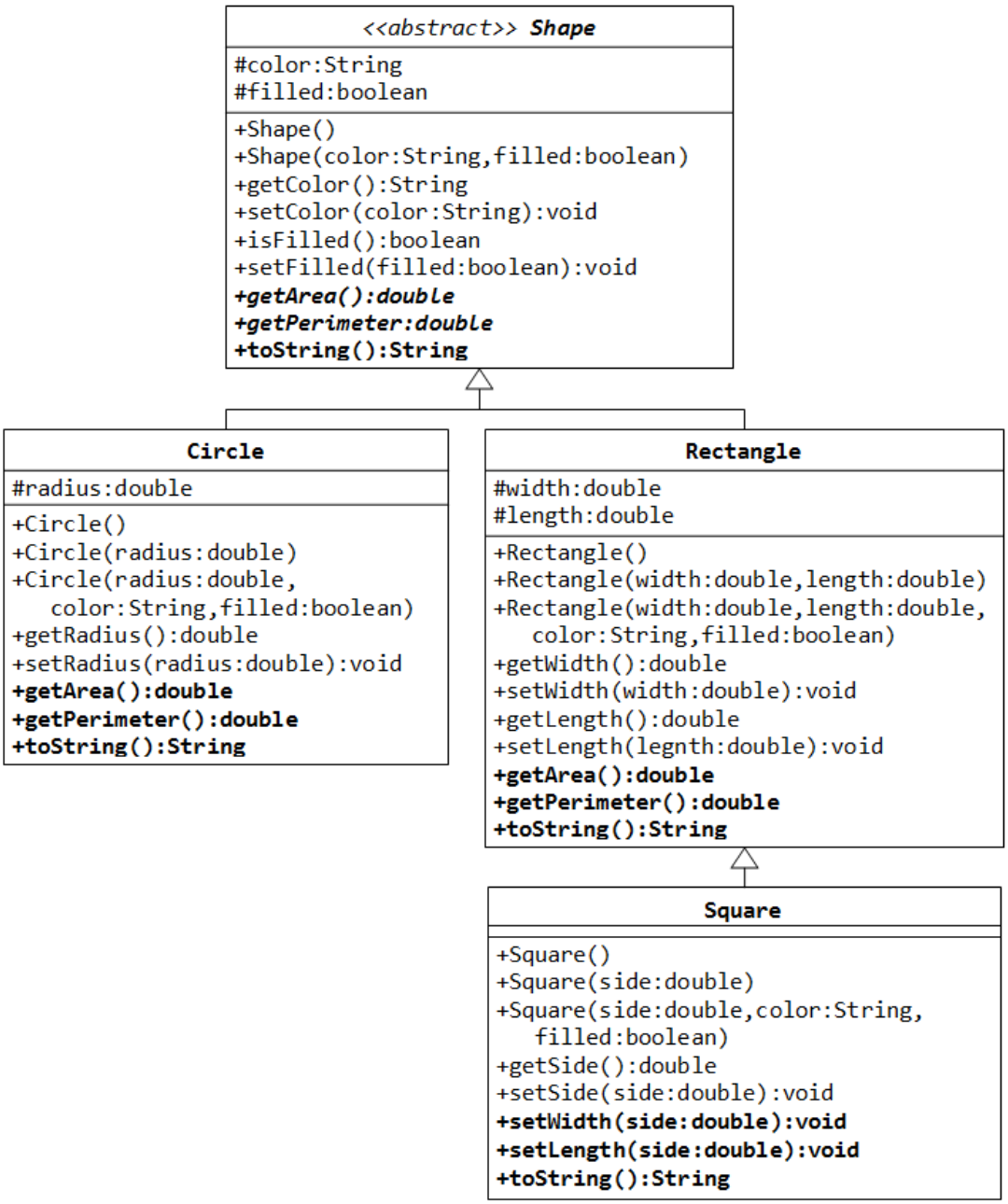
```
1 public abstract class Swim {
2     // абстрактный метод плавать()
3     abstract void neigh();
4 
5     // абстрактный класс может содержать и обычный метод
6     void run() {
7         System.out.println("Куда идешь?");
8     }
9 }
10 
11 class Swimmer extends Swim {
12 
13 }
```

ЗАДАНИЯ

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).

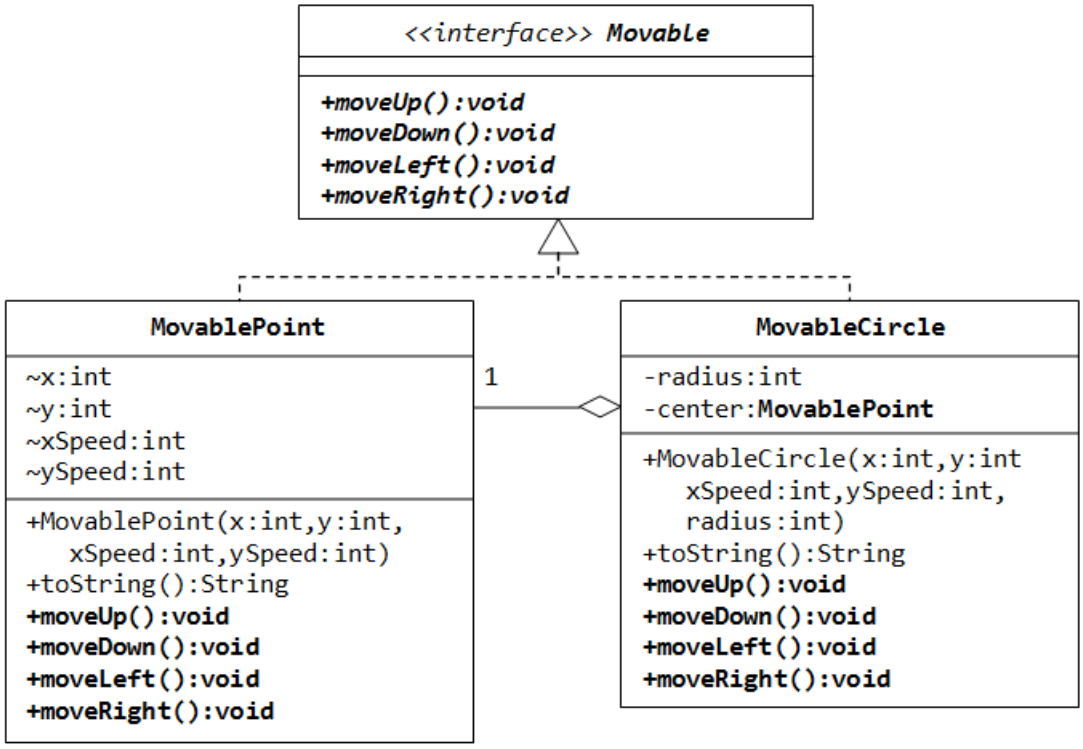
В подклассах Circle(круг) и Rectangle(прямоугольник) должны переопределяться абстрактные методы getArea() и getPerimeter(), чтобы обеспечить их надлежащее выполнение для конкретных экземпляров типа подкласс. Также необходимо для каждого подкласса переопределить toString() .

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.

```
Shape s1 = new Circle(5.5, "RED", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;                    // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "RED", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Задание 2.

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




```
public interface Movable { // saved as "Movable.java"
public void moveUp();
......
}
```

ПРИМЕР РЕШЕНИЯ ЗАДАНИЯ

Реализации класса Circle:

Circle.java

```
package shape;
import java.math.*;
public class Circle extends Shape{
    protected double radius;
    public Circle(){
        this.filled = false;
        this.color = "blue";
        radius = 1;
    }
    public Circle(double radius){
        this.filled = false;
        this.color = "blue";
        this.radius = radius;
    }
    public Circle(double radius, String color, boolean filled){
        this.radius = radius;
        this.color = color;
        this.filled = filled;
    }
    public double getRadius() {
        return radius;
    }
    public void setRadius(double radius) {
        this.radius = radius;
    }
```

```
    @Override
    public double getArea() {
        return Math.PI*radius*radius;
    }
    @Override
    public double getPerimeter() {
        return 2*Math.PI*radius;
    }
    @Override
    public String toString() {
        return "Shape: circle, radius: "+this.radius+", color:
"+this.color;
    }
}
```

2)

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).