|  |
| --- |
|  |
| 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, №4**

Выполнил студент группы ИНБО-02-20 Мандал Д. Т.

Принял Степанов П.В.

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

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

Отметка о выполнении

**Москва – 2020 г.**

**Практическое занятие №3**

Цель данной практической работы – освоить на практике работу

с абстрактными классами и наследованием на Java.

**Задание 1**

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).

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().

**Класс Circle**

package shapes;  
  
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){  
 super(color, filled);  
 this.radius = radius;  
 }  
  
 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;  
 }  
}

**Класс TestShapes**

package shapes;  
  
public class TestShapes {  
 public static void main(String[] args) {  
 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()); не будет работать, так как объект создан класса Shape у которого нет такого метода* 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(); не будет работать так как нельзя создавать экземпляры абстрактного класса* 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()); не будет работать, так как объект создан класса Shape у которого нет такого метода* 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()); не будет работать, так как объект создан класса Shape у которого нет такого метода  
  
// 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()); не будет работать, так как объект создан класса Rectangle у которого нет такого метода* 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());  
 }  
}

**Класс Rectangle**

package shapes;  
  
public class Rectangle extends Shape {  
 protected double width;  
 protected double length;  
  
 public Rectangle(){  
 width = 1;  
 length = 1;  
 color = "blue";  
 filled = false;  
 }  
  
 public Rectangle(double width, double length){  
 this.width = width;  
 this.length = length;  
 color = "blue";  
 filled = false;  
 }  
  
 public Rectangle(double width, double length, String color, boolean filled){  
 super(color, filled);  
 this.width = width;  
 this.length = length;  
 }  
  
 public double getLength() {  
 return length;  
 }  
  
 public double getWidth() {  
 return width;  
 }  
  
 public void setWidth(double width) {  
 this.width = width;  
 }  
  
 public void setLength(double length) {  
 this.length = length;  
 }  
  
 @Override  
 public String toString() {  
 return "Shape: rectangle, width and length: " + this.width + ", " + this.length + ", color: " + this.color;  
 }  
  
 @Override  
 public double getArea() {  
 return width \* length;  
 }  
  
 @Override  
 public double getPerimeter() {  
 return 2 \* width + 2 \* length;  
 }  
}

**Класс Shape**

package shapes;  
  
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 void setFilled(boolean filled) {  
 this.filled = filled;  
 }  
  
 public boolean isFilled() {  
 return filled;  
 }  
  
 public abstract String toString();  
  
 public abstract double getArea();  
  
 public abstract double getPerimeter();  
}

**Класс Square**

package shapes;  
  
public class Square extends Rectangle{  
 public Square(){  
 super(1, 1, "blue", false);  
 }  
  
 public Square(double side){  
 super(side, side, "blue", false);  
 }  
  
 public Square(double side, String color, boolean filled){  
 super(side, side, color, filled);  
 }  
  
 public double getSide(){  
 return super.getWidth();  
 }  
  
 public void setSide(double side){  
 super.setLength(side);  
 super.setWidth(side);  
 }  
  
 @Override  
 public void setLength(double side){  
 setSide(side);  
 }  
  
 @Override  
 public void setWidth(double side){  
 setSide(side);  
 }  
  
 @Override  
 public String toString(){  
 return "Shape: square, side: " + this.width + ", color: " + this.color;  
 }  
}

**Тест**

Shape: circle, radius: 5.5, color: RED

95.03317777109123

34.55751918948772

RED

false

Shape: circle, radius: 5.5, color: RED

95.03317777109123

34.55751918948772

RED

false

5.5

Shape: rectangle, width and length: 1.0, 2.0, color: RED

2.0

6.0

RED

Shape: rectangle, width and length: 1.0, 2.0, color: RED

2.0

RED

2.0

Shape: square, side: 6.6, color: blue

43.559999999999995

blue

Shape: square, side: 6.6, color: blue

43.559999999999995

blue

6.6

Shape: square, side: 6.6, color: blue

43.559999999999995

blue

6.6

6.6

**Задание 2**

Вам нужно написать два класса MovablePoint и MovableCircle - которые

реализуют интерфейс Movable.



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).

**Интерфейс Movable**

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

**Класс MovableCircle**

package movable;  
  
public class MovableCircle implements Movable{  
  
 private int radius;  
 private MovablePoint center;  
  
 public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius){  
  
 this.radius = radius;  
 this.center = new MovablePoint(x, y, xSpeed, ySpeed);  
 }  
  
 @Override  
 public String toString() {  
 return "MovableCircle{" +  
 "radius=" + radius +  
 ", center=" + center +  
 '}';  
 }  
  
 @Override  
 public void moveUp() {  
 center.moveUp();  
 }  
  
 @Override  
 public void moveDown() {  
 center.moveDown();  
 }  
  
 @Override  
 public void moveLeft() {  
 center.moveLeft();  
 }  
  
 @Override  
 public void moveRight() {  
 center.moveRight();  
 }  
}

**Класс MovablePoint**

package movable;  
  
import java.util.Objects;  
  
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;  
 if (xSpeed < 0){  
 xSpeed \*= -1;  
 }  
 if (ySpeed < 0){  
 ySpeed \*= -1;  
 }  
 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() {  
 x += xSpeed;  
 }  
  
 @Override  
 public String toString() {  
 return "MovablePoint{" +  
 "x=" + x +  
 ", y=" + y +  
 ", xSpeed=" + xSpeed +  
 ", ySpeed=" + ySpeed +  
 '}';  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 MovablePoint that = (MovablePoint) o;  
 return xSpeed == that.xSpeed &&  
 ySpeed == that.ySpeed;  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(xSpeed, ySpeed);  
 }  
}

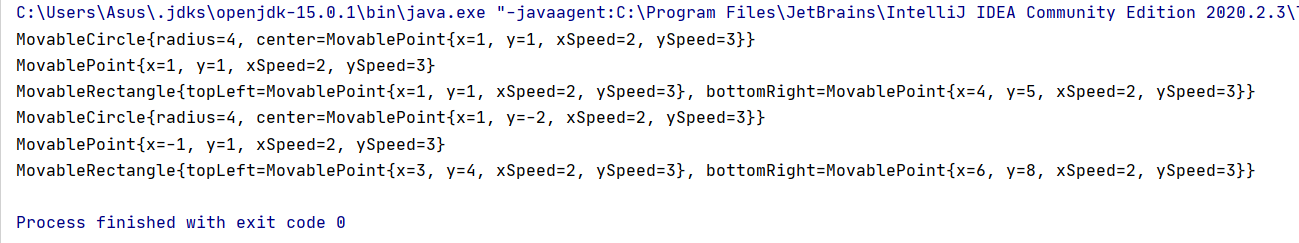
**Класс MovableRectangle**

package movable;  
  
public class MovableRectangle implements Movable{  
  
 private MovablePoint topLeft, bottomRight;  
  
 public MovableRectangle(int x1, int y1, int x2, int y2, int xSpeed, int ySpeed){  
 topLeft = new MovablePoint(x1, y1, xSpeed, ySpeed);  
 bottomRight = new MovablePoint(x2, y2, xSpeed, ySpeed);  
 }  
  
 @Override  
 public void moveUp() {  
 if (topLeft.equals(bottomRight)){  
 topLeft.moveUp();  
 bottomRight.moveUp();  
 }  
 }  
  
 @Override  
 public void moveDown() {  
 if (topLeft.equals(bottomRight)){  
 topLeft.moveDown();  
 bottomRight.moveDown();  
 }  
 }  
  
 @Override  
 public void moveLeft() {  
 if (topLeft.equals(bottomRight)){  
 topLeft.moveLeft();  
 bottomRight.moveLeft();  
 }  
 }  
  
 @Override  
 public void moveRight() {  
 if (topLeft.equals(bottomRight)){  
 topLeft.moveRight();  
 bottomRight.moveRight();  
 }  
 }  
  
 @Override  
 public String toString() {  
 return "MovableRectangle{" +  
 "topLeft=" + topLeft +  
 ", bottomRight=" + bottomRight +  
 '}';  
 }  
}

**Класс TestMovable**

package movable;  
  
public class TestMovable {  
 public static void main(String[] args) {  
 Movable m1 = new MovableCircle(1, 1, 2, 3, 4);  
 Movable m2 = new MovablePoint(1, 1, 2, 3);  
 Movable m3 = new MovableRectangle(1, 1, 4, 5, 2, 3);  
  
 System.*out*.println(m1);  
 System.*out*.println(m2);  
 System.*out*.println(m3);  
  
 m1.moveDown();  
 m2.moveLeft();  
 m3.moveRight();  
 m3.moveUp();  
  
 System.*out*.println(m1);  
 System.*out*.println(m2);  
 System.*out*.println(m3);  
 }  
}

**Тест**



**Практическое занятие №4**

**Цель работы**: Введение в событийное программирование

**Задание:**

Создайте JFrame приложение у которо есть следующие компоненты GUI:

Одна кнопка JButton labeled “AC Milan”

Другая JButton подписана “Real Madrid”

Надпись JLabel содержит текст “Result: 0 X 0”

Надпись JLabel содержит текст “Last Scorer: N/A ”

Надпись Label содержит текст “Winner: DRAW”;

Теперь всякий раз, когда вы нажимаете на кнопку AC Milan, результат будет

увеличиваться для Милана, чтобы стать сначала 1 X 0, затем 2 X 0. Last Scorer

означает последнюю забившую команду. В этом случае: AC Milan. И

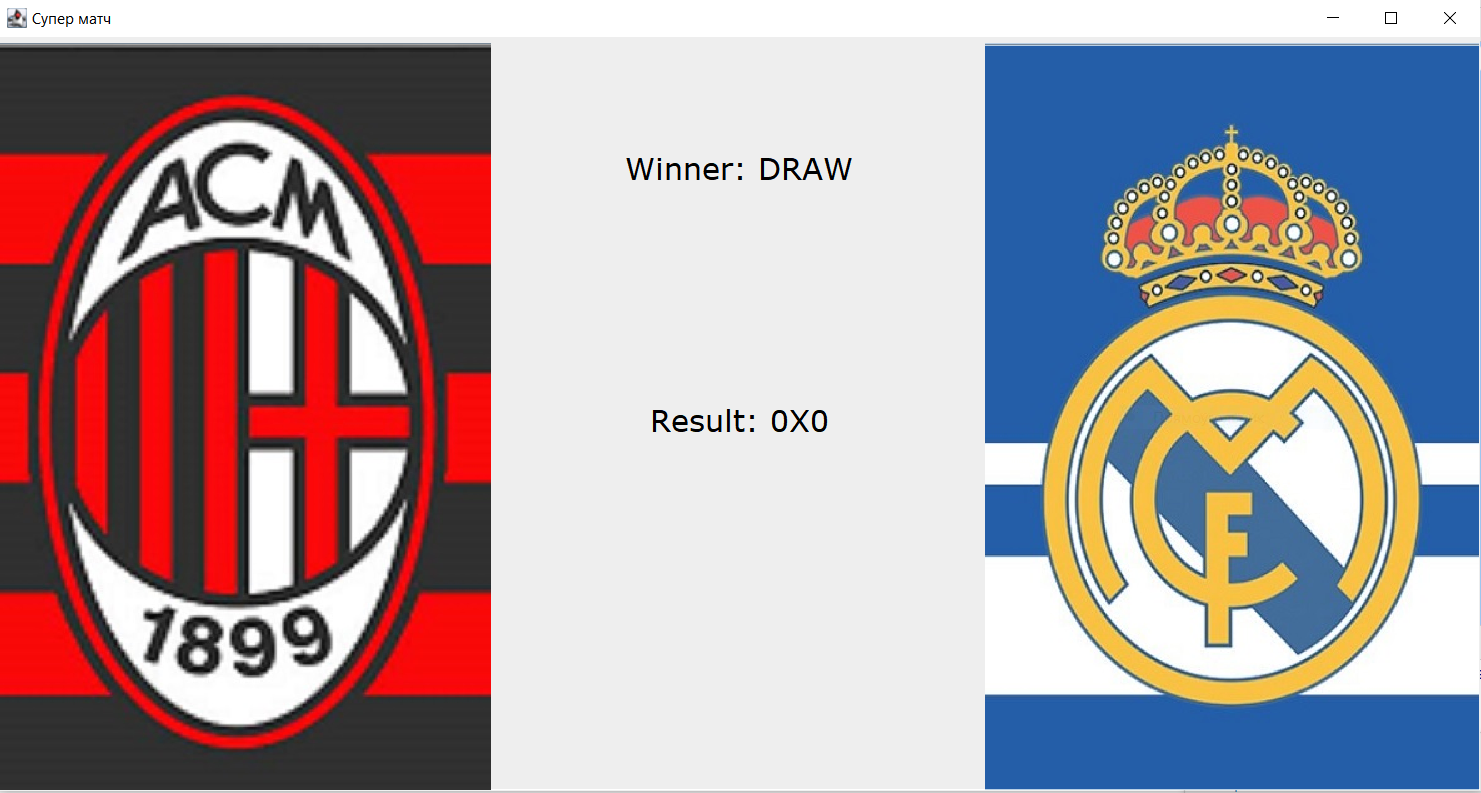
победителем становится команда, которая имеет больше кликов кнопку на

соответствующую, чем другая.

**Класс Match**

import javax.imageio.ImageIO;  
import javax.swing.\*;  
import javax.swing.border.Border;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.io.File;  
import java.io.IOException;  
  
public class Match extends JFrame {  
  
 JLabel LastScorer = new JLabel(), Result = new JLabel(), Winner = new JLabel();  
 int re = 0, mil = 0;  
 int end = (int)(Math.*random*() \* 10);  
  
 JButton Re = new JButton(), Mil = new JButton();  
 JPanel[] pnl = new JPanel[3];  
 JPanel[] pnl2 = new JPanel[3];  
  
 public void updateScore(){  
 Result.setText("Result: " + mil + "X" + re);  
 }  
  
 public void updateWinner(){  
 if (mil > re){  
 Winner.setText("Winner: AC Milan");  
 } else if (mil == re){  
 Winner.setText("Winner: DRAW");  
 } else {  
 Winner.setText("Winner: Real Madrid");  
 }  
 }  
  
 public void updateLast(String last){  
 LastScorer.setText("Last Scorer " + last);  
 }  
  
 Match(){  
 super("Супер матч");  
 setDefaultCloseOperation( *EXIT\_ON\_CLOSE* );  
 setLayout(new GridLayout(1, 3));  
  
 for(int i = 0 ; i < pnl.length ; i++)  
 {  
 pnl[i] = new JPanel();  
 add(pnl[i]);  
 }  
  
 pnl[1].setLayout(new GridLayout(3, 1));  
  
 for(int i = 0 ; i < pnl2.length ; i++)  
 {  
 pnl2[i] = new JPanel();  
 pnl[1].add(pnl2[i]);  
 }  
  
 updateScore();  
 updateWinner();  
 LastScorer.setText("");  
  
 Mil.setIcon(new ImageIcon("C:\\Users\\Asus\\Desktop\\mirea\\3 sem\\java\\github\\4\\4\\.idea\\mil.jpg"));  
 Re.setIcon(new ImageIcon("C:\\Users\\Asus\\Desktop\\mirea\\3 sem\\java\\github\\4\\4\\.idea\\rel.jpg"));  
  
 Mil.addActionListener(new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 mil++;  
 updateScore();  
 updateWinner();  
 updateLast("AC Milan");  
 }  
 });  
  
 Re.addActionListener(new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 re++;  
 updateScore();  
 updateWinner();  
 updateLast("Real Madrid");  
 }  
 });  
  
 Mil.setPreferredSize(new Dimension(900, 600));  
 Result.setPreferredSize(new Dimension(900, 200));  
 Re.setPreferredSize(new Dimension(900, 600));  
 LastScorer.setPreferredSize(new Dimension(900, 200));  
 Winner.setPreferredSize(new Dimension(900, 200));  
  
 Font font = new Font("Verdana", Font.*PLAIN*, 24);  
  
 Result.setVerticalAlignment(JLabel.*CENTER*);  
 Result.setHorizontalAlignment(JLabel.*CENTER*);  
 Winner.setVerticalAlignment(JLabel.*CENTER*);  
 Winner.setHorizontalAlignment(JLabel.*CENTER*);  
 LastScorer.setVerticalAlignment(JLabel.*CENTER*);  
 LastScorer.setHorizontalAlignment(JLabel.*CENTER*);  
  
 Result.setForeground(Color.*BLACK*);  
 Re.setForeground(Color.*BLACK*);  
 Mil.setForeground(Color.*BLACK*);  
 LastScorer.setForeground(Color.*BLACK*);  
 Winner.setForeground(Color.*BLACK*);  
  
 Result.setFont(font);  
 LastScorer.setFont(font);  
 Winner.setFont(font);  
 Re.setFont(font);  
 Mil.setFont(font);  
  
 pnl[0].add(Mil);  
 pnl[2].add(Re);  
 pnl2[0].add(Winner);  
 pnl2[1].add(Result);  
 pnl2[2].add(LastScorer);  
  
 setSize(1200, 640);  
 }  
  
 public static void main(String[] args) {  
 new Match().setVisible(true);  
 }  
  
}

**Тест**



**Ссылка на материалы**

<https://github.com/dmitrybot/Java3.git>

https://github.com/dmitrybot/Java4.git

**Выводы**

Я научился работать с UML-диаграммами и узнал о событийном программировании.