|  |
| --- |
|  |
| 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 Зотов Е. А.

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.



**Класс Circle**

public class Circle extends Shape {  
 protected double radius;  
 @Override  
 public double getArea() {  
 return Math.*PI*\*radius\*radius;  
 }  
  
 @Override  
 public double getPerimeter() {  
 return 2\*Math.*PI*\*radius;  
 }  
  
 @Override  
 public String toString() {  
 return "Circle{" +  
 "radius=" + radius +  
 ", color='" + color + '\'' +  
 ", filled=" + filled +  
 '}';  
 }  
 public Circle() {  
 }  
  
 public Circle(double radius) {  
 this.radius = radius;  
 }  
  
 public Circle(double radius, String color, boolean filled ) {  
 super(color, filled);  
 this.radius = radius;  
 }  
  
 public void setRadius(double radius) {  
 this.radius = radius;  
 }  
  
 public double getRadius() {  
 return radius;  
 }  
}

**Класс Main**

public class Main {  
 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()); объект суперкласса не может вызывать методы классов наследников  
  
 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(); нельзя создавать объекты 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()); объект суперкласса не может вызывать методы классов наследников  
  
 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()); объект суперкласса не может вызывать методы классов наследников  
  
// 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()); // у класса rectangle нет метода 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());  
 }  
}

**Класс Rectangle**

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

**Класс Shape**

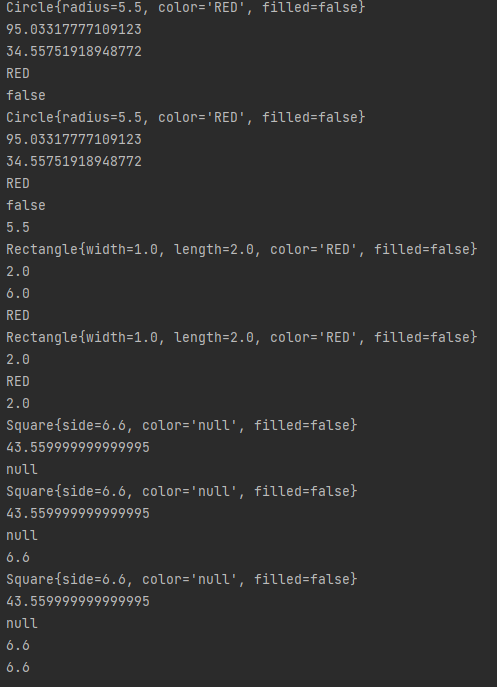
public abstract class Shape {  
 protected String color;  
 protected boolean filled;  
 public Shape() {  
  
 };  
  
 public Shape(String color, boolean filled) {  
 this.color = color;  
 this.filled = filled;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
 public boolean isFilled() {  
 return filled;  
 }  
  
 public void setColor(String color) {  
 this.color = color;  
 }  
  
 public void setFilled(boolean filled) {  
 this.filled = filled;  
 }  
  
 @Override  
 public String toString() {  
 return "Shape{" +  
 "color='" + color + '\'' +  
 ", filled=" + filled +  
 '}';  
 }  
 public abstract double getArea();  
 public abstract double getPerimeter();  
}

}

**Класс Square**

package ru.mirea.practice3.task1;  
  
public class Square extends Rectangle{  
 public Square() {  
 }  
  
 public Square(double side) {  
 super(side, side);  
 }  
  
 public Square(String color, boolean filled, double side) {  
 super(side, side, color, filled);  
 }  
 public double getSide() {  
 return super.getLength();  
 }  
 public void setSide(double side) {  
 super.setLength(side);  
 super.setWidth(side);  
 }  
  
 @Override  
 public void setWidth(double side) {  
 super.setWidth(side);  
 }  
  
 @Override  
 public void setLength(double side) {  
 super.setLength(side);  
 }  
  
 @Override  
 public String toString() {  
 return "Square{" +  
 "side=" + width +  
 ", color='" + color + '\'' +  
 ", filled=" + filled +  
 '}';  
 }  
}

**Тест**



**Задание 2**

Вам нужно написать два класса MovablePoint и MovableCircle - которые

реализуют интерфейс Movable.



**Интерфейс Movable**

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

**Класс MovableCircle**

public class MovableCircle implements Movable{  
 private int radius;  
 private MovablePoint center = new MovablePoint(0,0,0,0);  
  
 @Override  
 public String toString() {  
 return "MovableCircle{" +  
 "radius=" + radius +  
 ", center= (" + center.x + "," + center.y + ")" +  
 '}' + '\n';  
 }  
  
  
 public MovableCircle(int x, int y, int xSpeed, int ySpeed,int radius) {  
 this.radius = radius;  
 center.x = x;  
 center.y = y;  
 center.xSpeed = xSpeed;  
 center.ySpeed = ySpeed;  
 }  
  
 @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**

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

**Класс MovableRectangle**

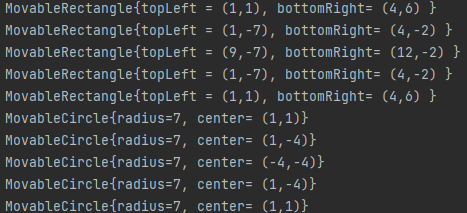
public class MovableRectangle implements Movable{  
 private MovablePoint topLeft = new MovablePoint(0,0,0,0);  
 private MovablePoint bottomRight = new MovablePoint(0,0,0,0);  
  
 public MovableRectangle(int x1, int y1, int x2, int y2, int xSpeed, int ySpeed) {  
 topLeft.y = y1;  
 topLeft.x = x1;  
 topLeft.xSpeed = xSpeed;  
 topLeft.ySpeed = ySpeed;  
 bottomRight.xSpeed = xSpeed;  
 bottomRight.y = y2;  
 bottomRight.x = x2;  
 bottomRight.ySpeed = ySpeed;  
 }  
  
 @Override  
 public String toString() {  
 return "MovableRectangle{" +  
 "topLeft = (" + topLeft.x + "," + topLeft.y + ")" +  
 ", bottomRight= (" + bottomRight.x + "," + bottomRight.y + ") " +  
 '}' + '\n';  
 }  
  
 @Override  
 public void moveUp() {  
 if (checkPointsSpeed()) {  
 topLeft.moveUp();  
 bottomRight.moveUp();  
 }  
 }  
  
 @Override  
 public void moveDown() {  
 if (checkPointsSpeed()) {  
 topLeft.moveDown();  
 bottomRight.moveDown();  
 }  
 }  
  
 @Override  
 public void moveLeft() {  
 if (checkPointsSpeed()) {  
 topLeft.moveLeft();  
 bottomRight.moveLeft();  
 }  
 }  
  
 @Override  
 public void moveRight() {  
 if (checkPointsSpeed()) {  
 topLeft.moveRight();  
 bottomRight.moveRight();  
 }  
 }  
 public boolean checkPointsSpeed() {  
 if (topLeft.xSpeed != bottomRight.xSpeed || topLeft.ySpeed != bottomRight.ySpeed) {  
 return false;  
 } else return true;  
 }  
}

}

**Класс Main**

public class Main {  
 public static void main(String[] args) {  
 MovableRectangle one = new MovableRectangle(1,1,4,6, 8,8);  
 System.*out*.print(one);  
 one.moveDown();  
 System.*out*.print(one);  
 one.moveRight();  
 System.*out*.print(one);  
 one.moveLeft();  
 System.*out*.print(one);  
 one.moveUp();  
 System.*out*.print(one);  
 MovableCircle two = new MovableCircle(1,1,5,5,7);  
 System.*out*.print(two);  
 two.moveDown();  
 System.*out*.print(two);  
 two.moveLeft();  
 System.*out*.print(two);  
 two.moveRight();  
 System.*out*.print(two);  
 two.moveUp();  
 System.*out*.print(two);  
  
 }  
}

**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

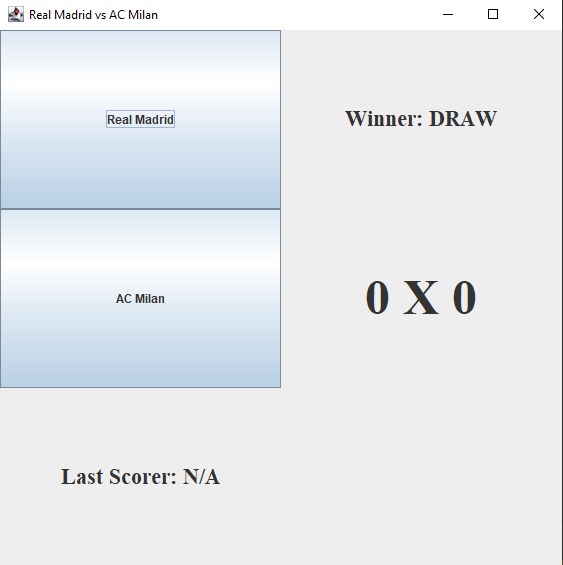
победителем становится команда, которая имеет больше кликов кнопку на

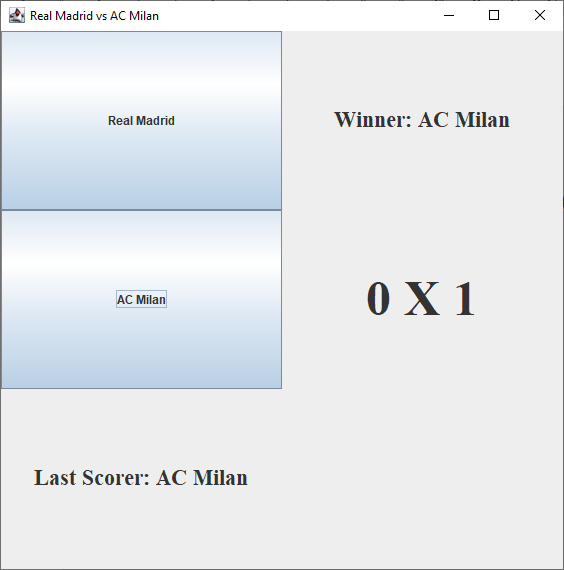
соответствующую, чем другая.

**Класс Match**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
  
public class Match extends JFrame {  
  
 JButton realM = new JButton("Real Madrid");  
 JButton milan = new JButton("AC Milan");  
  
 JLabel score = new JLabel();  
 JLabel winTeam = new JLabel();  
 JLabel lastGoal = new JLabel("Last Scorer: N/A");  
  
 int milGoal = 0, madrGoal = 0;  
  
 public Match() {  
 super("Real Madrid vs AC Milan");  
 setDefaultCloseOperation( *EXIT\_ON\_CLOSE* );  
 setLayout(new GridLayout(3, 1));  
 add(realM);  
 setSize(500,500);  
  
 updateScore();  
 updateWinTeam();  
  
 milan.addActionListener(new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 ++milGoal;  
 updateWinTeam();  
 updateScore();  
 updateLastGoal("AC Milan");  
 }  
 });  
  
 realM.addActionListener(new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 madrGoal++;  
 updateWinTeam();  
 updateScore();  
 updateLastGoal("Real Madrid");  
 }  
 });  
  
 Font fnt = new Font("Times new roman",Font.*BOLD*,50);  
 Font fnt22 = new Font("Times new roman",Font.*BOLD*,22);  
 add(winTeam);  
 winTeam.setVerticalAlignment(JLabel.*CENTER*);  
 winTeam.setHorizontalAlignment(JLabel.*CENTER*);  
 winTeam.setFont(fnt22);  
 add(milan);  
 add(score);  
 score.setVerticalAlignment(JLabel.*CENTER*);  
 score.setHorizontalAlignment(JLabel.*CENTER*);  
 score.setFont(fnt);  
 add(lastGoal);  
 lastGoal.setVerticalAlignment(JLabel.*CENTER*);  
 lastGoal.setHorizontalAlignment(JLabel.*CENTER*);  
 lastGoal.setFont(fnt22);  
 }  
  
 public void updateWinTeam() {  
 if (milGoal > madrGoal)  
 winTeam.setText("Winner: AC Milan");  
 else if (milGoal < madrGoal)  
 winTeam.setText("Winner: Real Madrid");  
 else  
 winTeam.setText("Winner: DRAW");  
 }  
  
 public void updateLastGoal(String Team) {  
 lastGoal.setText("Last Scorer: " + Team);  
 }  
  
 public void updateScore() {  
 score.setText( madrGoal + " X " + milGoal);  
 }  
  
 public static void main(String[] args) {  
 new Match().setVisible(true);  
 }  
}

**Тест**







**Выводы**

Были получены навыки работы с UML-диаграммами и событийным программированием.

Ссылка на репозиторий с работой - <https://github.com/kwazr/mirea-practice/tree/master/src/ru/mirea>