|  |
| --- |
|  |
| 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)**

**ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ №3, №4**

**по дисциплине «Программирование на языке Джава»**

Выполнил студент группы ИНБО-02-20 Чернов Я. А.

Принял преподаватель Степанов П.В.

Практические работы выполнены «\_\_\_»\_\_\_\_\_\_\_2021г.

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

Отметка о выполнении

**Москва 2021 г.**

**Практическое занятие №3**

Цель данной практической работы – освоить на практике работу

с абстрактными классами и наследованием на Java.

**Задание 1**

Переписать 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 Main**

**package** ru.mirea.Practice3.task1;  
  
**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());* 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());  
 }  
}

**Класс Circle**

**package** ru.mirea.Practice3.task1;  
  
**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**;  
 }  
 }

**Класс Rectangle**

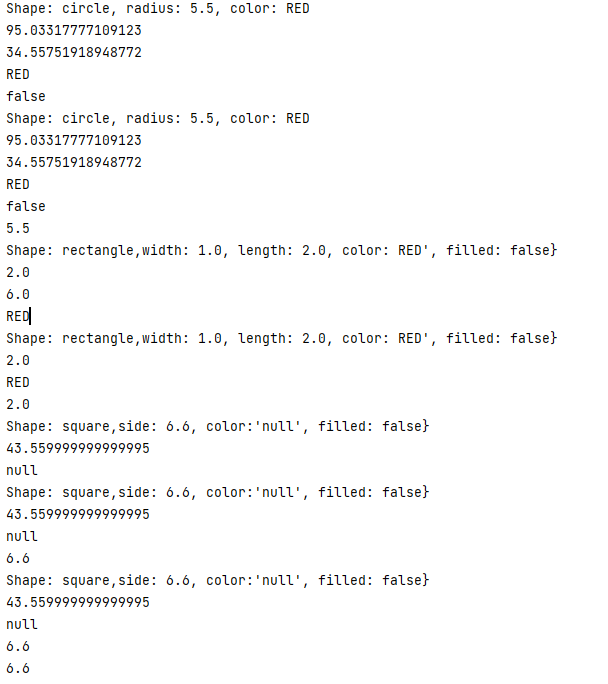
**package** ru.mirea.Practice3.task1;  
  
**public class** Rectangle **extends** Shape{  
 **protected double** width;  
 **protected double** length;  
 @Override  
 **public double** getArea() {  
 **return width**\***length**;  
 }  
 @Override  
 **public double** getPerimeter() {  
 **return** (2\***width**+2\***length**);  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Shape: rectangle,"** +  
 **"width: "** + **width** +  
 **", length: "** + **length** +  
 **", color: "** + **color** + **'\''** +  
 **", filled: "** + **filled** +  
 **'}'**;  
 }  
 **public** Rectangle() {  
  
 }  
 **public** Rectangle(**double** wight, **double** length) {  
 **this**.width = wight;  
 **this**.length = length;  
 }  
  
 **public** Rectangle(**double** wight, **double** length,String color, **boolean** filled ) {  
 **super**(color, filled);  
 **this**.width = wight;  
 **this**.length = length;  
 }  
  
 **public double** getWight() {  
 **return** width;  
 }  
  
 **public double** getLength() {  
 **return** length;  
 }  
  
 **public void** setWight(**double** wight) {  
 **this**.width = wight;  
 }  
  
 **public void** setLength(**double** length) {  
 **this**.length = length;  
 }  
  
}

**Класс Shape**

**package** ru.mirea.Practice3.task1;  
  
**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**.setWight(side);  
 }  
  
 @Override  
 **public void** setWight(**double** side) {  
  
 **super**.setWight(side);  
 }  
  
 @Override  
 **public void** setLength(**double** side) {  
  
 **super**.setLength(side);  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Shape: square,"** +  
 **"side: "** + getSide() +  
 **", color:'"** + **color** + **'\''** +  
 **", filled: "** + **filled** +  
 **'}'**;  
 }  
}

**Тестирование**

**Задание 2**

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).

**Класс Main**

**package** ru.mirea.Practice3.task2;  
  
**public class** Main {  
 **public static void** main(String[] args) {  
 MovableRectangle one = **new** MovableRectangle(1,1,2,4, 6,6);  
 System.***out***.print( one.toString());  
 one.moveDown();  
 System.***out***.print( one.toString());  
 one.moveRight();  
 System.***out***.print( one.toString());  
 one.moveLeft();  
 System.***out***.print( one.toString());  
 one.moveUp();  
 System.***out***.print( one.toString());  
 MovableCircle two = **new** MovableCircle(1,1,4,4,5);  
 System.***out***.print(two.toString());  
 two.moveDown();  
 System.***out***.print(two.toString());  
 two.moveLeft();  
 System.***out***.print(two.toString());  
 two.moveRight();  
 System.***out***.print(two.toString());  
 two.moveUp();  
 System.***out***.print(two.toString());  
  
 }  
}

**Интерфейс Movable**

**package** ru.mirea.Practice3.task2;  
  
**public interface** Movable {  
 **void** moveUp();  
 **void** moveDown();  
 **void** moveLeft();  
 **void** moveRight();  
}

**Класс MovablePoint**

**package** ru.mirea.Practice3.task2;  
  
**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**;  
 }  
}

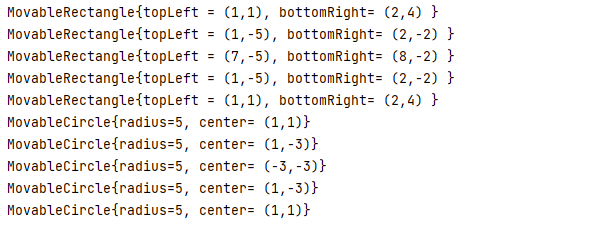
**Класс MovableRectangle**

**package** ru.mirea.Practice3.task2;  
  
**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 MovableCircle**

**package** ru.mirea.Practice3.task2;  
  
**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();  
 }  
}

**Тестирование**

****

**Практическое занятие №4**

**Цель работы**: Введение в событийное программирование.

**Задание:**

Создать 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 кнопка нажимается 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

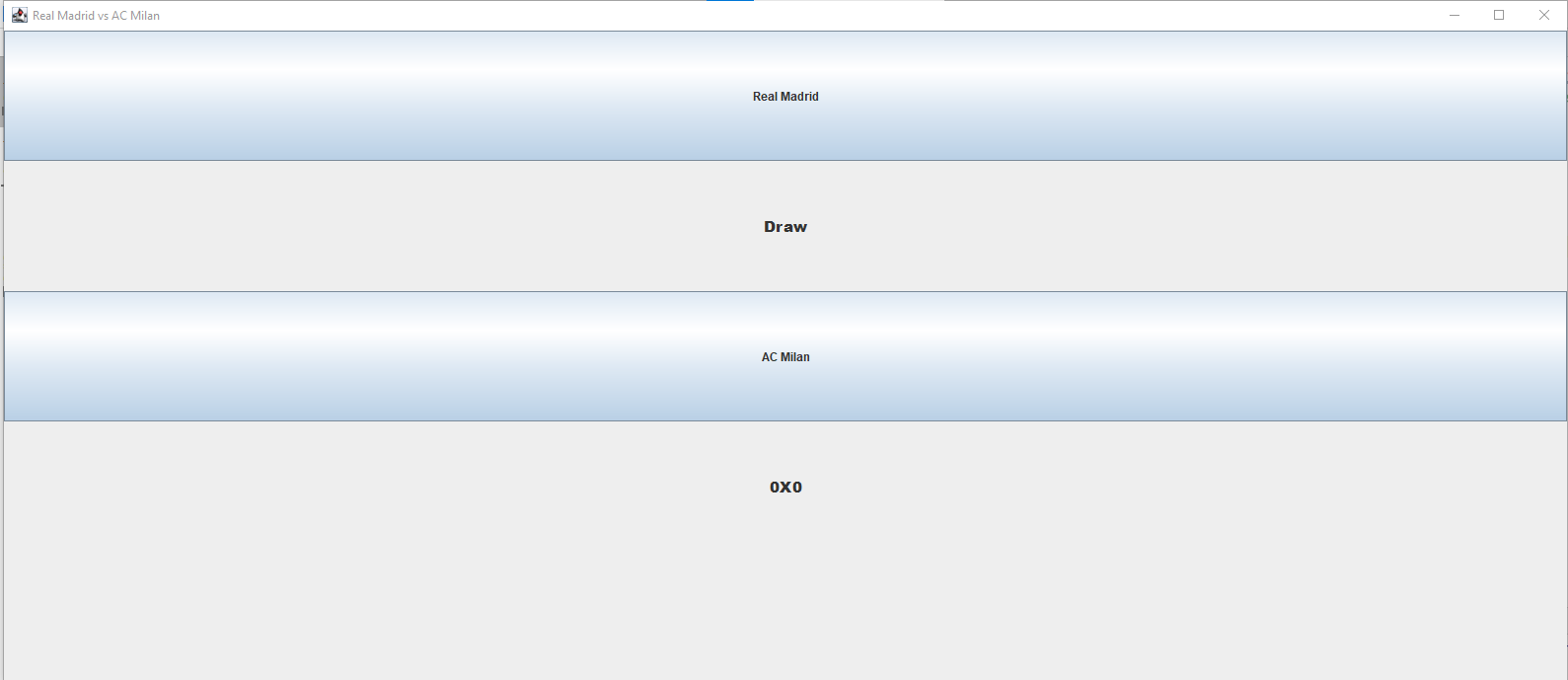
победителем становится команда, которая имеет больше кликов кнопку на

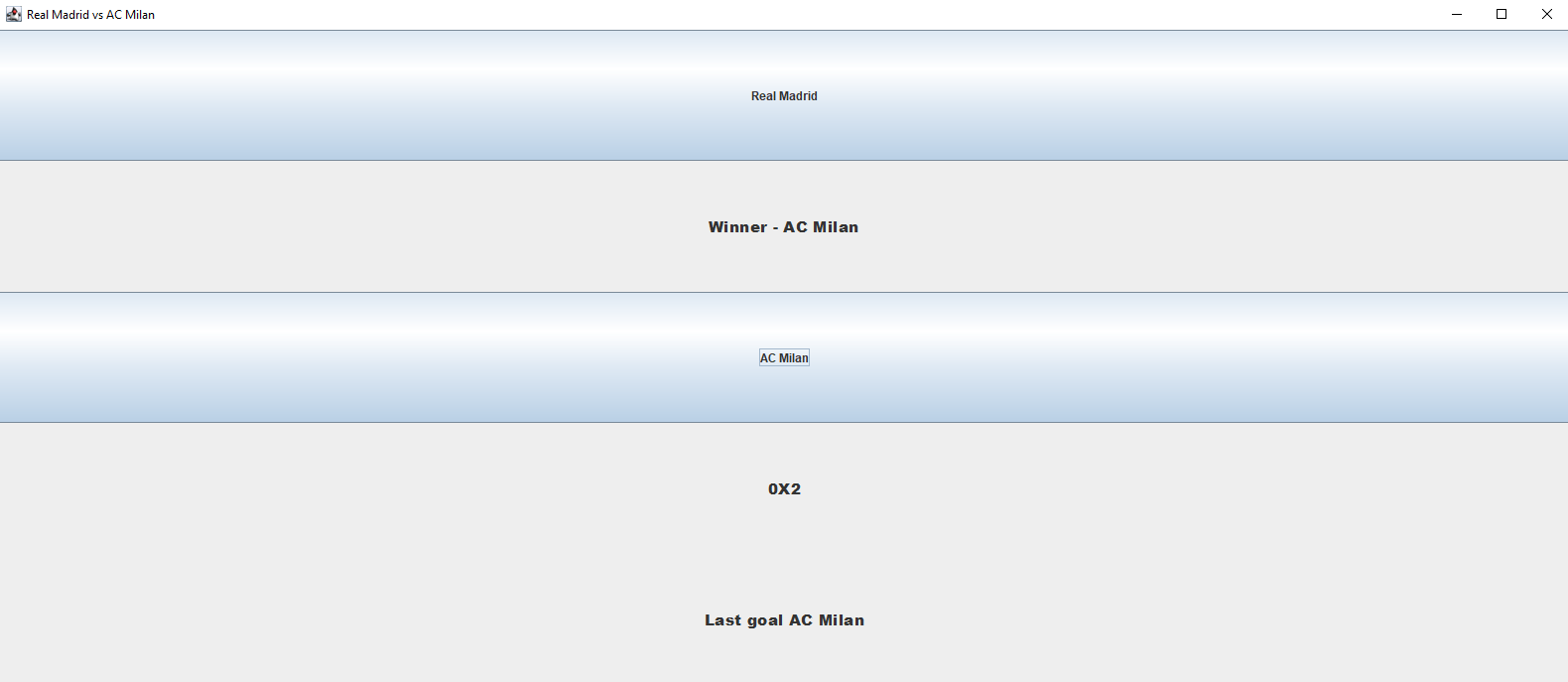
соответствующую, чем другая.

**Класс Game**

**package** ru.mirea.Practice4;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
  
**public class** JFRfootball **extends** JFrame{  
 JButton **RealMadrid** = **new** JButton(**"Real Madrid"**);  
 JButton **ACMilan** = **new** JButton(**"AC Milan"**);  
  
 JLabel **Score** = **new** JLabel();  
 JLabel **WinFT** = **new** JLabel();  
 JLabel **LastGoal** = **new** JLabel();  
  
 **int MilanGoal** = 0, **MadridGoal** = 0;  
  
 **public** JFRfootball() {  
 **super**(**"Real Madrid vs AC Milan"**);  
 setDefaultCloseOperation( ***EXIT\_ON\_CLOSE*** );  
 setLayout(**new** GridLayout(5, 3));  
 add(**RealMadrid**);  
 setSize(1600,700);  
  
 newScore();  
 newWinFT();  
  
 **ACMilan**.addActionListener(**new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 ++**MilanGoal**;  
 newWinFT();  
 newScore();  
 newLastGoal(**"AC Milan"**);  
 }  
 });  
  
 **RealMadrid**.addActionListener(**new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **MadridGoal**++;  
 newWinFT();  
 newScore();  
 newLastGoal(**"Real Madrid"**);  
 }  
 });  
  
 Font fnt = **new** Font(**"Arial Black"**,Font.***BOLD***,15);  
 Font fnt22 = **new** Font(**"Arial Black"**,Font.***BOLD***,15);  
 add(**WinFT**);  
 **WinFT**.setVerticalAlignment(JLabel.***CENTER***);  
 **WinFT**.setHorizontalAlignment(JLabel.***CENTER***);  
 **WinFT**.setFont(fnt22);  
 add(**ACMilan**);  
 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** newWinFT() {  
 **if** (**MilanGoal** > **MadridGoal**)  
 **WinFT**.setText(**"Winner - AC Milan"**);  
 **else if** (**MilanGoal** < **MadridGoal**)  
 **WinFT**.setText(**"Winner - Real Madrid"**);  
 **else  
 WinFT**.setText(**"Draw"**);  
 }  
  
 **public void** newLastGoal(String comand) {  
 **LastGoal**.setText(**"Last goal "** + comand);  
 }  
  
 **public void** newScore() {  
 **Score**.setText( **MadridGoal** + **"X"** + **MilanGoal**);  
 }  
  
 **public static void** main(String[] args) {  
 **new** JFRfootball().setVisible(**true**);  
 }  
}

**Тестирование**







**Выводы**

На практике была освоена работа с абстрактными классами и наследованием на Java, а также с событийным программированием.