|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГОПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиППО)

Практические РАБОТы

по дисциплине «Программирование на языке Джава»

Выполнил студент группы ИВБО-06-20  *Калашников А.А.*

Принял старший преподаватель *Рачков А.В.*

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

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

Москва 2021

**СОДЕРЖАНИЕ**

[Практическая работа № 1 3](#_Toc82952653)

[Практическая работа № 2 7](#_Toc82952654)

[Практическая работа № 3 10](#_Toc82952655)

[Практическая работа № 4 26](#_Toc82952656)

Практическая работа № 1

Цель работы:

Освоить на практике работу с классами на Java.

Теоретическое введение

Класс — это тип данных, создаваемый программистом для  
решения задач. Он представляет из себя шаблон, или прототип, который  
определяет и описывает статические свойства и динамическое поведение, общие для всех объектов одного и того же вида. Экземпляр класса - реализация конкретного объекта типа класса. Все экземпляры класса имеют  
аналогичные свойства, как задано в определении класса. Например, вы можете определить класс с именем "Студент " и создать три экземпляра класса "Студент": " Петр", " Павел" и " Полина ".

Чтобы создать экземпляр класса, вы должны выполнить следующие  
действия:

* объявить идентификатор экземпляра (имя экземпляра) конкретного класса
* cконструировать экземпляр класса (то есть выделить память для  
  экземпляра и инициализировать его) с помощью оператора "new".

Доступ к компонентам класса осуществляется с помощью операции получения доступа, а именно операции точка “.”

Переменные — поля данных класса и методы класса и являются компонентами класса. Для ссылки на переменную-поле данных класса или метод, вы должны:

* сначала создать экземпляр класса, который вам нужен;
* затем, использовать оператор точка “.” чтобы сослаться на элемент класса (переменную-поле данных или метод класса).

Метод может

* принимать параметры из вызова (как в функции);
* выполнять операции, описанные в теле метода, и;
* возвращать часть результата (или void) в точку вызова.

Конструктор – это специальный метод класса, который имеет то же имя, что используется в качестве имени класса. Он отличается от обычного метода следующим:

* название метода-конструктора совпадает с именем класса, а имя класса по конвенции, начинается с заглавной буквы;
* конструктор не имеет возвращаемого значения типа, таким образом, нет объявления типа возвращаемого значения при объявлении;
* конструктор может быть вызван только через оператор «new», он может быть использован только один раз, чтобы инициализировать построенный экземпляр.
* вы не можете впоследствии вызвать конструктор в теле программы подобно обычным методам (функциям);
* конструкторы не наследуется (будет объяснено позже).

Конструктор без параметров называется конструктором по умолчанию, который инициализирует поля данных через их значения по умолчанию.

Выполнение лабораторной работы

Задание:

*Реализовать простейший класс «Книга».*

Решение:

Создаём новый класс Book. В нём будут описаны 2 конструктора и методы для установки и получения значений полей класса.

package ru.mirea.lab1;

import java.lang.\*;

public class Book {

private String name;

private String author;

private int pages;

public Book(String name, String author, int pages) {

this.name = name;

this.author = author;

this.pages = pages;

}

public Book(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;

}

public int getPages() {

return pages;

}

public void setPages(int pages) {

this.pages = pages;

}

public String toString() {

return "Книга: " + name + "; Автор: " + author + "; Страниц: " + pages;

}

}

Создаём новый класс TestBook. В нём создаём метод main для запуска программы. Также создаем три объекта класса Book и с помощью метода toString() класса Book выводим на экран информацию о каждом объекте.

package ru.mirea.lab1;

import java.lang.\*;

public class TestBook {

public static void main(String[] args) {

Book b1 = new Book("Война и мир", "Лев Толстой", 1696);

Book b2 = new Book("Мертвые души", "Николай Гоголь", 355);

Book b3 = new Book("Евгений Онегин");

b3.setAuthor("Александр Пушкин");

b3.setPages(448);

System.out.println(b1.toString());

System.out.println(b2.toString());

System.out.print(b3.toString());

}

}

На рисунке 1.1 показан вывод программы. По нему можно убедится, что программа работает правильно.

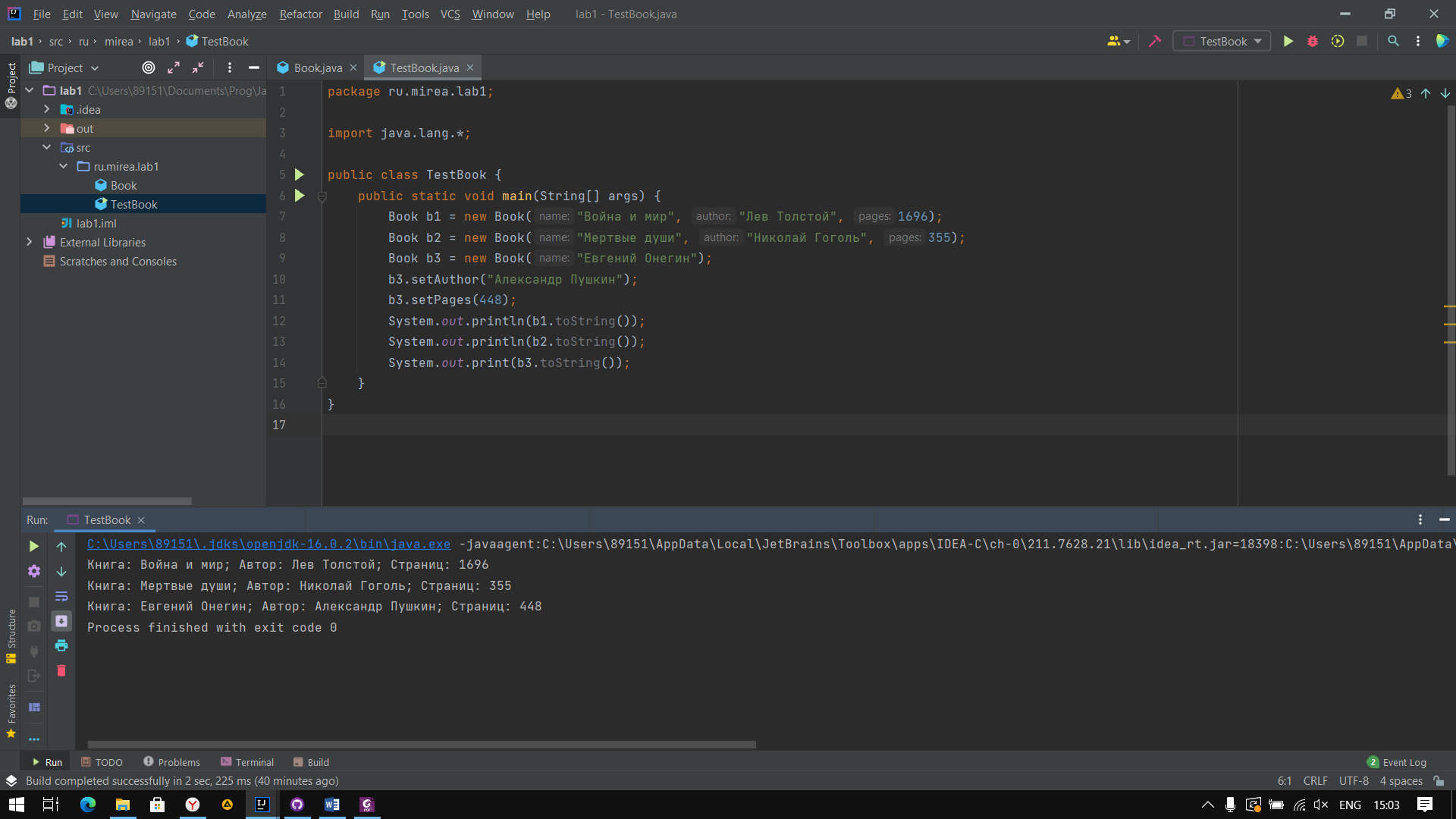


Рисунок 1.1 - Вывод программы.

Выводы по работе:

Выполняя данную практическую работу, я научился создавать класс, поля, методы и конструкторы в нём, а также экземпляры класса и обращаться к его полям и методам.

Практическая работа № 2

Цель работы:

Работа с UML-диаграммами классов.

Теоретическое введение

Язык моделирования Unified Modeling Language (UML) является стандартом с 1998 года для проектирования и документирования объектно-ориентированных программ.

Средствами UML в виде диаграмм можно графически изобразить класс и экземпляр класса. Графически представляем класс в виде прямоугольника, разделенного на три области – область именования класса, область инкапсуляции данных и область операций (методы).

Имя определяет класс. Переменные содержат статические атрибуты класса, или описывают свойства класса. Методы описывают динамическое поведение класса. Другими словами, класс инкапсулирует статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“коробке” или прямоугольнике).

На рисунке 2.1 приведен общий вид UML диаграммы класса.

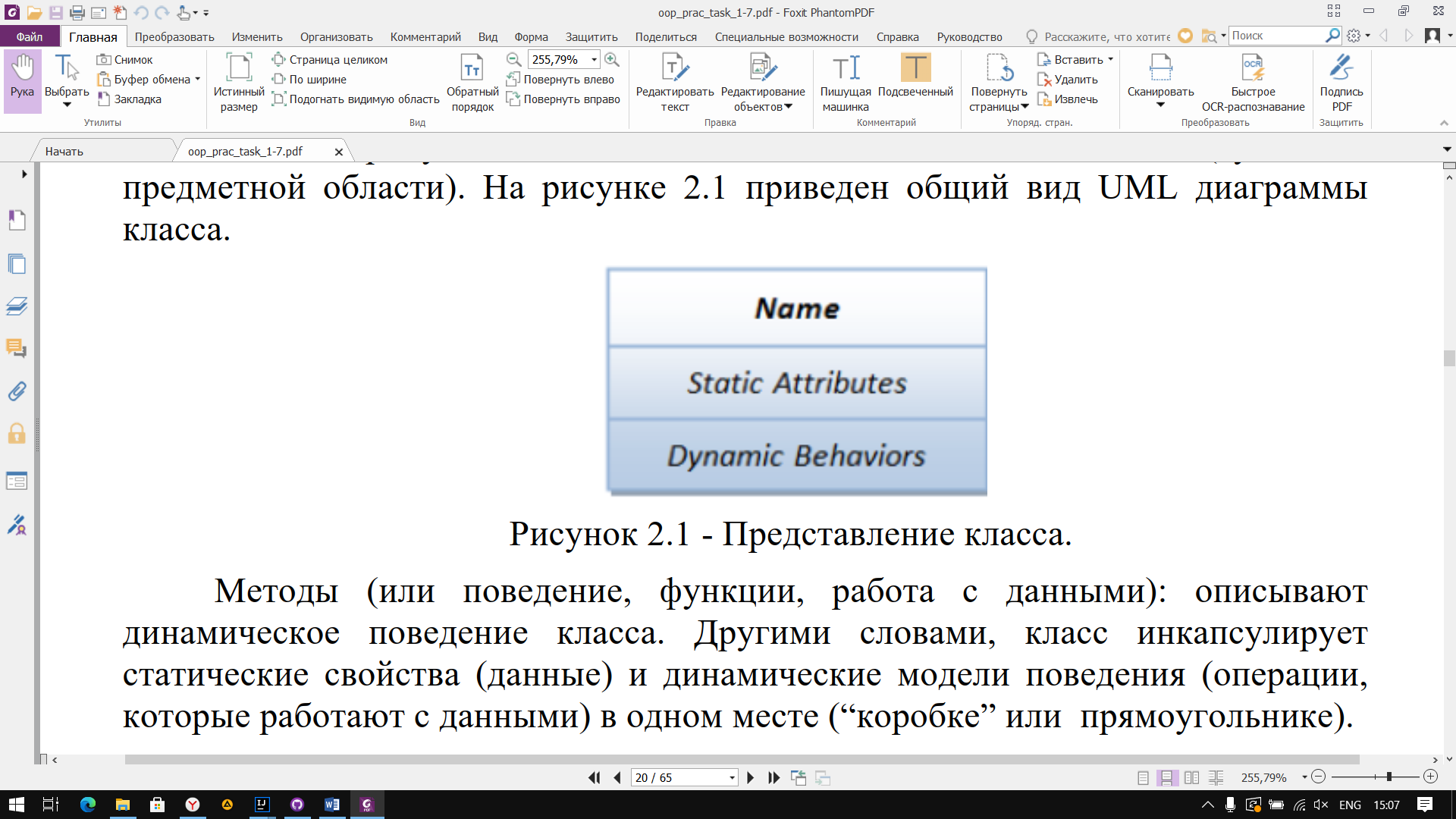


Рисунок 2.1 - Представление класса.

На рисунке 2.2 показаны два экземпляра класса типа Student "paul" и  
"peter".

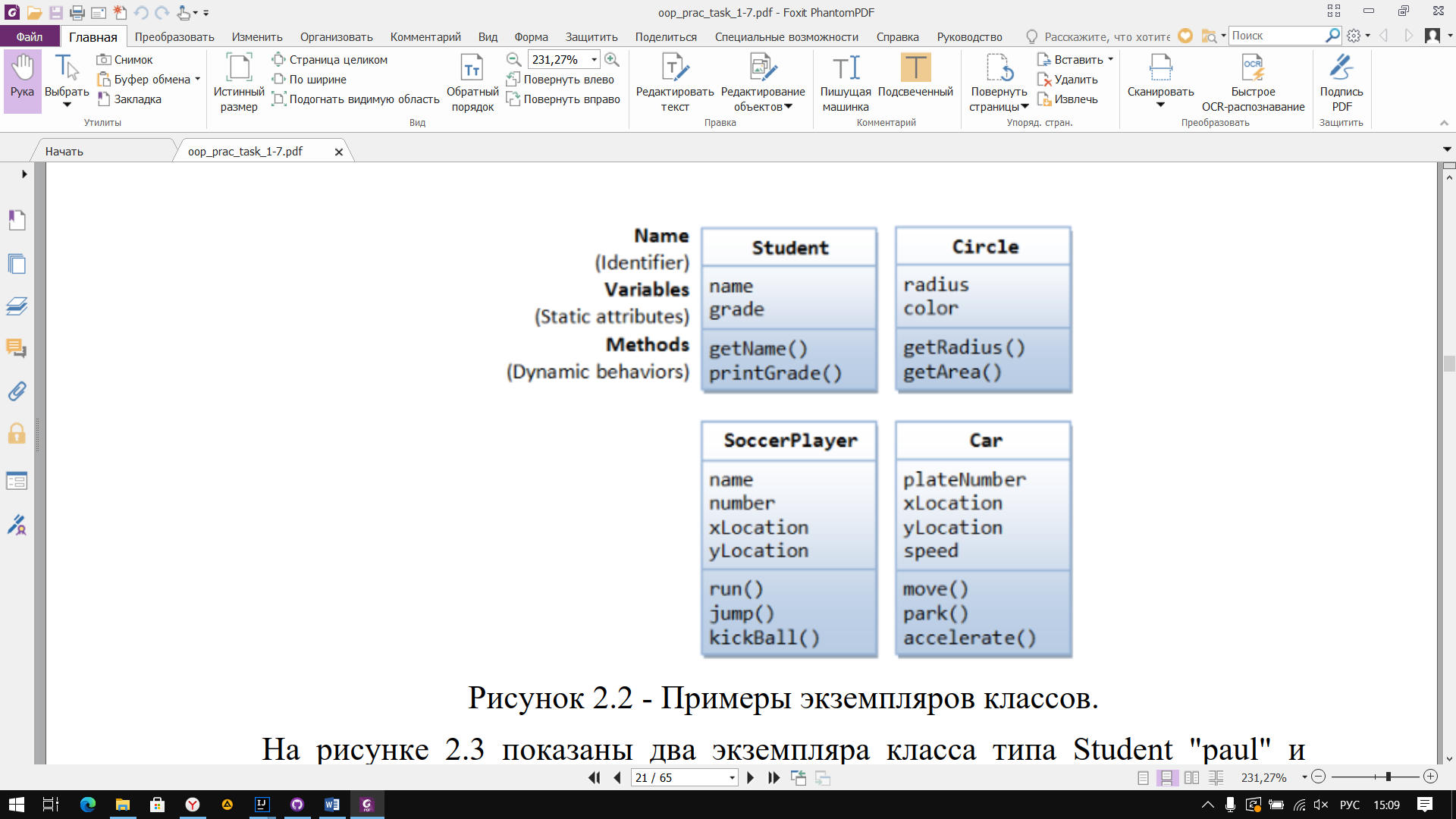


Рисунок 2.2 - Примеры экземпляров классов.

Приведенные выше диаграммы классов описаны в соответствии с UML

нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три области, одна содержит название, две вторых содержат поля и методы класса, соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр также представляется в виде прямоугольника, разделенного на три части, в первой части помещается надпись с именем экземпляра.

Выполнение лабораторной работы

Задание:

По диаграмме класса UML описывающей сущность Автор. Необходимо написать программу, которая состоит из двух классов Author и TestAuthor. Класс Author должен содержать реализацию методов, представленных на диаграмме класса на рисунке 2.3.

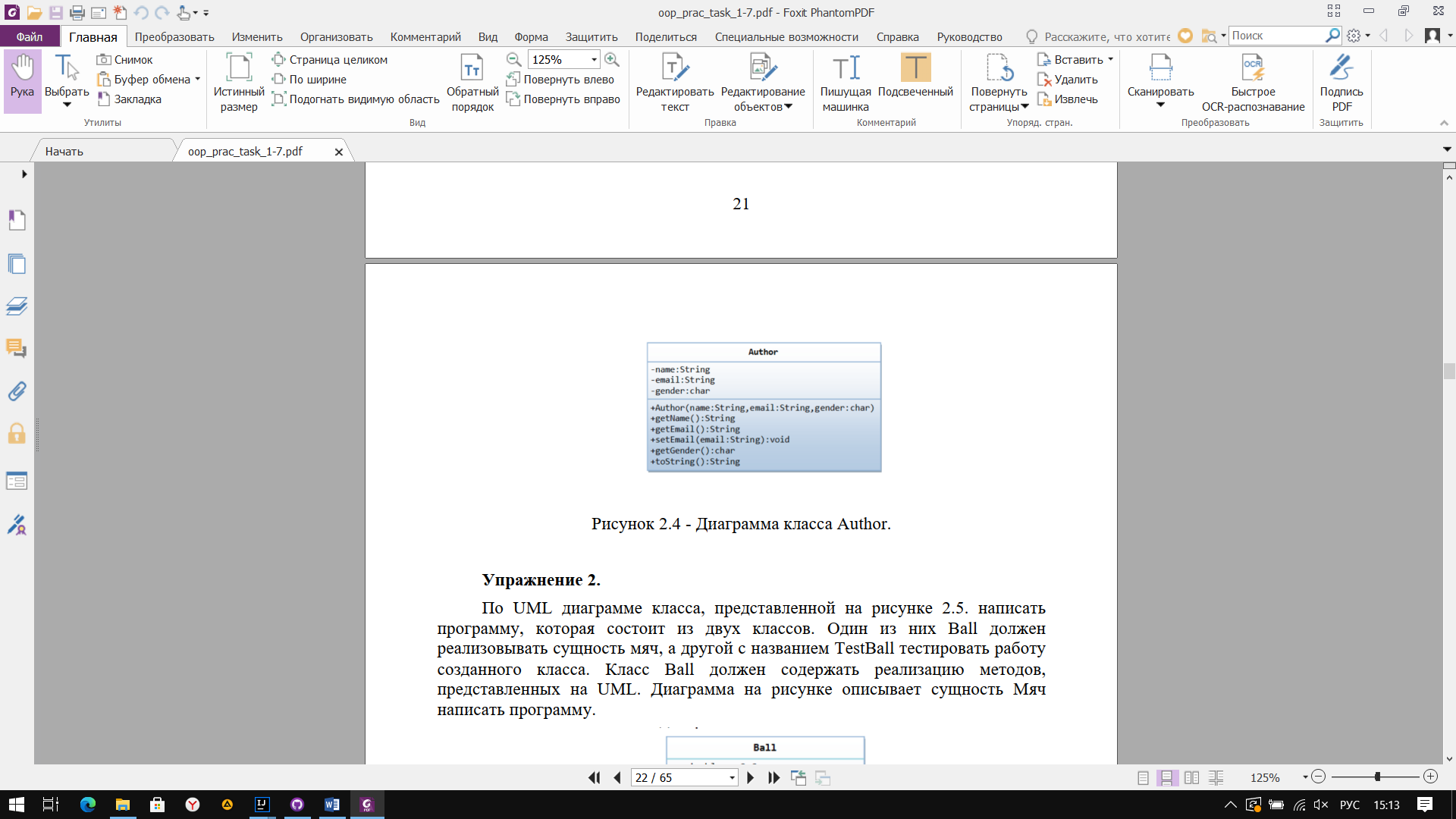


Рисунок 2.3 – Диаграмма класса Author.

Решение:

Создаём новый класс Author. В этом классе создаем поля для имени, пола и адреса электронной почты, а также конструктор и методы получения и установки для полей класса.

package ru.mirea.lab2;

import java.lang.\*;

public class Author {

private String name;

private String email;

private char gender;

public Author(String name, String email, char gender) {

this.name = name;

this.email = email;

this.gender = gender;

}

public String getName() {

return name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public char getGender() {

return gender;

}

public String toString() {

String genderForReturn;

if (gender == 'M') {

genderForReturn = "m";

} else if (gender == 'F') {

genderForReturn = "ms";

} else {

genderForReturn = "unknown";

}

return name + " (" + genderForReturn + ") at " + email;

}

}

Также создаём класс TestAuthor. В нём создаем метод main для работы программы, а также 2 объекта класса Author. С помощью метода toString() класса Author выводим информацию о каждом из объектов.

package ru.mirea.lab2;

import java.lang.\*;

public class TestAuthor {

public static void main(String[] args) {

Author a1 = new Author("Ivan Popov", "ivPopov@somewhere.com", 'M');

Author a2 = new Author("Anna Ivanova", "anIvanova@somewhere.com", 'F');

System.out.println(a1.toString());

System.out.print(a2.toString());

}

}

На рисунке 2.4 показан вывод программы. По нему можно убедится, что программа работает правильно.

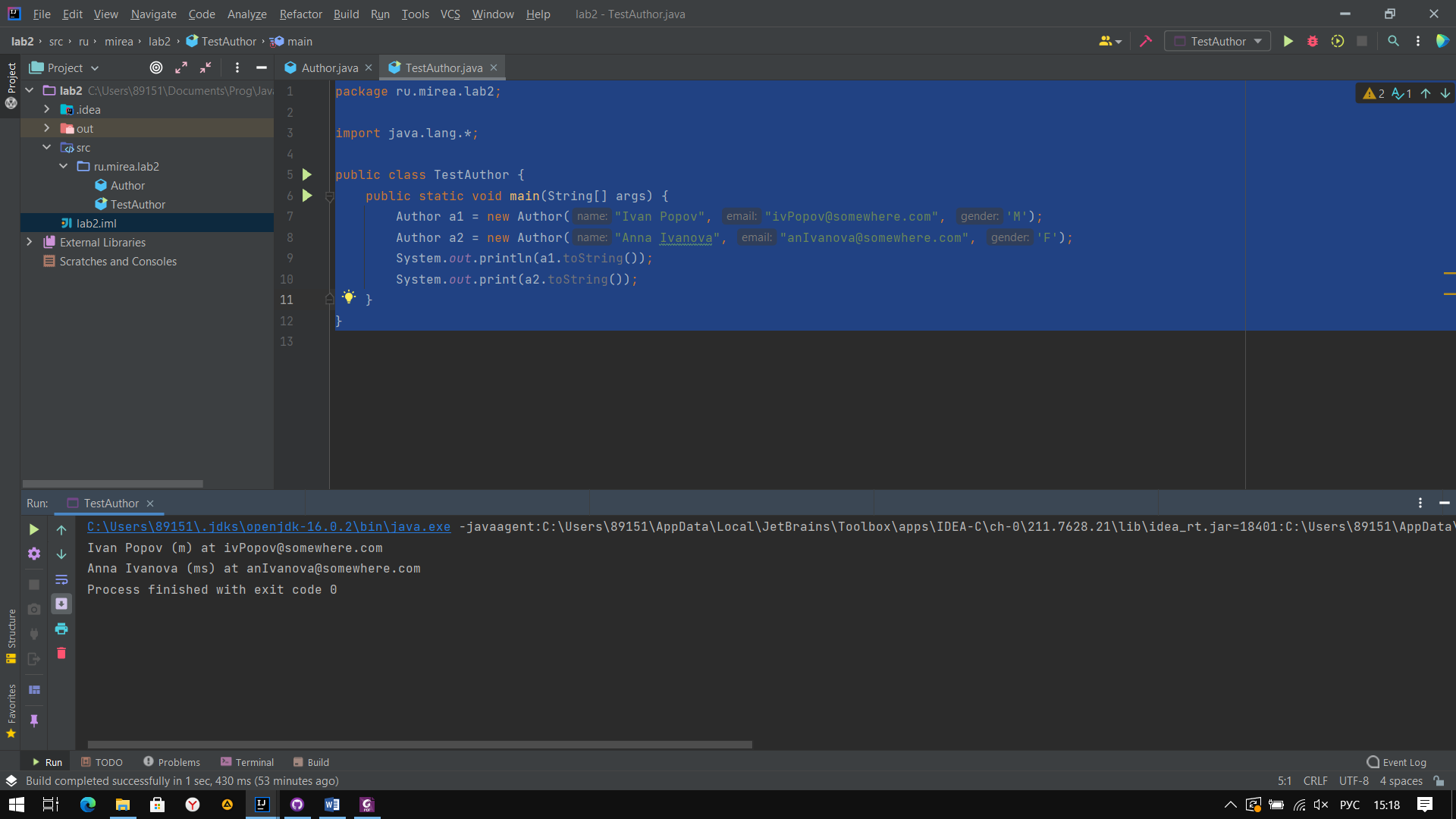


Рисунок 2.4 - Вывод программы.

**Выводы по работе:**

Выполняя данную практическую работу, я научился создавать класс, поля, методы и конструкторы в нем по диаграммам класса UML, а также обращаться к его полям и методам.

Практическая работа № 3

Цель работы

Изучение наследования, абстрактные суперклассы и их подклассы в Java.

Теоретическое введение

Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы при определении помечаются ключевым словом abstract. Абстрактный метод внутри абстрактного класса не имеет тела, только прототип. Он состоит только из объявления и не имеет тела: abstract void yourMethod();

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. 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.

Выполнение лабораторной работы

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 рисунке 3.1 Circle, Rectangle and Square.

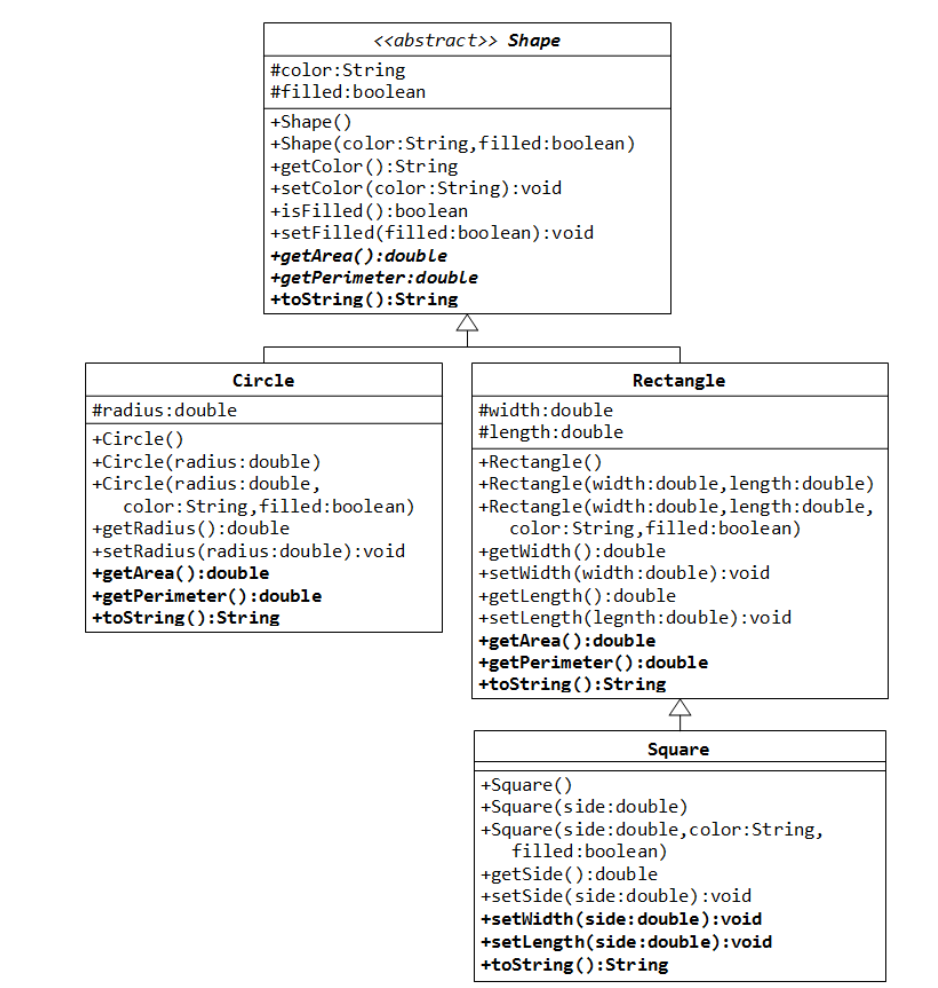


Рисунок 3.1 – Диаграмма реализации класса Shape.

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.

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());  
28  
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());

Упражнение 4.

Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable. Диаграмма реализации интерфейса показана на рисунке 3.2

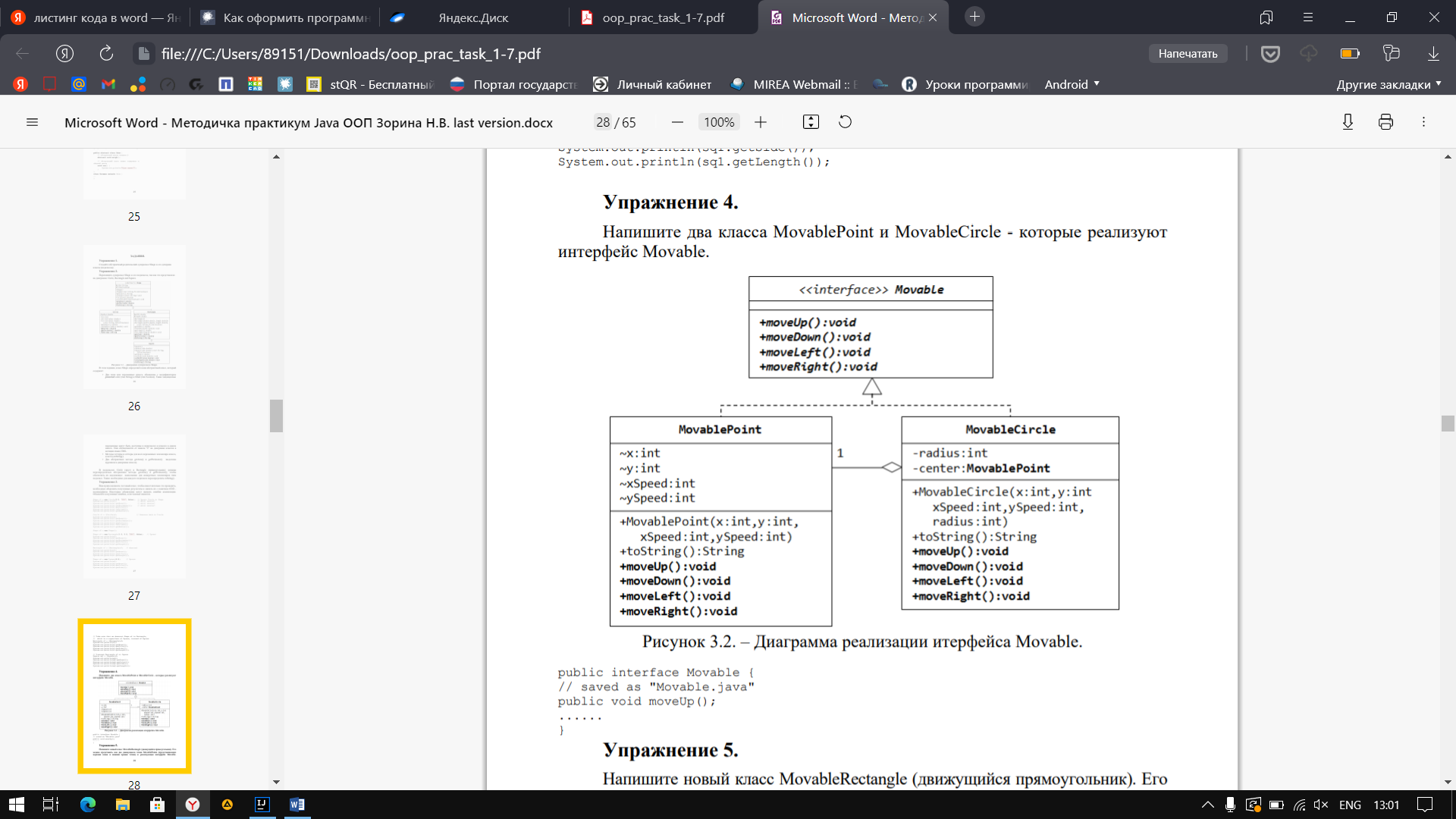


Рисунок 3.2 – Диаграмма реализации интерфейса Movable.

**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).

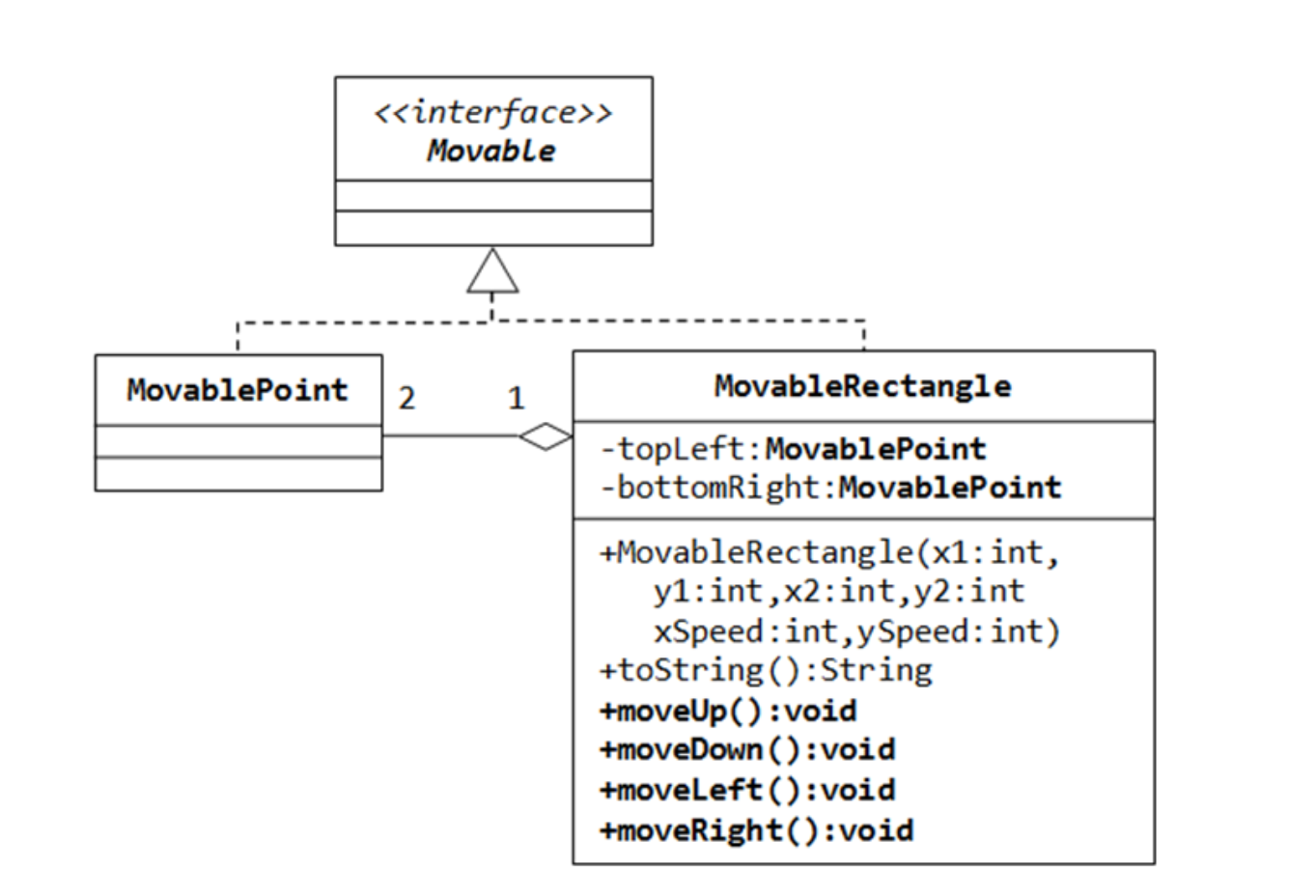


Рисунок 3.3 – Диаграмма класса MovableRectangle.

Решение:

Упражнение 1.

Создаём абстрактный класс Shape. В нём создаём конструктор по умолчанию, конструктор с параметрами цвета (строковый тип) и закрашенности (логический тип), а также методы получения и установки значений полей.

package ru.mirea.lab3;

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;

}

}

Упражнение 2.

Переписываем класс Shape и добавляем методы для вычисления периметра, площади и получения полной информации об объекте.

Создаём класс Circle (наследуется от класса Shape) и определяем в нём создаём конструктор по умолчанию, конструктор с параметрами цвета (строковый тип) и закрашенности (логический тип) и конструктор с параметрами радиуса окружности (вещественный тип), цвета (строковый тип) и закрашенности (логический тип). Создаём методы для получения и установки радиуса и переписываем методы для получения площади, периметра и полной информации об объекте.

package ru.mirea.lab3;

import java.lang.\*;

public class Circle extends Shape {

protected double radius;

public Circle() {

}

public Circle(String color, boolean filled) {

super(color, filled);

}

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 Math.PI \* radius \* 2;

}

@Override

public String toString() {

return "Circle: " +

"radius=" + radius +

", area = " + getArea() +

", perimeter =" + getPerimeter()+

", color = "+color+

", filled = "+filled+" }";

}

}

Создаём класс Rectangle (наследуется от класса Shape) и определяем в нём создаём конструктор по умолчанию, конструктор с параметрами длины (вещественный тип) и ширины (вещественный тип) и конструктор с параметрами длины (вещественный тип) и ширины (вещественный тип), цвета (строковый тип) и закрашенности (логический тип). Создаём методы для получения и установки длины и ширины и переписываем методы для получения площади, периметра и полной информации об объекте.

package ru.mirea.lab3;

import java.lang.\*;

public class Rectangle extends Shape {

protected double width;

protected double length;

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 void setWidth(double width) {

this.width = width;

}

public double getLength() {

return length;

}

public void setLength(double length) {

this.length = length;

}

@Override

public double getArea() {

return length \* width;

}

@Override

public double getPerimeter() {

return 2 \* (length + width);

}

@Override

public String toString() {

return "Rectangle{" +

"width = " + width +

", length = " + length +

", area = " + getArea() +

", perimeter = " + getPerimeter() +

", color = " + color +

", filled = " + filled+" }";

}

}

Создаём класс Square (наследуется от класса Rectangle) и определяем в нём создаём конструктор по умолчанию, конструктор с параметрами длины (вещественный тип) и конструктор с параметрами длины (вещественный тип), цвета (строковый тип) и закрашенности (логический тип). Создаём методы для получения и установки длины и переписываем методы для получения площади, периметра и полной информации об объекте.

package ru.mirea.lab3;

import java.lang.\*;

public class Square extends Rectangle {

public Square() {

}

public Square(double side) {

super(side, side);

}

public Square(double side, String color, boolean filled){

super(side, side, color, filled);

}

public double getSide(){

return super.length;

}

public void setSide(double side){

super.setLength(side);

}

@Override

public void setWidth(double width) {

super.setWidth(width);

}

@Override

public double getLength() {

return super.getLength();

}

@Override

public double getArea() {

return super.getArea();

}

@Override

public double getPerimeter() {

return super.getPerimeter();

}

@Override

public String toString() {

return "Square{" +

"width = " + width +

", length = " + length +

", area = " + getArea() +

", perimeter = " + getPerimeter() +

", color = " + color +

", filled = " + filled+" }";

}

}

Упражнение 3.

Также создаём тестовый класс TestShape. В нём создаем метод main для работы программы, создаём объекты разных классов и проверяем код из условия задачи.

package ru.mirea.lab3;

public class testShape {

public static void main(String[] args) {

Shape s1 = new Circle(5.5, "RED", false);

System.out.println(s1);

System.out.println(s1.getArea());

System.out.println(s1.getPerimeter());

System.out.println(s1.getColor());

System.out.println(s1.isFilled());

Circle c1 = (Circle) s1;

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 s3 = new Rectangle(1.0, 2.0, "RED", false);

System.out.println(s3);

System.out.println(s3.getArea());

System.out.println(s3.getPerimeter());

System.out.println(s3.getColor());

Rectangle r1 = (Rectangle) s3;

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);

System.out.println(s4);

System.out.println(s4.getArea());

System.out.println(s4.getColor());

Rectangle r2 = (Rectangle) s4;

System.out.println(r2);

System.out.println(r2.getArea());

System.out.println(r2.getColor());

System.out.println(r2.getLength());

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());

}

}

Строки System.out.println(s1.getRadius()), System.out.println(s3.getLength()), System.out.println(s4.getSide()) и System.out.println(r2.getSide()) вызывают ошибку компиляции из-за того, что в классе Shape нет методов getRadius(), getLength() и getSide() соответственно. Также строка Shape s2 = new Shape() вызывает ошибку из-за того, что класс Shape – абстрактный и на его основе не может быть создан объект.

На рисунке 3.4 показан вывод программы при проверке работоспособности кода из условия. По нему можно понять, что программа работает правильно.

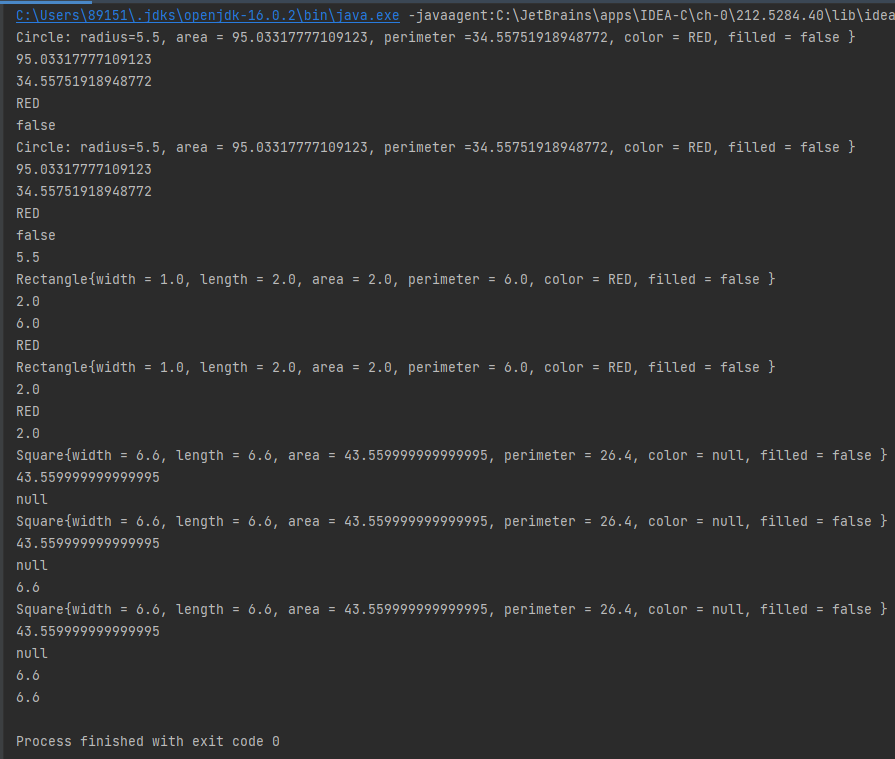


Рисунок 3.4 – Вывод программы.

Упражнение 4.

Сначала создаём интерфейс Movable и в нём методы для движения вверх, вниз, влево и вправо.

package ru.mirea.lab3;

public interface Movable {

public void moveUp();

public void moveDown();

public void moveLeft();

public void moveRight();

}

Затем создаём класс MovablePoint на основе интерфейса Movable. Он будет отвечать за движение точки. В нём создаем защищенные поля для хранения координат точки (целочисленный тип) и скорости их движения (целочисленный тип). Также определяем конструктор и переписываем методы движения и метод получения информации о точке.

package ru.mirea.lab3;

public class MovablePoint implements Movable {

protected int x, y, xSpeed, 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;

}

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 +

'}';

}

}

Наследуя класс MovablePoint мы создаём класс MovableCircle. В нём также создаём поля для хранения радиуса окружности, конструктор и переписываем методы движения и метод получения информации об окружности.

package ru.mirea.lab3;

public class MovableCircle extends MovablePoint {

private int radius;

private MovablePoint center;

public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {

super(x, y, xSpeed, ySpeed);

this.radius = radius;

center = new MovablePoint(this.x, this.y, this.xSpeed, this.ySpeed);

}

@Override

public String toString() {

return "MovableCircle: " +

"radius=" + radius +

", center=(" + center.x + " ; " + center.y + ")";

}

@Override

public void moveUp() {

center.moveUp();

}

@Override

public void moveDown() {

center.moveDown();

}

@Override

public void moveLeft() {

center.moveLeft();

}

@Override

public void moveRight() {

center.moveRight();

}

}

Создадим также класс TestMovableCircle для проверки работоспособности класса MovableCircle. В нём определим объект класса MovableCircle и выведем информацию о нём до начала перемещения и после.

package ru.mirea.lab3;

import java.lang.\*;

public class TestMovableCircle {

public static void main(String[] args) {

MovableCircle circle = new MovableCircle(3,5,1,1,5);

System.out.println(circle.toString());

circle.moveDown();

circle.moveLeft();

circle.moveDown();

circle.moveLeft();

System.out.println(circle.toString());

}

}

На рисунке 3.5 показан вывод программы при проверке работоспособности класса MovableCircle. По нему можно понять, что программа работает правильно.

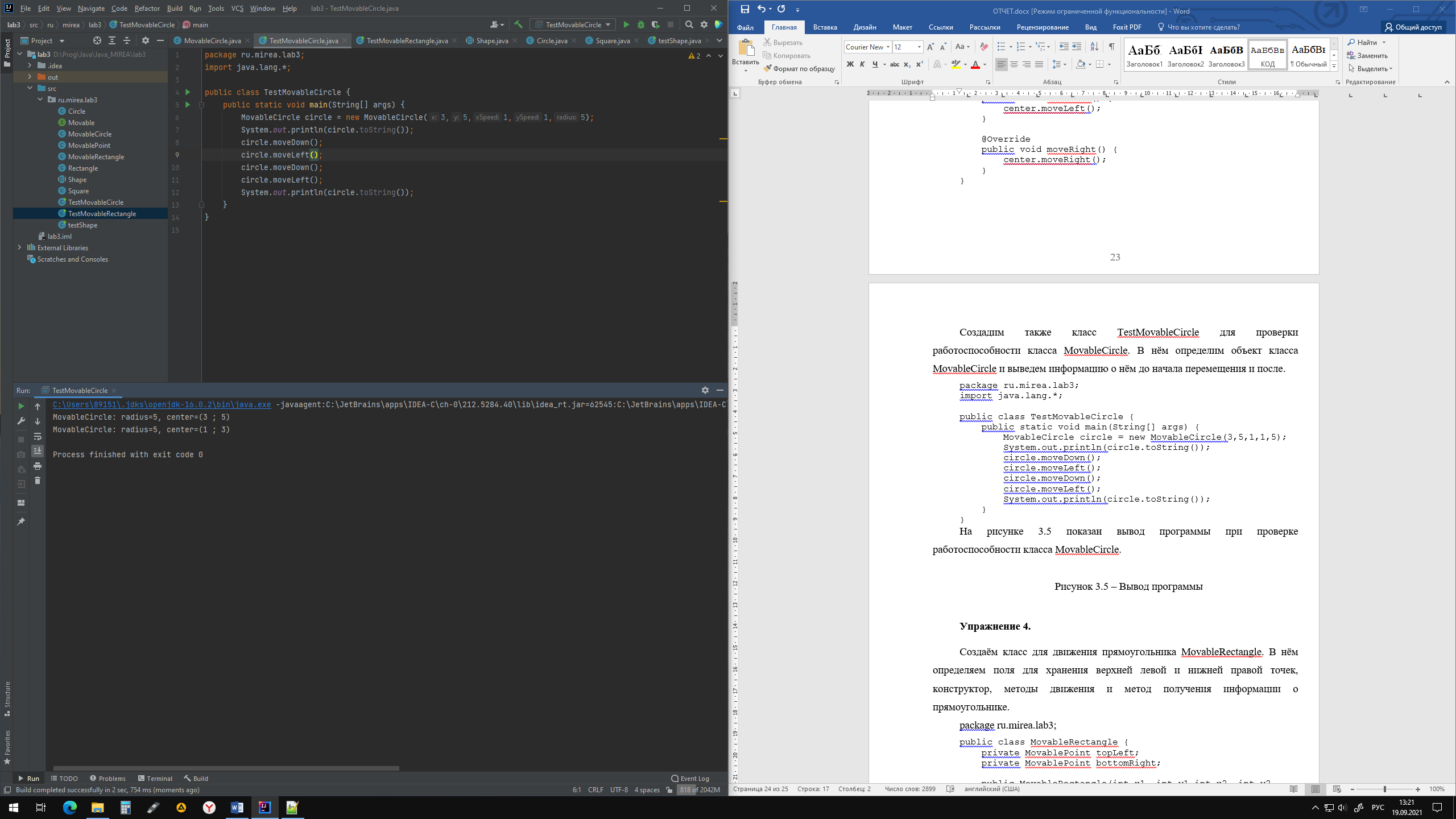


Рисунок 3.5 – Вывод программы.

Упражнение 4.

Создаём класс для движения прямоугольника MovableRectangle. В нём определяем поля для хранения верхней левой и нижней правой точек, конструктор, методы движения и метод получения информации о прямоугольнике.

package ru.mirea.lab3;

public class MovableRectangle {

private MovablePoint topLeft;

private MovablePoint 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 String toString() {

return "MovableCircle{ " +

"topLeft=(" + topLeft.x + " ; " + topLeft.y + ")" +

", bottomRight=(" + bottomRight.x + " ; " + bottomRight.y + ")}";

}

public void moveUp() {

bottomRight.moveUp();

topLeft.moveUp();

}

public void moveDown() {

bottomRight.moveDown();

topLeft.moveDown();

}

public void moveLeft() {

bottomRight.moveLeft();

topLeft.moveLeft();

}

public void moveRight() {

bottomRight.moveRight();

topLeft.moveRight();

}

}

Создадим также класс TestMovableRectangle для проверки работоспособности класса MovableRectangle. В нём определим объект класса MovableRectangle и выведем информацию о нём до начала перемещения и после.

package ru.mirea.lab3;

public class TestMovableRectangle {

public static void main(String[] args) {

MovableRectangle rectangle = new MovableRectangle(0,3,5,0,1,1);

System.out.println(rectangle.toString());

rectangle.moveDown();

rectangle.moveLeft();

rectangle.moveDown();

rectangle.moveLeft();

System.out.println(rectangle.toString());

}

}

На рисунке 3.6 показан вывод программы при проверке работоспособности класса MovableCircle. По нему можно понять, что программа работает правильно.

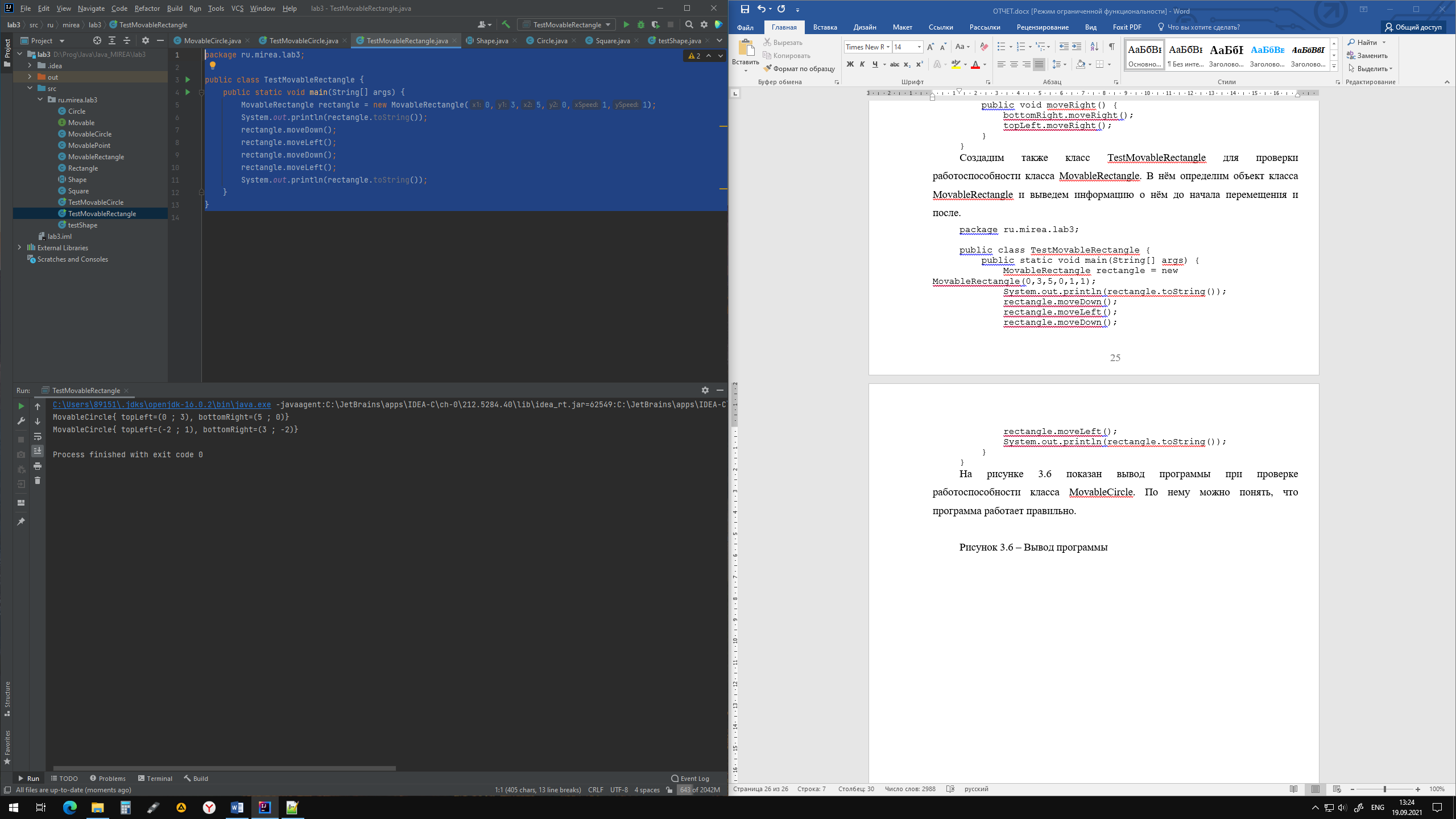


Рисунок 3.6 – Вывод программы.

**Выводы по работе:**

Выполняя данную практическую работу, я научился создавать абстрактный класс, наследовать от него другие классы, создавать интерфейс, описывать в нём необходимые для реализации в классах методы и создавать их.

Практическая работа № 4

Цель работы

Введение в событийное программирование на языке Java.

Теоретическое введение

Text Fields - текстовое поле или поля для ввода текста. Примерами текстовых полей являются поля для ввода логина и пароля, например, используемые, при входе в электронную почту.

Пример создания объекта класса JTextField:

JTextField jta = new JTextField (10);

В параметрах конструктора задано число 10, это количество символов, которые могут быть видны в текстовом поле. Текст веденный в поле JText может быть возвращен с помощью метода getText(). Также в поле можно записать новое значение с помощью метода setText(String s). Как и у других компонентов, мы можем изменять цвет и шрифт текста в текстовом поле.

Компонент TextArea похож на TextField, но в него можно вводить текст более одной строки. В качестве примера TextArea можно рассмотреть текст, который мы набираем в теле сообщения электронной почты.

**Создание меню.**

Добавление меню в программе Java проста. Java определяет три компонента для обработки:

* JMenuBar: который представляет собой компонент, который содержит меню.
* JMenu: который представляет меню элементов для выбора.
* JMenuItem: представляет собой элемент, который можно кликнуть из меню.

Выполнение лабораторной работы

Задание:

Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид. Создайте JFrame приложение у которого 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”;

Всякий раз, когда пользователь нажимает на кнопку AC Milan, результат будет увеличиваться для Милана, сначала 1 X 0, затем 2 X 0 и так далее. Last Scorer означает последнюю забившую команду. В этом случае: AC Milan. Если  
пользователь нажимает кнопку для команды Мадрид, то счет приписывается ей. Победителем становится команда, которая имеет больше кликов кнопку на соответствующую, чем другая.

Решение:

Для решения данной задачи создаём класс Frame, наследуемый от JFrame. Создаём объекты кнопок и надписей и два поля, отвечающих за хранение количества забитых мячей. Также создаём конструктор, в котором устанавливаем размер окна и его видимость. Добавляем кнопки и надписи в окно, а также добавляем слушатель для двух кнопок, по нажатию на которые должен увеличиваться счет и обновляться значения надписей на экране. Добавляем действие завершения программы при закрытии окна.

package ru.mirea.lab4;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Frame extends JFrame {

JButton AC\_Milan = new JButton("AC Milan");

JButton Real\_Madrid = new JButton("Real Madrid");

JLabel result = new JLabel("Result 0X0");

JLabel lastScorer = new JLabel("Last Scorer: N/A");

JLabel winner = new JLabel("Winner: DRAW");

Font font = new Font("Montserrat", Font.ITALIC, 20);

int AC\_Milan\_Score = 0;

int Real\_Madrid\_Score = 0;

Frame() {

super("Score counter");

setLayout(new FlowLayout());

setSize(400, 400);

setVisible(true);

JPanel buttons = new JPanel();

buttons.setLayout(new BoxLayout(buttons, BoxLayout.X\_AXIS));

AC\_Milan.setFont(font);

Real\_Madrid.setFont(font);

buttons.add(AC\_Milan);

buttons.add(Real\_Madrid);

add(buttons);

JPanel results = new JPanel();

results.setLayout(new BoxLayout(results, BoxLayout.Y\_AXIS));

result.setFont(font);

lastScorer.setFont(font);

winner.setFont(font);

results.add(result);

results.add(lastScorer);

results.add(winner);

add(results);

AC\_Milan.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

AC\_Milan\_Score++;

resultUpdater("AC Milan");

}

});

Real\_Madrid.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

Real\_Madrid\_Score++;

resultUpdater("Real Madrid");

}

});

super.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

private void resultUpdater(String goal) {

result.setText("Result: " + AC\_Milan\_Score + "X" + Real\_Madrid\_Score);

lastScorer.setText("Last Scorer: " + goal);

if (AC\_Milan\_Score > Real\_Madrid\_Score) winner.setText("Winner: AC Milan");

else if (AC\_Milan\_Score < Real\_Madrid\_Score) winner.setText("Winner: Real Madrid");

else winner.setText("Winner: DRAW");

}

}

Далее создаём класс TestFrame, в котором просто создаём объект класса Frame для проверки работоспособности нашего окна.

package ru.mirea.lab4;

public class TestFrame {

public static void main(String[] args) {

Frame frame = new Frame();

}

}

На рисунке 4.1 показан экран при запуске программы

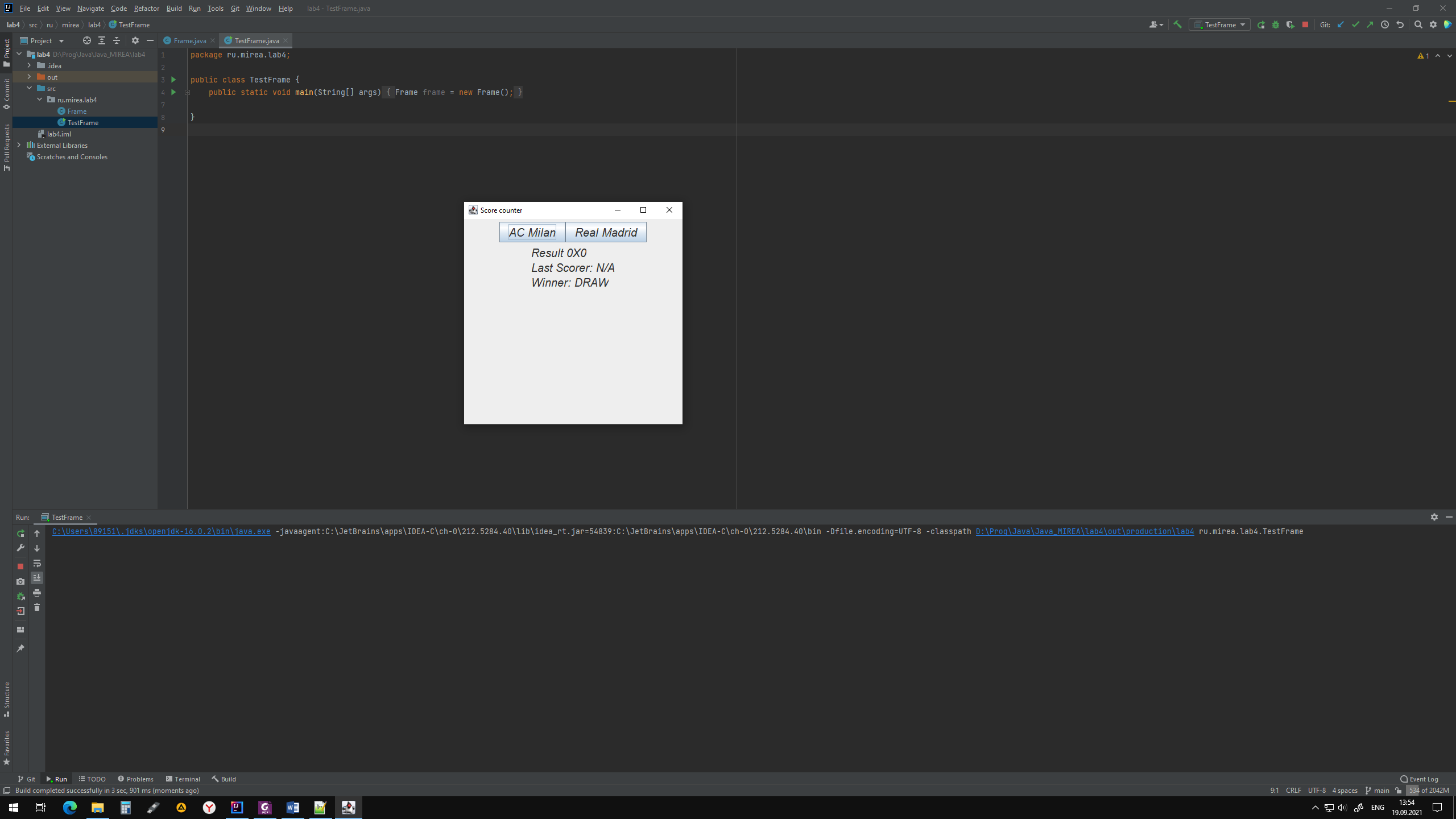


Рисунок 4.1 – Окно программы

При нажатии на кнопки значения в полях окна будут меняться, например, как на рисунке 4.2. По нему видно, что программа работает правильно.

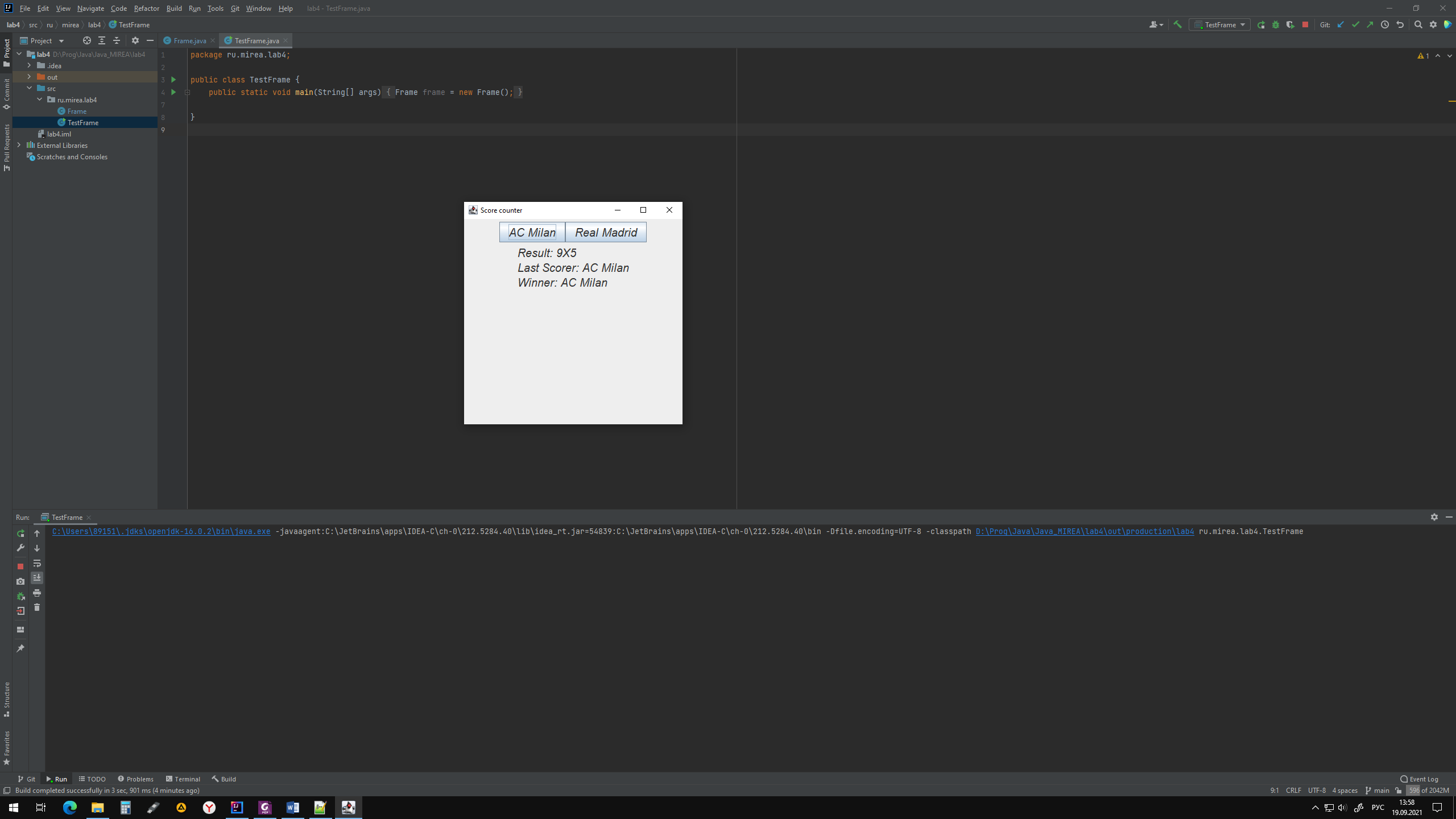


Рисунок 4.2 – Окно программы с измененным счётом

**Выводы по работе:**

Выполняя данную практическую работу, я научился создавать графический интерфейс, наполнять его кнопками и текстовыми полями, а также обрабатывать нажатия на кнопки.