|  |  |
| --- | --- |
|  |  |
| 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](#_Toc84874940)

[Практическая работа № 2 7](#_Toc84874941)

[Практическая работа № 3 11](#_Toc84874942)

[Практическая работа № 4 27](#_Toc84874943)

[Практическая работа № 5 32](#_Toc84874944)

[Практическая работа № 6 37](#_Toc84874945)

[Практическая работа № 7 43](#_Toc84874946)

[Практическая работа № 18 48](#_Toc84874947)

[Практическая работа № 19 52](#_Toc84874948)

[Практическая работа № 20 56](#_Toc84874949)

[Практическая работа № 21 61](#_Toc84874950)

[Практическая работа № 22 64](#_Toc84874951)

[Практическая работа № 23 67](#_Toc84874952)

[Практическая работа № 24 72](#_Toc84874953)

[Практическая работа № 25 78](#_Toc84874954)

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.

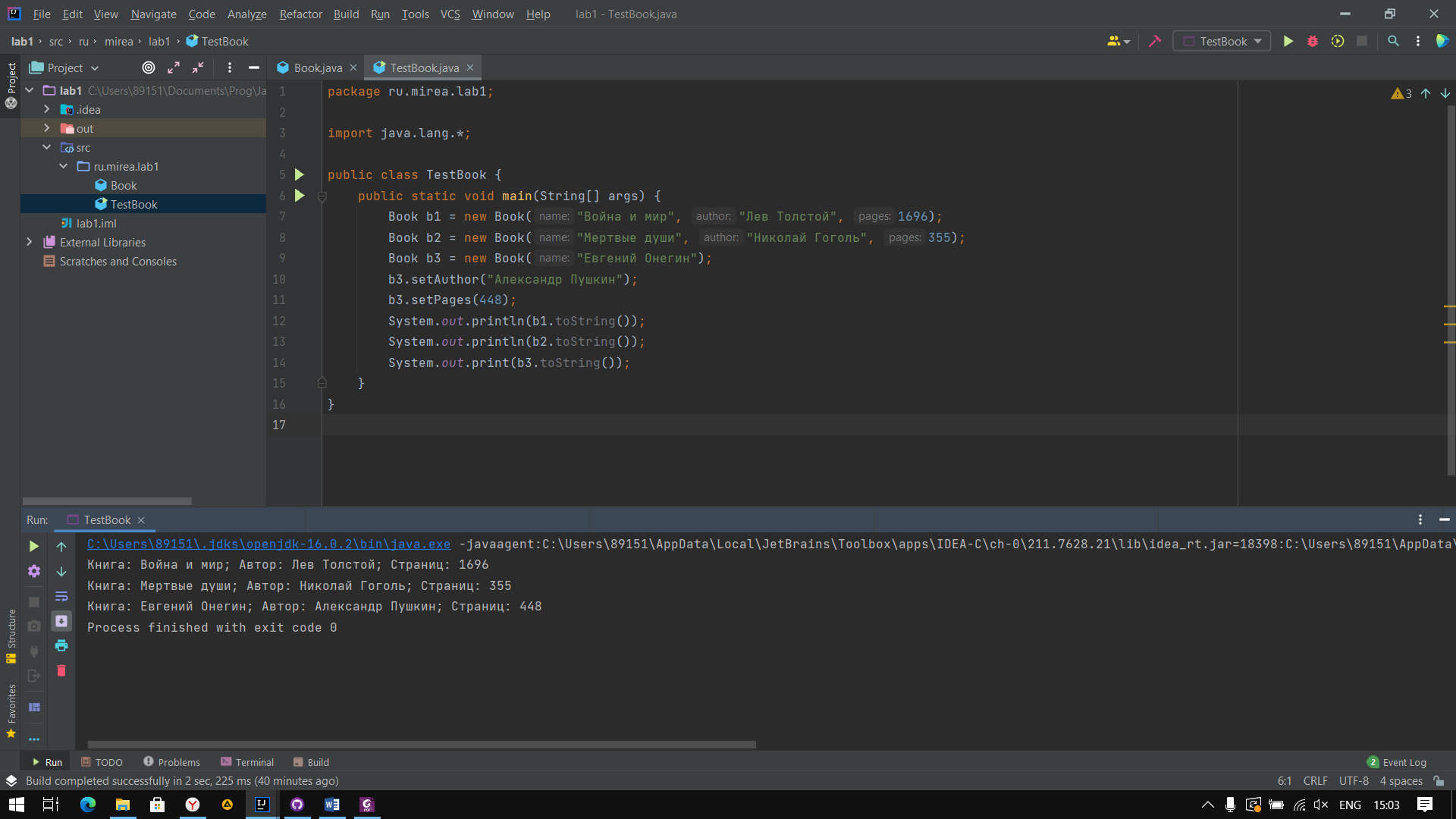


Рисунок 1.1 - Вывод программы.

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.

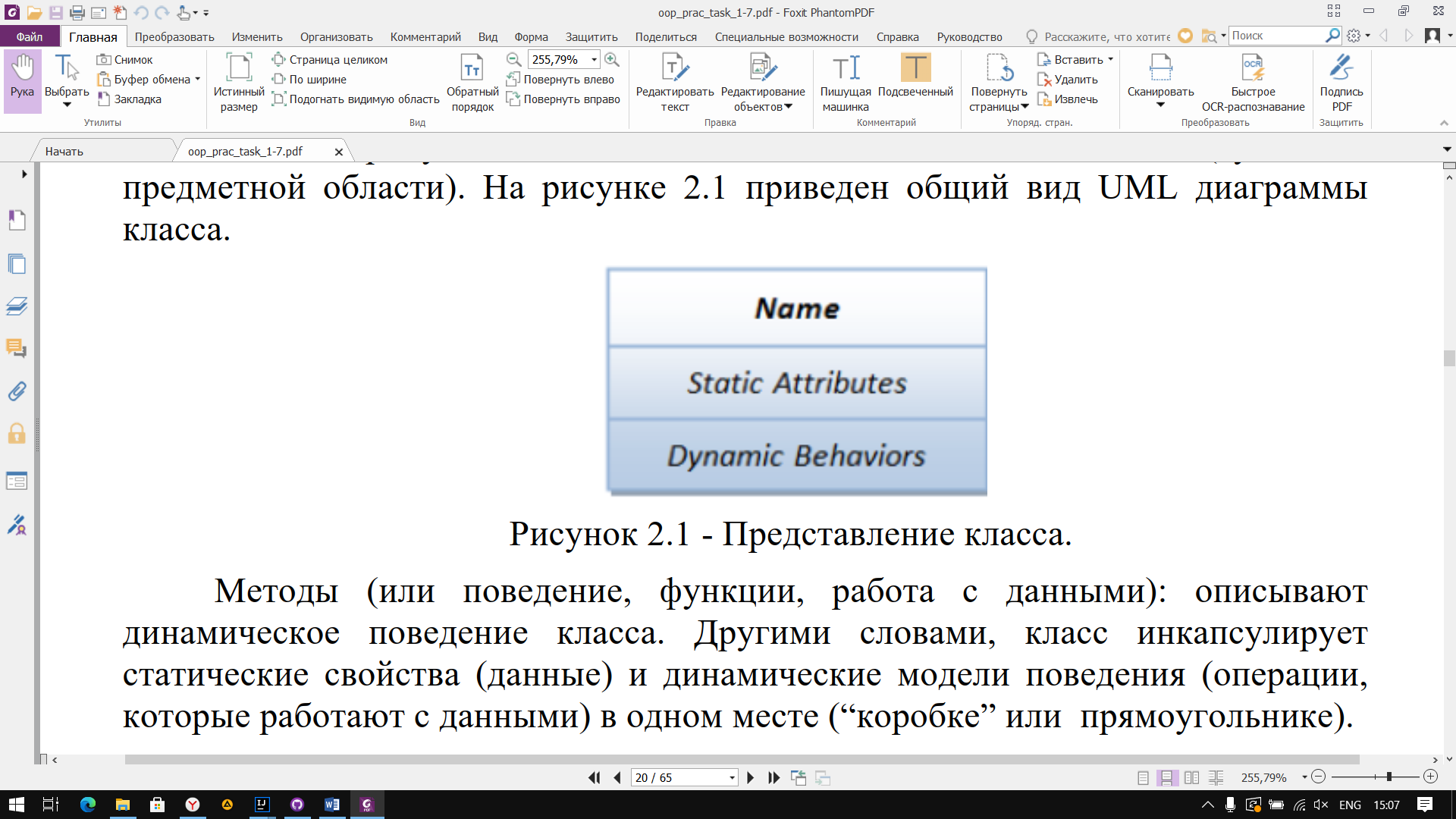


Рисунок 2.1 - Представление класса.

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

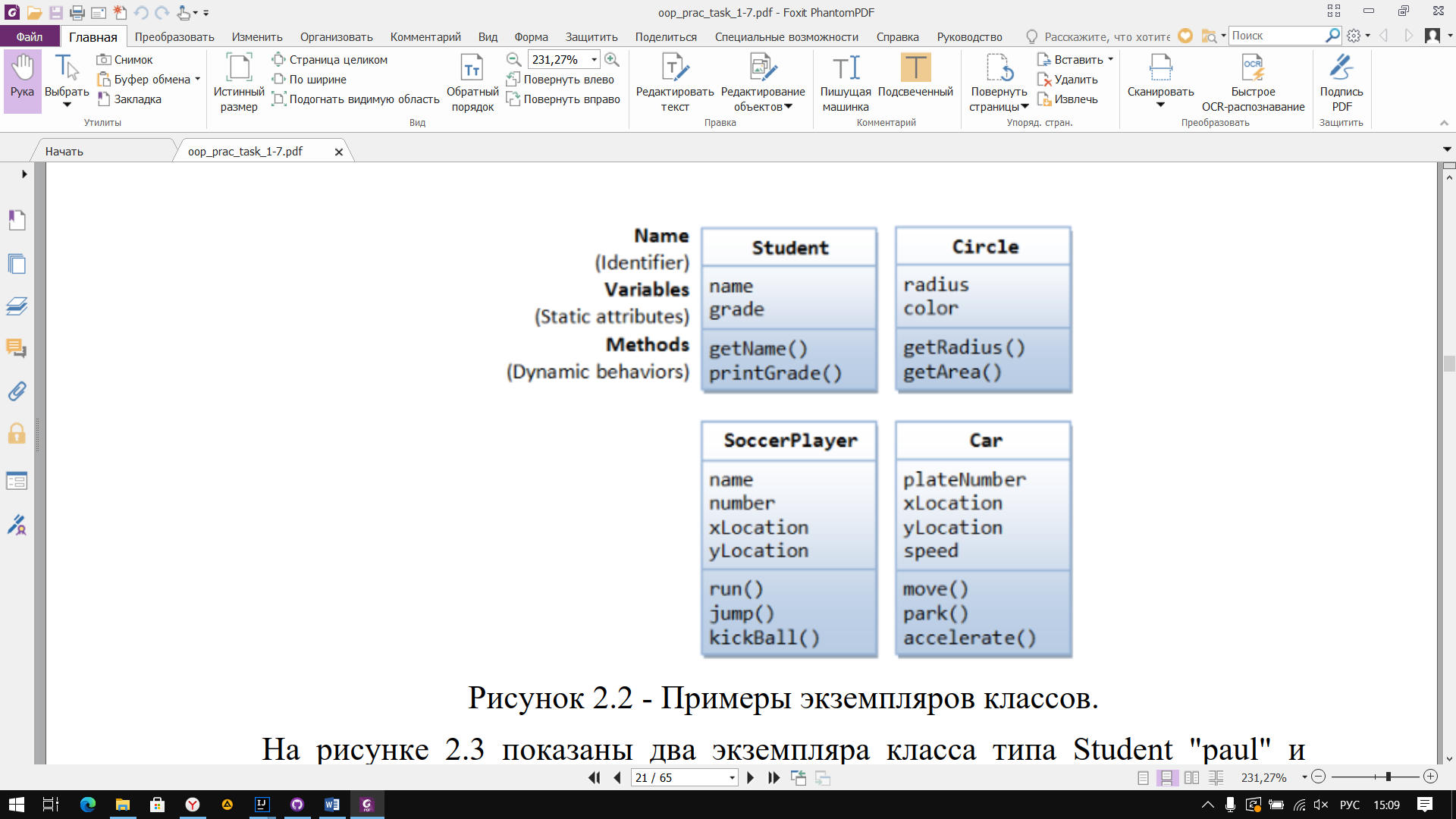


Рисунок 2.2 - Примеры экземпляров классов.

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.

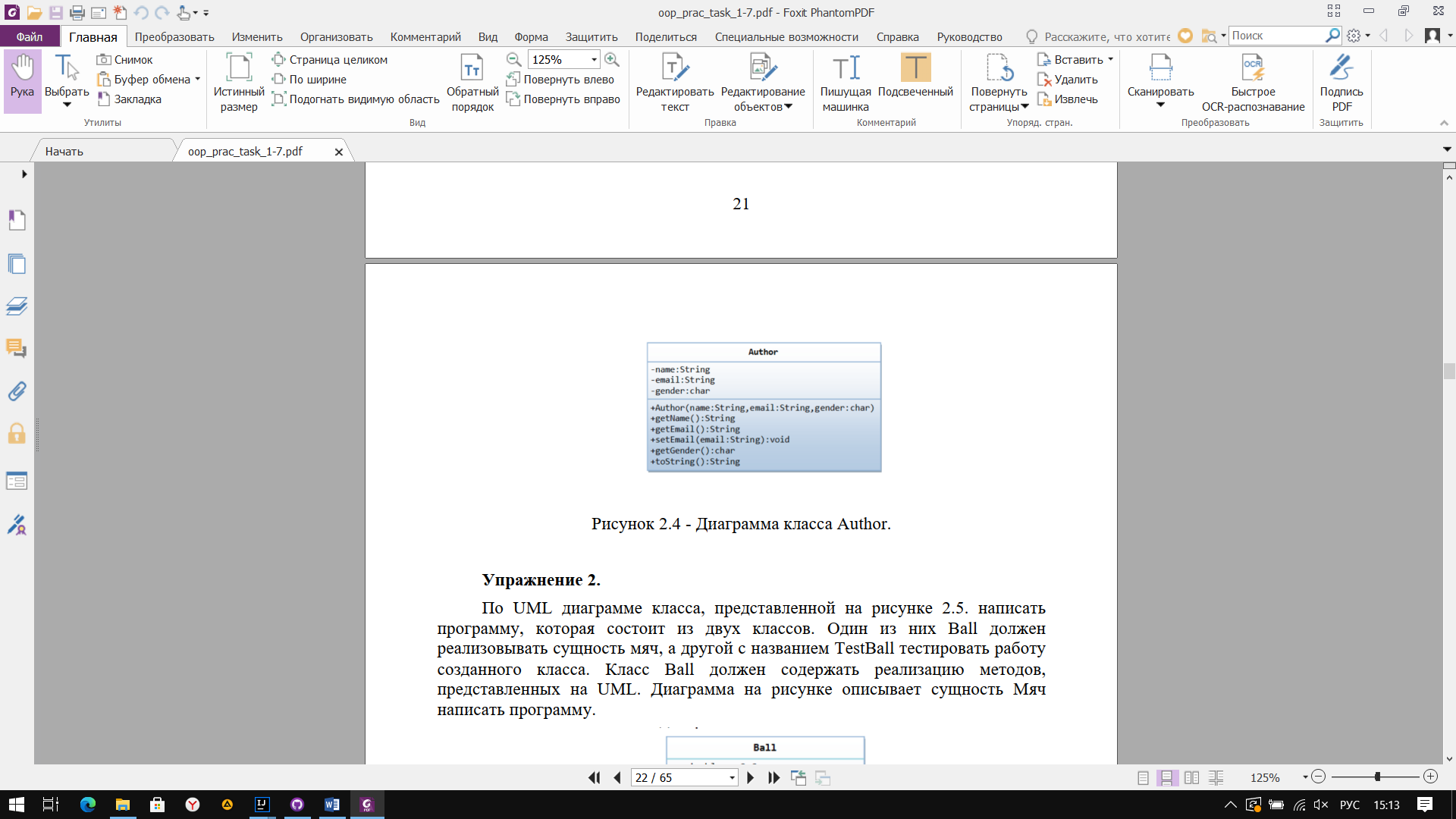


Рисунок 2.3 – Диаграмма класса Author.

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.

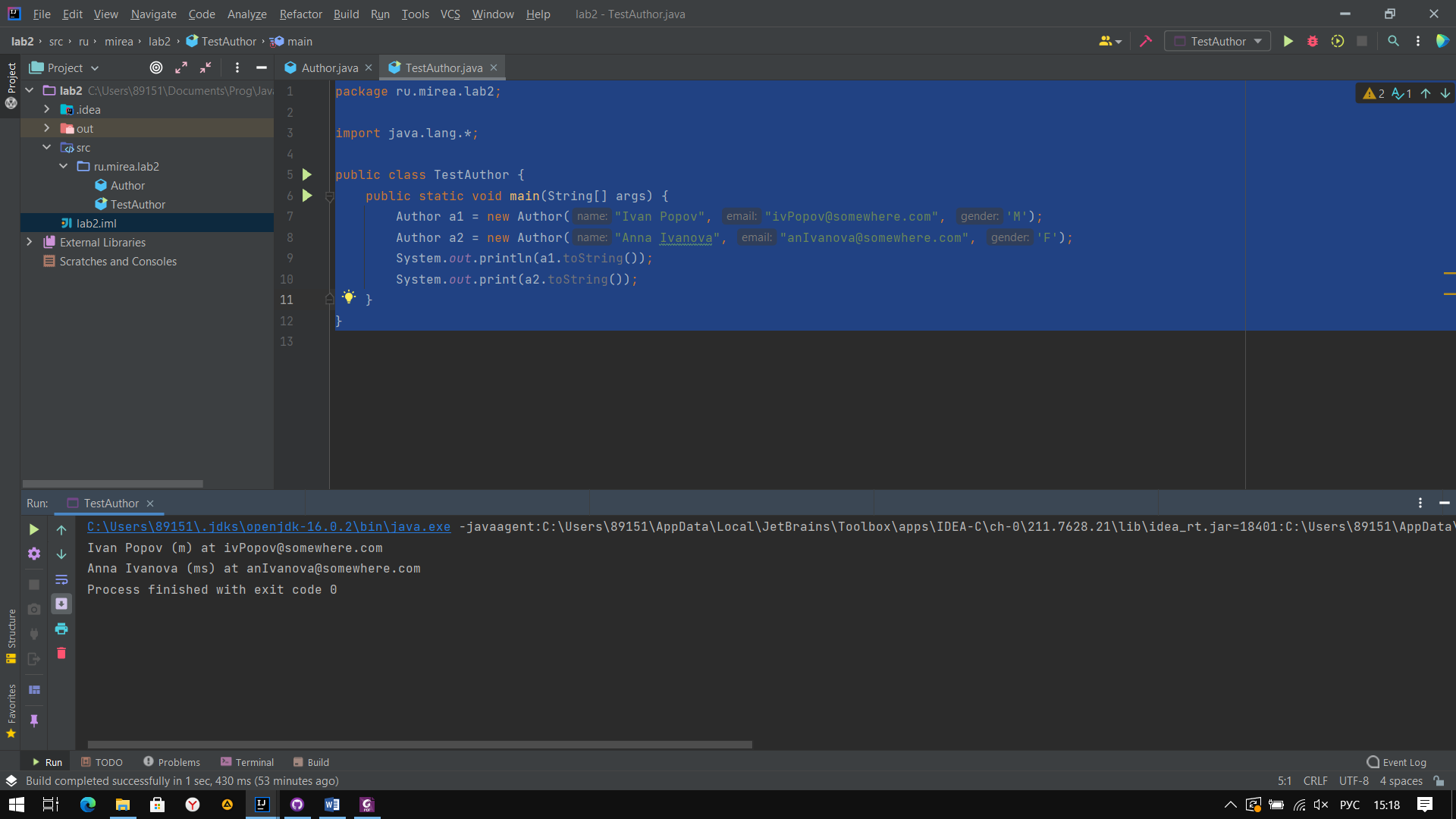


Рисунок 2.4 - Вывод программы.

**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.

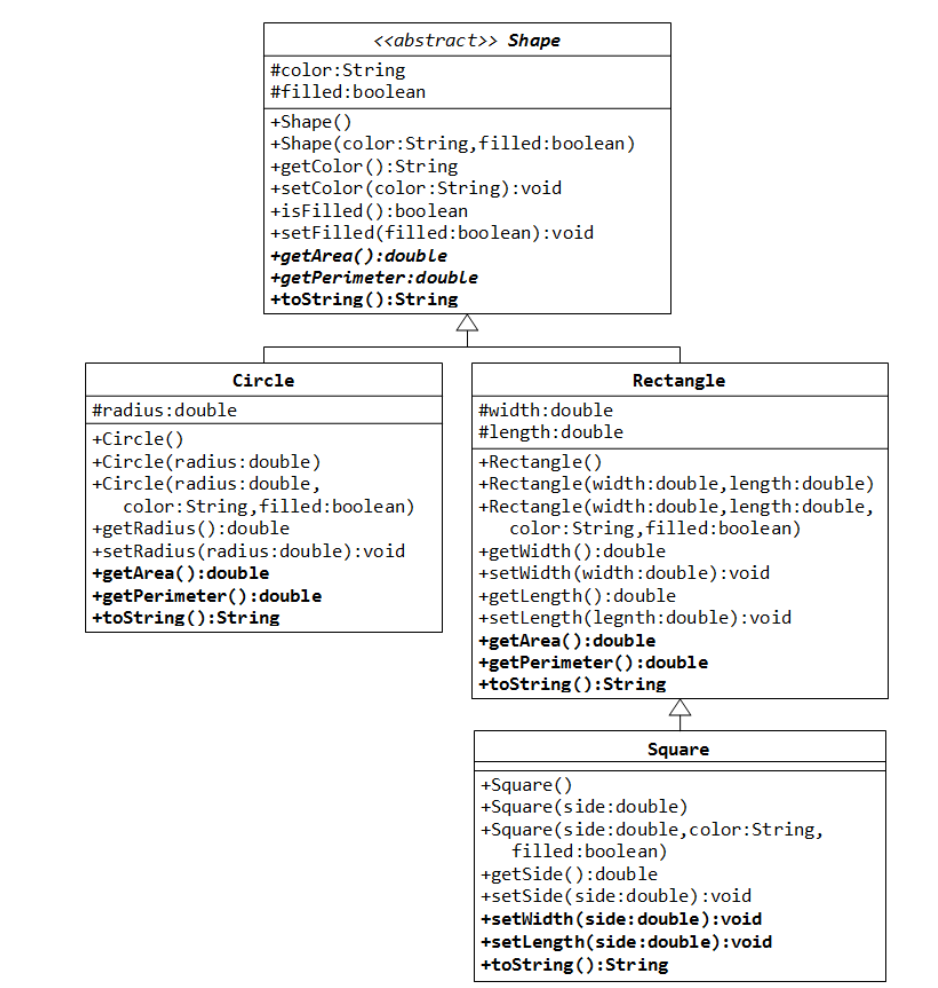


Рисунок 3.1 – Диаграмма реализации класса Shape.

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

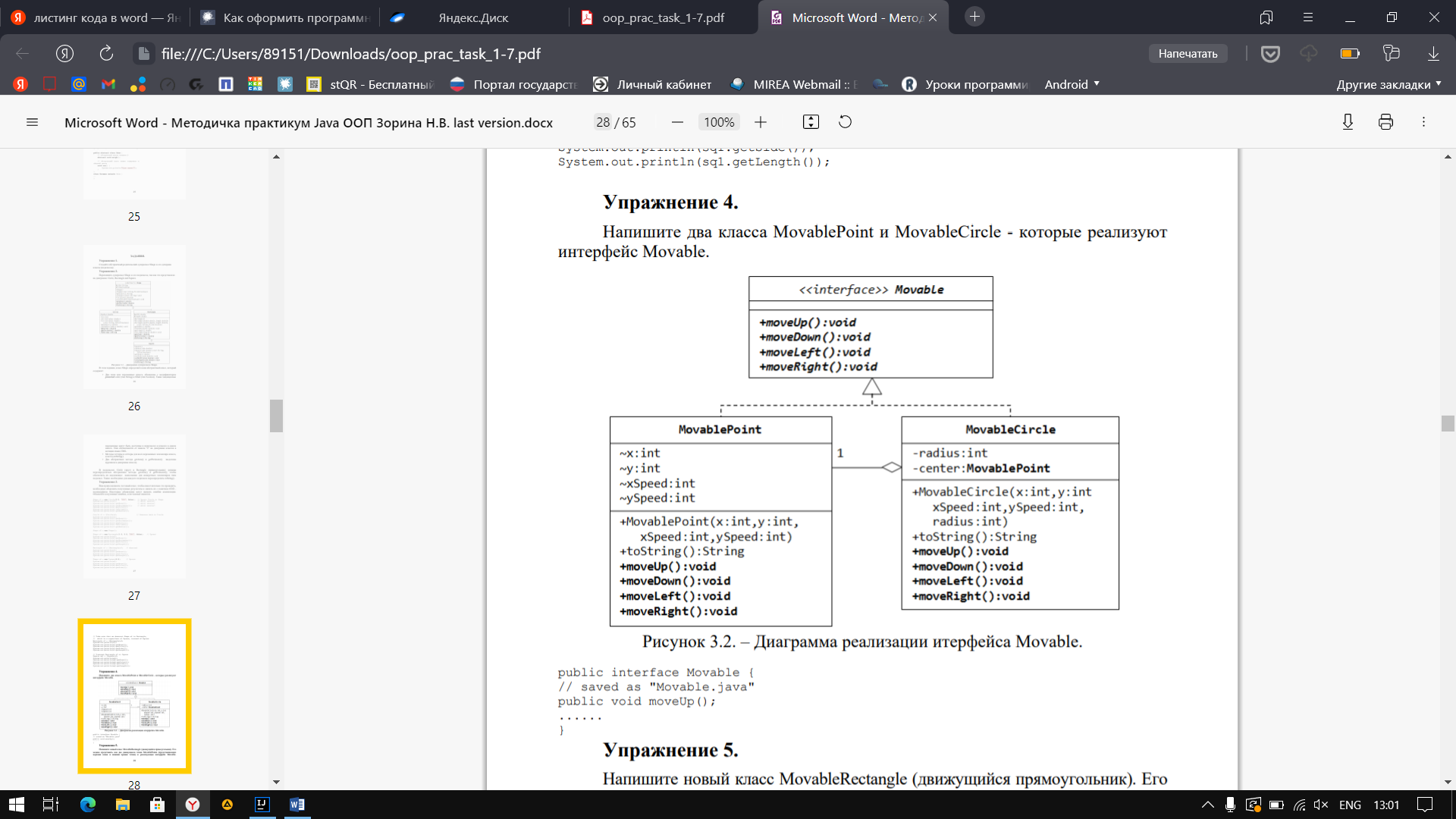


Рисунок 3.2 – Диаграмма реализации интерфейса Movable.

**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).

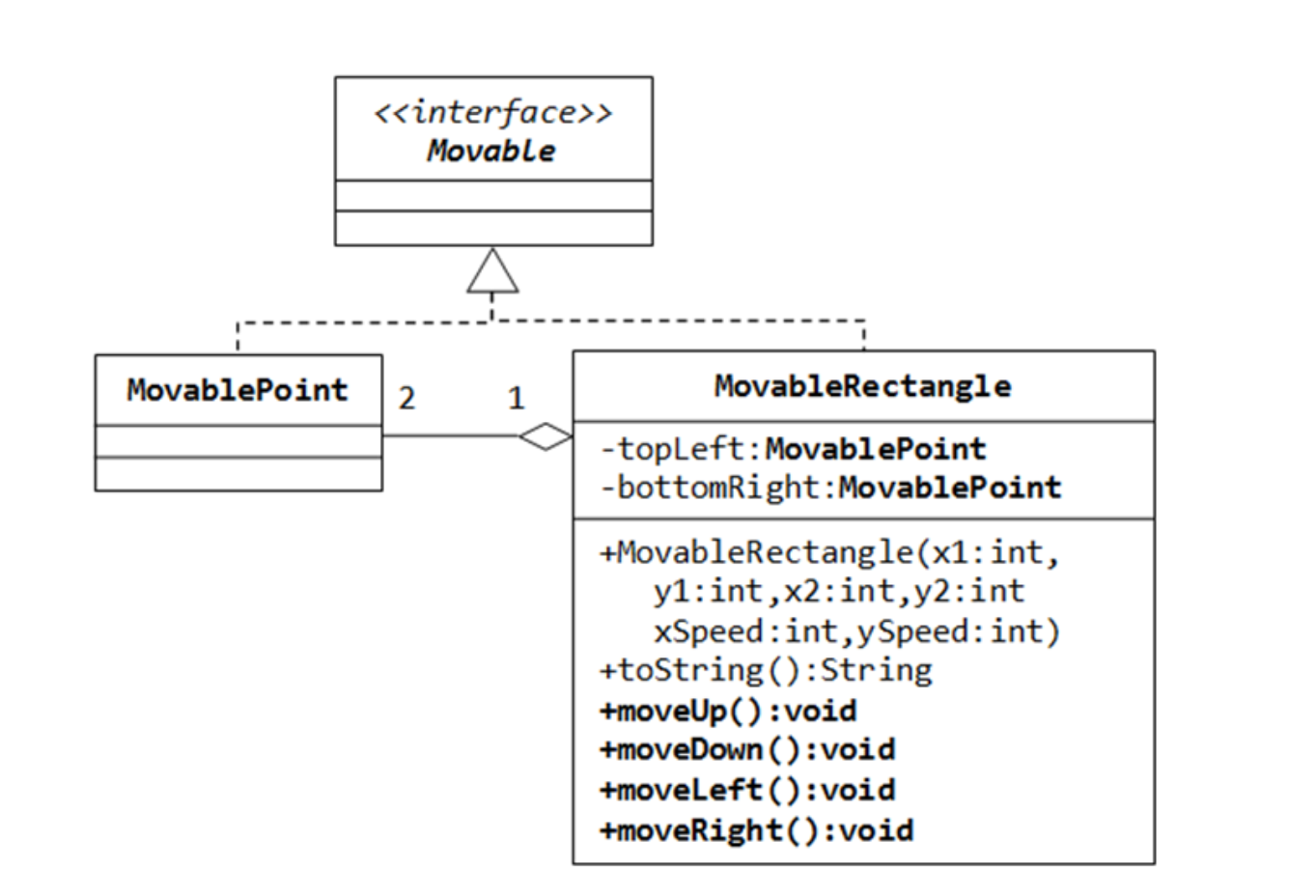


Рисунок 3.3 – Диаграмма класса MovableRectangle.

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.

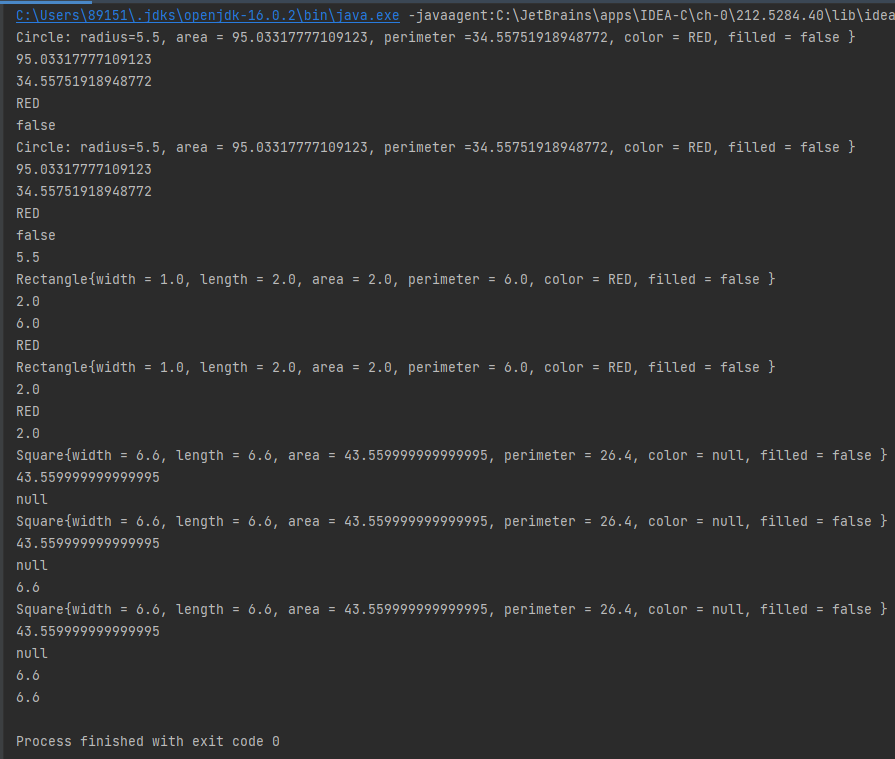


Рисунок 3.4 – Вывод программы.

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.

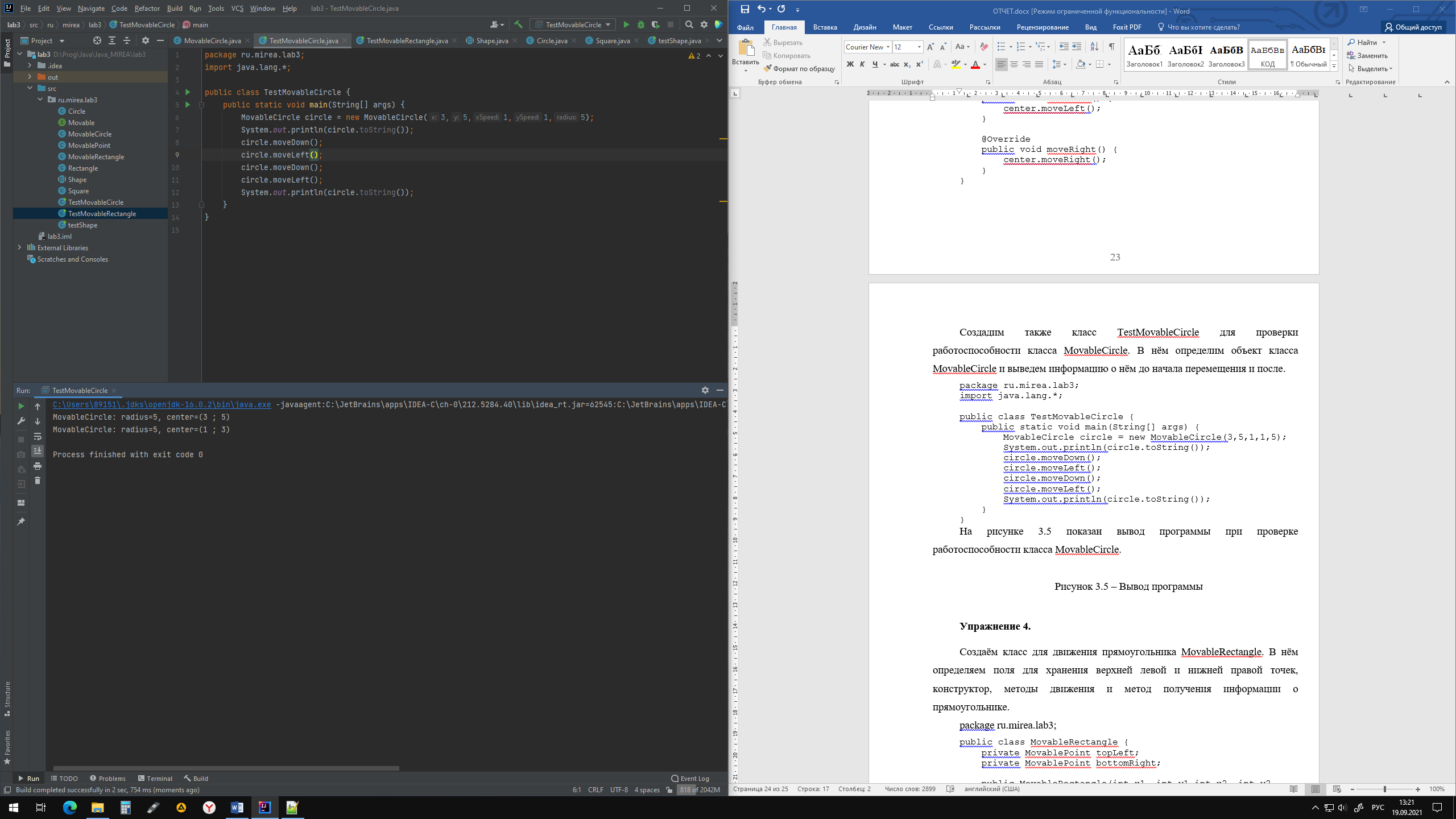


Рисунок 3.5 – Вывод программы.

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.

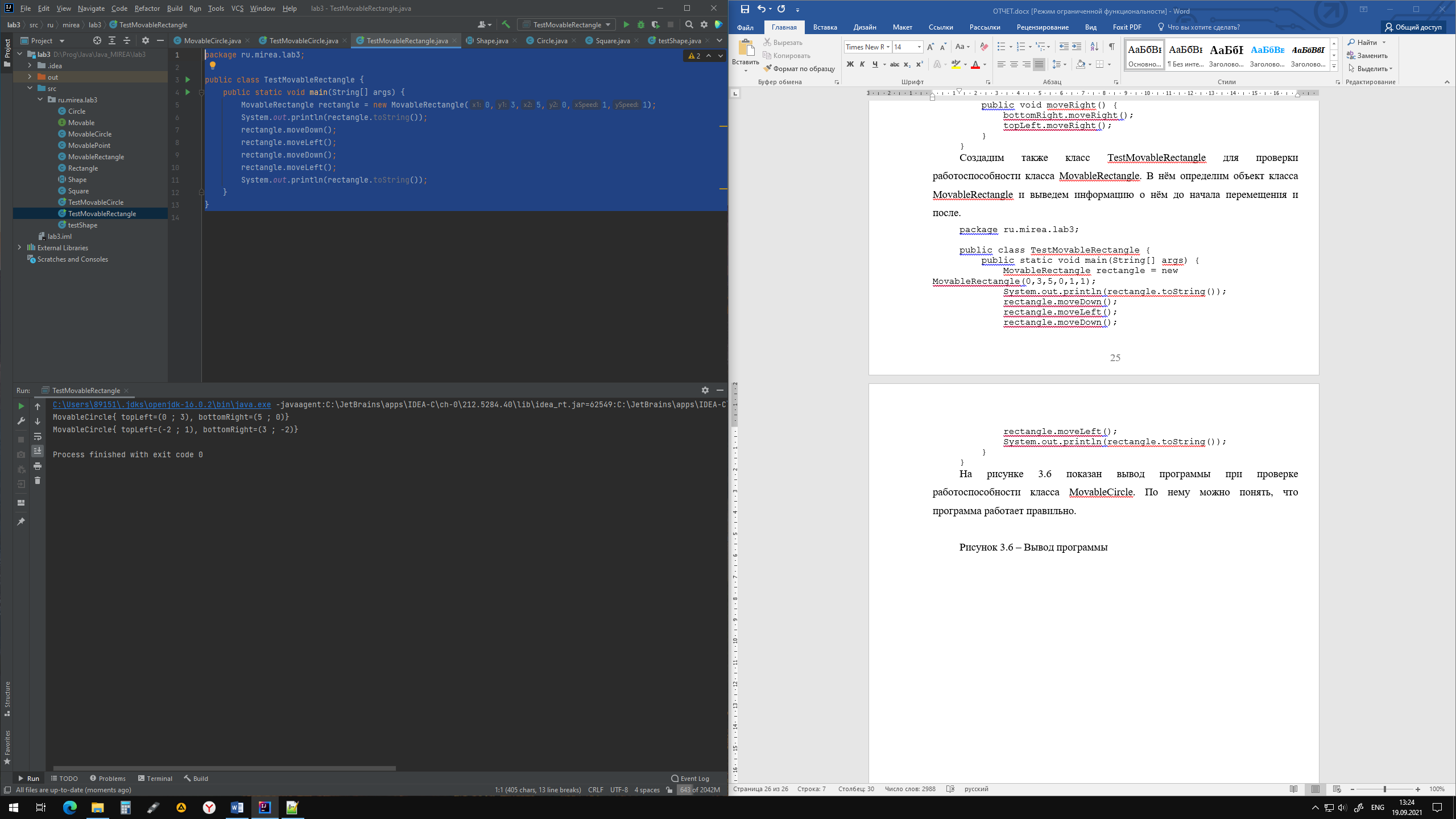


Рисунок 3.6 – Вывод программы.

**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

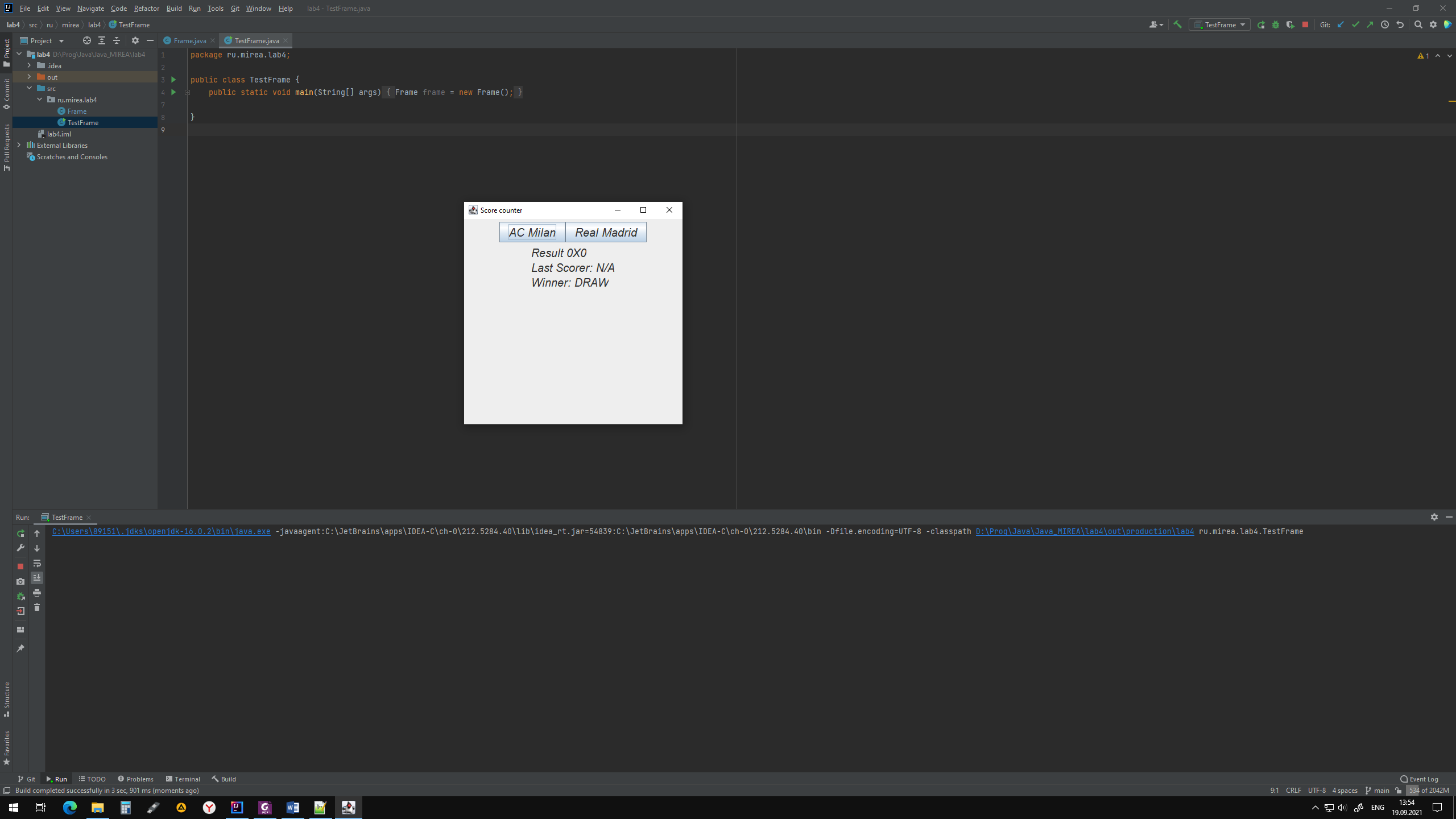


Рисунок 4.1 – Окно программы

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

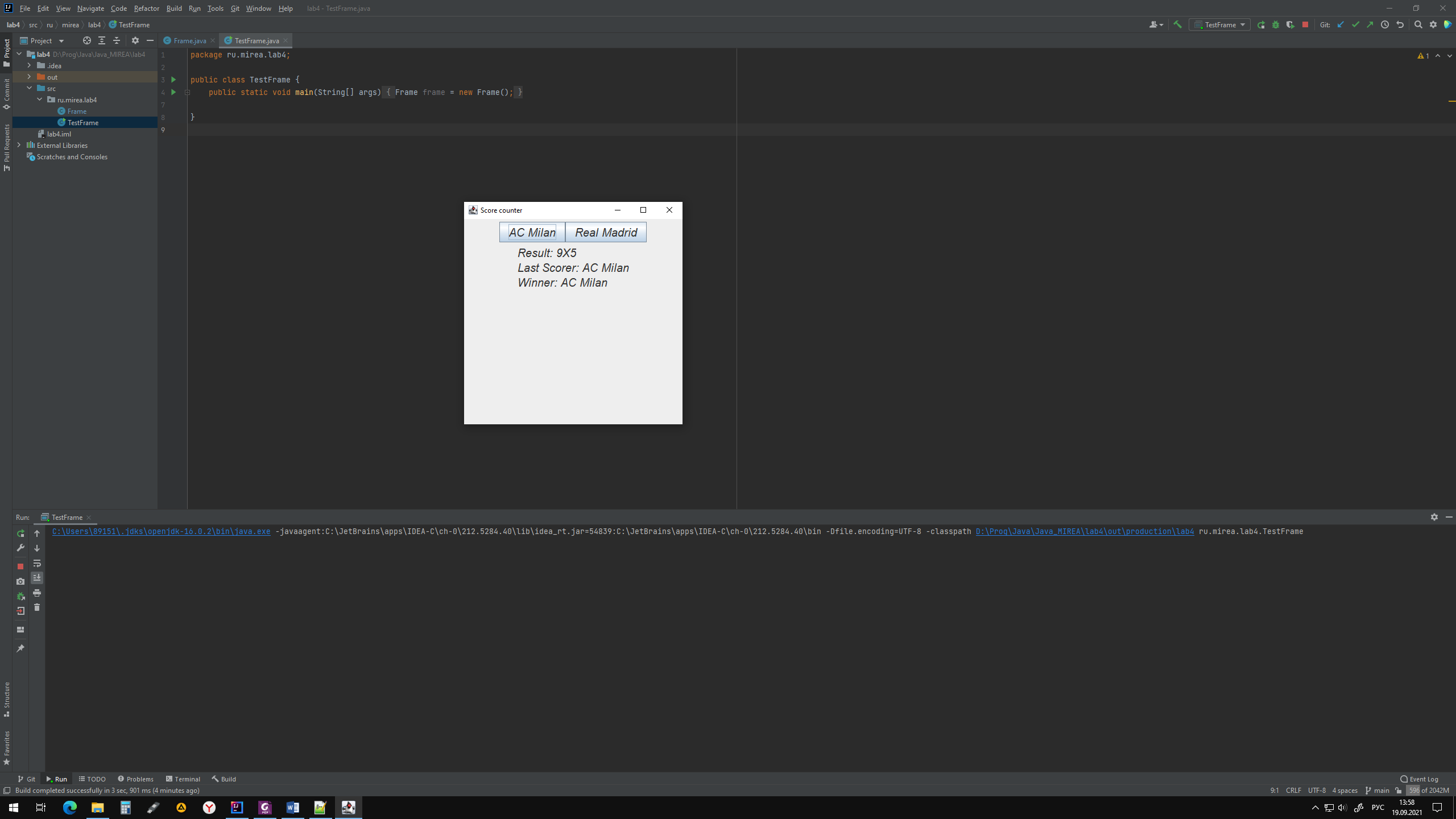


Рисунок 4.2 – Окно программы с измененным счётом

**Выводы по работе:**

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

Практическая работа № 5

Цель работы

Разработка и программирование рекурсивных алгоритмов на языке Java.

Теоретическое введение

Рекурсия — это некий активный метод**,** вызываемый сам по себе непосредственно, иливызываемой другим методом косвенно.Рекурсия — это своего рода перебор. Вообще говоря, всё то,что решается итеративно можно решить рекурсивно, то есть с использованиемрекурсивной функции.Так же, как и у перебора (цикла) у рекурсии должно быть условиеостановки — базовый случай. Это условие и является тем случаем, к которому рекурсия идет(шаг рекурсии). При каждом шаге вызывается рекурсивная функция до тех пор,пока при следующем вызове не сработает базовое условие и не произойдетостановка рекурсии (а точнее возврат к последнему вызову функции). Всёрешение сводится к поиску решения для базового случая. В случае, когдарекурсивная функция вызывается для решения сложной задачи (не базовогослучая) выполняется некоторое количество рекурсивных вызовов или шагов, сцелью сведения задачи к более простой. И так до тех пор, пока не получимбазовое решение.Итак, рекурсивная функция состоит из:

условия остановки или же базового случая или условия;

условия продолжения или шага рекурсии — способ сведения сложной задачи к более простым подзадачам.

Выполнение лабораторной работы

Задание:

Упражнение 1

Даны числа a и b. Определите, сколько существует последовательностей из a нулей и b единиц, в которых никакие два нуля не стоят рядом.

Упражнение 2

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Выведите все нечетные числа из этой последовательности, сохраняя их порядок. В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции.

Упражнение 3

Дано натуральное число N. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами или новыми строками. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика

Решение:

Упражнение 1

Создаём класс Main и функцию main для запуска программы. В ней мы считываем два числа от пользователя и вызываем рекурсивную функцию. В процессе работы она либо вызывает саму себя, либо возвращает нуль, если разность между параметрами больше одного, либо возвращает один, если один из параметров равен нулю.

package ru.mirea.lab5\_9;

import java.lang.\*;

import java.util.Scanner;

public class Main {

public static int counter(int a, int b) {

if (a - b > 1) return 0;

if (a == 0 || b == 0) return 1;

return counter(a, b - 1) + counter(a - 1, b - 1);

}

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

int a = scan.nextInt();

int b = scan.nextInt();

System.out.println(counter(a, b));

}

}

На рисунке 5.1 показан вывод программы. Если число единиц и нулей равно по одному, то мы получим два варианта чисел (10 или 01). Значит всё работает правильно.

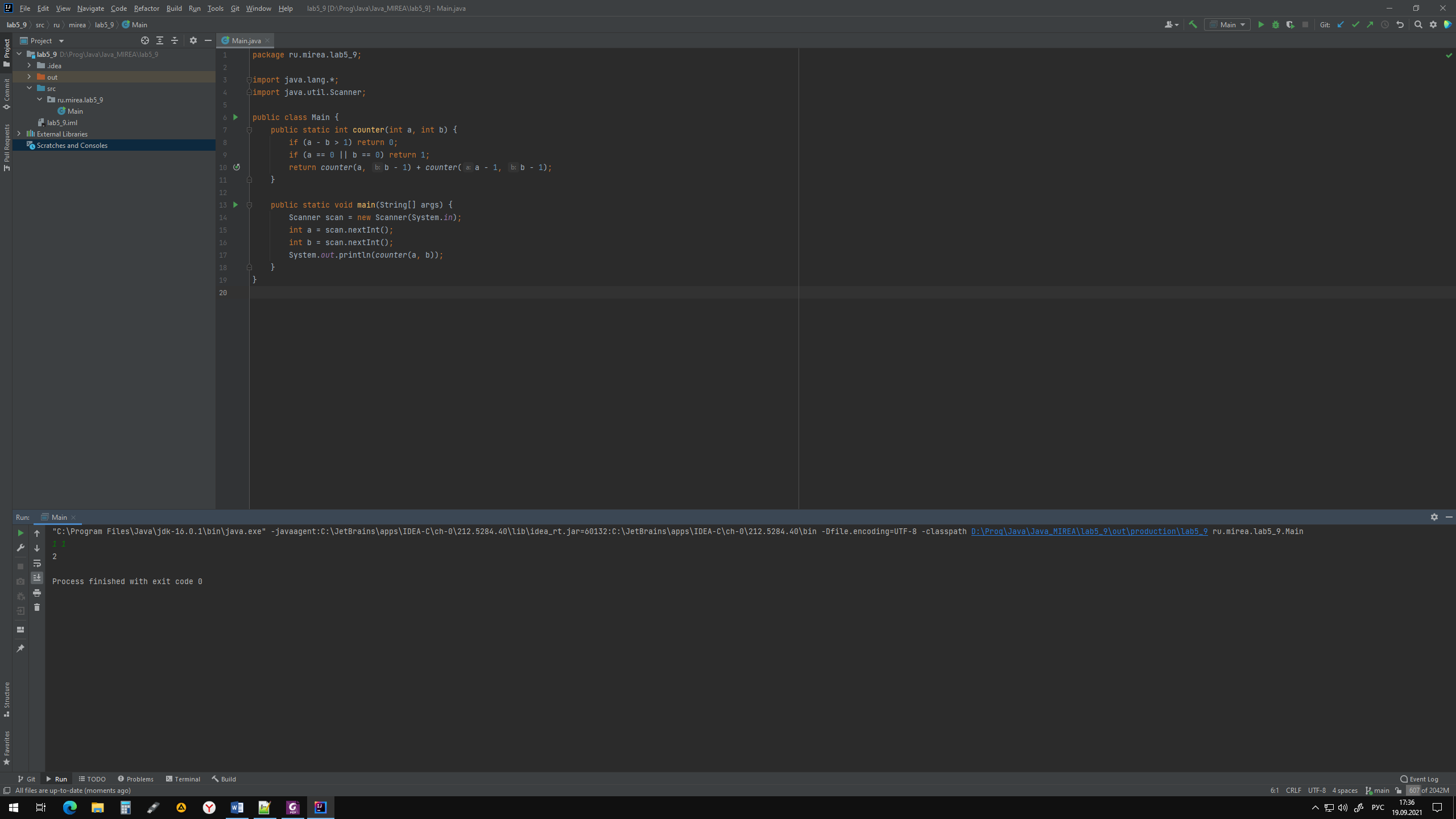


Рисунок 5.1 – Вывод программы.

Упражнение 2

Создаём класс Main и функцию main для запуска программы. В ней мы вызываем рекурсивную функцию. В ней мы считываем число от пользователя и проверяем его на чётность. Если оно нечётное, то сразу же выводим его пользователю, если же оно чётное, то просто пропускаем его.

package lab5\_12;

import java.lang.\*;

import java.util.Scanner;

public class Main {

private static Scanner scan = new Scanner(System.in);

public static void counter() {

int a = scan.nextInt();

if (a > 0) {

if (a % 2 == 1) {

System.out.println(a);

counter();

} else {

counter();

}

}

}

public static void main(String[] args) {

counter();

}

}

На рисунке 5.2 показан вывод программы, где зеленым выделены числа, вводимые пользователем, а белым – выводимые программой числа.

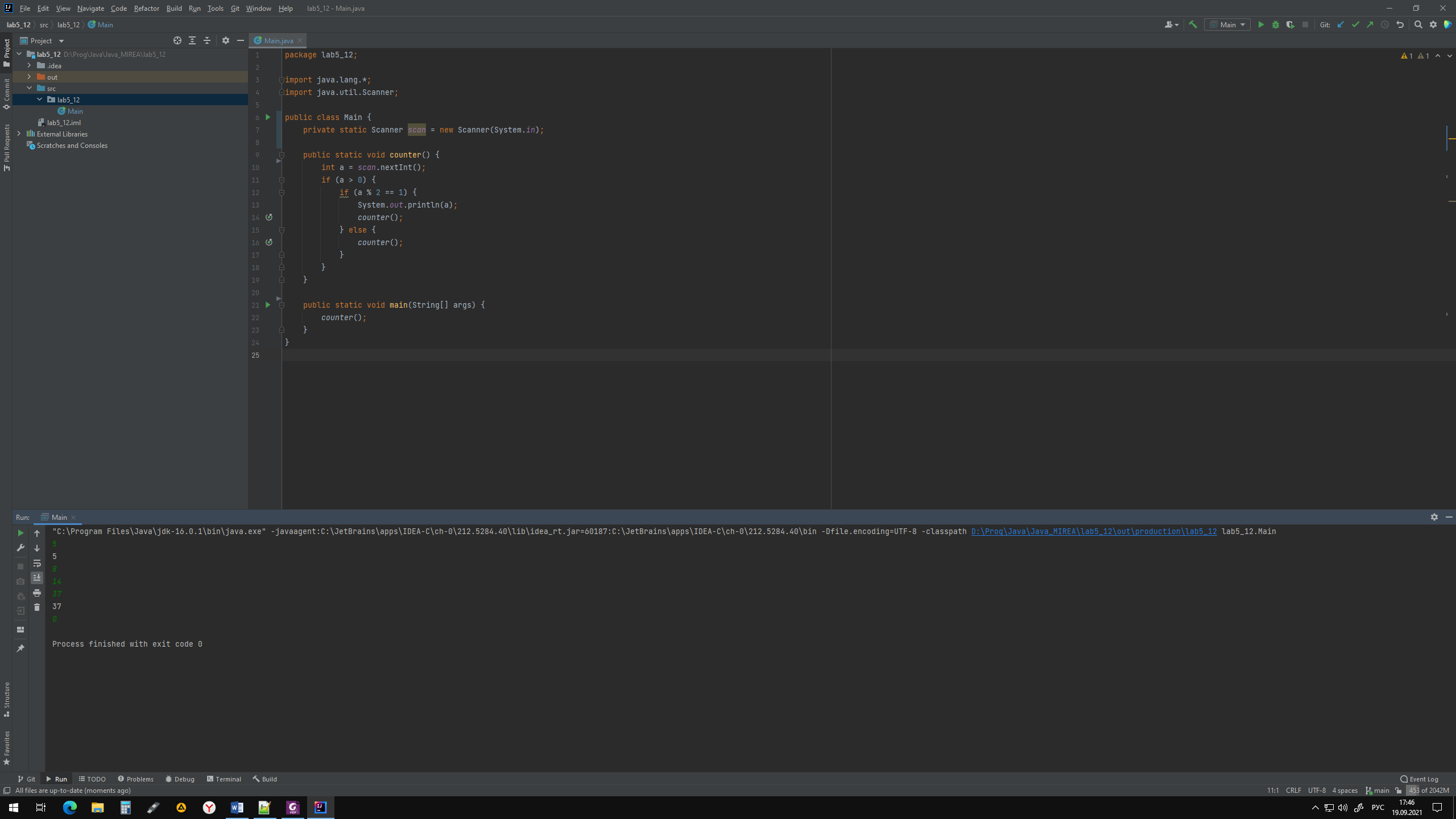


Рисунок 5.2 – Вывод программы.

Упражнение 3

Создаём класс Main и функцию main для запуска программы. В ней мы считываем число и передаем его в рекурсивную функцию. Если число получаемое функцией меньше десяти, то возвращаем его, если нет то печатаем цифру, и вызываем её еще раз и передаем число деленное на десять.

package ru.mirea.lab5\_15;

import java.util.Scanner;

public class Main {

public static int reverse(int n) {

if (n < 10) {

return n;

} else {

System.out.print(n % 10 + " ");

return reverse(n / 10);

}

}

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println(reverse(scan.nextInt()));

}

}

На рисунке 5.3 показан вывод программы. По нему можно понять, что программа работает правильно.

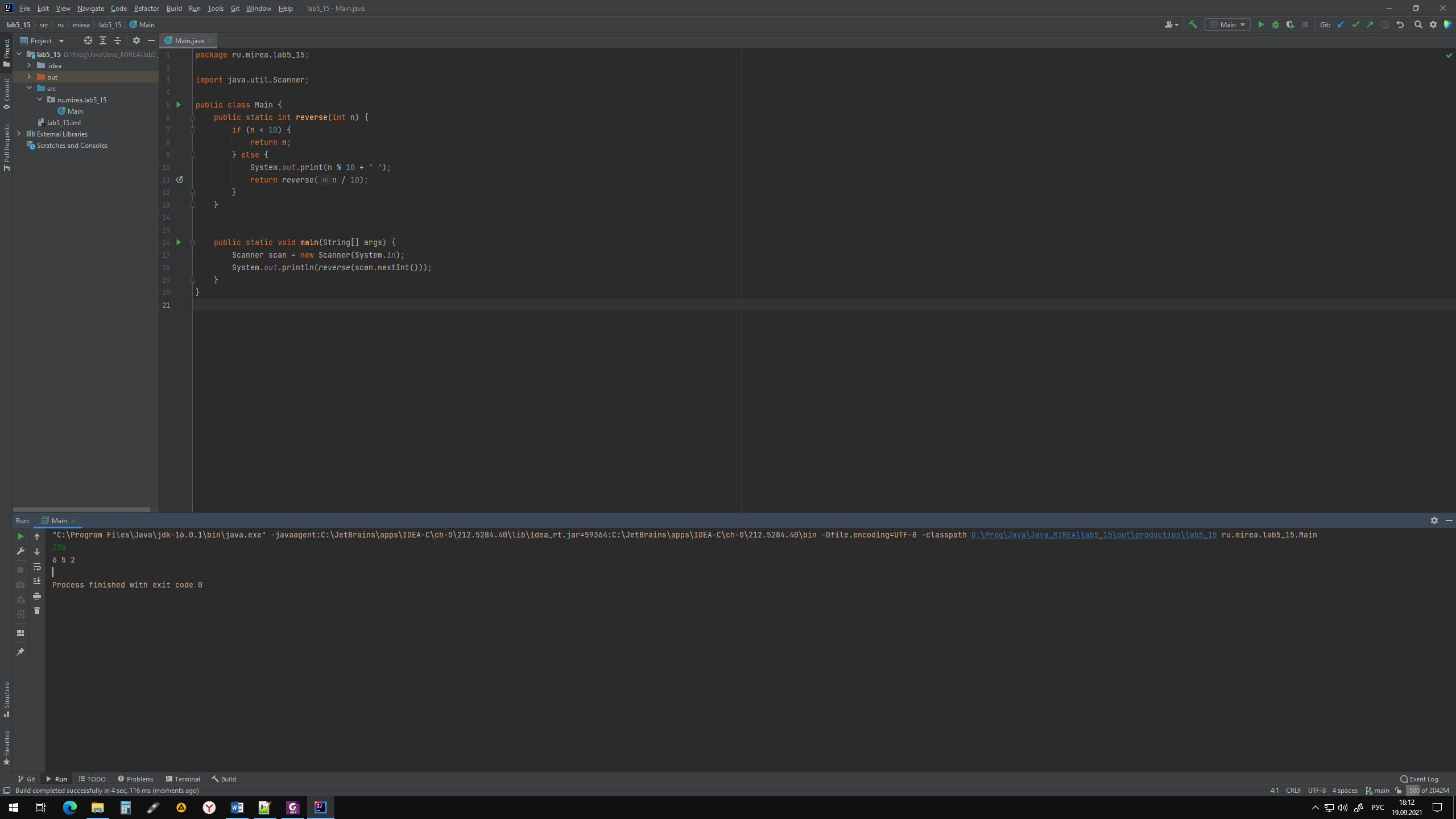


Рисунок 5.3 – Вывод программы.

**Выводы по работе:**

Выполняя данную практическую работу, я научился создавать рекурсивные функции и условия выхода из них.

Практическая работа № 6

Цель работы

Освоить на практике метод сортировки с использованием приемов программирования на объектно-ориентированном языке Java. Теоретическое введение

Сортировка — это процесс упорядочивания списка элементов (организация в определенном порядке) исходного списка элементов, который возможно организован в виде контейнера или храниться в виде коллекции.

Процесс сортировки основан на упорядочивании конкретных значений, например:

* сортировка списка результатов экзаменов баллов в порядке возрастания результата;
* сортировка списка людей в алфавитном порядке по фамилии.
* Есть много алгоритмов для сортировки списка элементов, которые различаются по эффективности.

*Алгоритм сортировки вставками****.***

Работа метода сортировки состоит из следующих шагов:

* рассматриваем первый элемент списка как отсортированный подсписок (то есть первый элемент списка);
* вставим второй элемент в отсортированный подсписок, сдвигая первый элемент по мере необходимости, чтобы освободить место для вставки нового элемента;
* вставим третий элемент в отсортированный подсписок (из двух элементов), сдвигая элементы по мере необходимости;
* повторяем до тех пор, пока все значения не будут вставлены на свои соответствующие позиции.

*Алгоритм быстрой сортировки (Quick Sort).*

Состоит из последовательного выполнения двух шагов:

* массив A[1..n] разбивается на два непустых подмассивов по отношению к "опорному элементу”;
* два подмассива сортируются рекурсивно посредством Quick Sort.

*Алгоритм сортировка слиянием (Merge Sort).*

Состоит из последовательного выполнения трех шагов:

* разделить массив A[1..n] на 2 равные части;
* провести сортировку слиянием двух подмассивов (рекурсивно);
* объединить (соединить) два отсортированных подмассива.

Выполнение лабораторной работы

Задание:

Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках с использованием алгоритма сортировки слиянием.

Создаём класс студента, в котором определяем поля для хранения средней оценки (вещественный тип), курса (целочисленный тип) и имя студента (строковый тип), а также конструктор, методы для получения всех поле и метод сравнения двух объектов данного класса.

package ru.mirea.lab6;

public class Student implements Comparable<Student> {

private double averageMark;

private int course;

private String name;

public Student(String name, double averageMark, int course) {

this.averageMark = averageMark;

this.course = course;

this.name = name;

}

public double getAverageMark() {

return averageMark;

}

public void setAverageMark(double averageMark) {

this.averageMark = averageMark;

}

public int getCourse() {

return course;

}

public void setCourse(int course) {

this.course = course;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public int compareTo(Student o) {

return averageMark > o.getAverageMark() ? -1 : 1;

}

}

Теперь создаём класс Sorting и определяем функции для сортировки, объединения массивов и главную функцию. В ней мы создаём два массива со студентами. После чего объединяем их и отправляем на сортировку.

package ru.mirea.lab6;

public class Sorting {

public static void mergeSort(Comparable<Student>[] array, int low, int high) {

if (high <= low) return;

int mid = (low + high) / 2;

mergeSort(array, low, mid);

mergeSort(array, mid + 1, high);

merge(array, low, mid, high);

}

public static void merge(Comparable<Student>[] array, int low, int mid, int high) {

Comparable<Student>[] leftArray = new Comparable[mid - low + 1];

Comparable<Student>[] rightArray = new Comparable[high - mid];

for (int i = 0; i < leftArray.length; i++)

leftArray[i] = array[low + i];

for (int i = 0; i < rightArray.length; i++)

rightArray[i] = array[mid + i + 1];

int leftIndex = 0;

int rightIndex = 0;

for (int i = low; i < high + 1; i++) {

if (leftIndex < leftArray.length && rightIndex < rightArray.length) {

if (leftArray[leftIndex].compareTo((Student) rightArray[rightIndex]) < 0) {

array[i] = leftArray[leftIndex];

leftIndex++;

} else {

array[i] = rightArray[rightIndex];

rightIndex++;

}

} else if (leftIndex < leftArray.length) {

array[i] = leftArray[leftIndex];

leftIndex++;

} else if (rightIndex < rightArray.length) {

array[i] = rightArray[rightIndex];

rightIndex++;

}

}

}

public static void main(String[] args) {

Comparable<Student>[] studentsM = new Comparable[8];

studentsM[0] = new Student("Alexey", 4.78, 2);

studentsM[1] = new Student("Alexander", 4.06, 1);

studentsM[2] = new Student("Nikolay", 4.26, 5);

studentsM[3] = new Student("Igor", 3.87, 3);

studentsM[4] = new Student("Danila", 3.93, 2);

studentsM[5] = new Student("Egor", 3.74, 3);

studentsM[6] = new Student("Pavel", 4.58, 1);

studentsM[7] = new Student("Artem", 4.83, 4);

Comparable<Student>[] studentsF = new Comparable[8];

studentsF[0] = new Student("Anastasia", 4.95, 2);

studentsF[1] = new Student("Alexandra", 4.02, 1);

studentsF[2] = new Student("Anna", 4.23, 5);

studentsF[3] = new Student("Tatiana", 3.86, 3);

studentsF[4] = new Student("Masha", 3.96, 2);

studentsF[5] = new Student("Polina", 3.78, 3);

studentsF[6] = new Student("Victoria", 4.58, 1);

studentsF[7] = new Student("Ekaterina", 4.88, 4);

Comparable<Student>[] students = new Comparable[16];

for (int i = 0; i < studentsM.length; i++) students[i] = studentsM[i];

for (int i = 0; i < studentsM.length; i++) students[i + studentsM.length] = studentsF[i];

for (Comparable<Student> elem : students) {

Student student = (Student)elem;

System.out.printf("%-10s %-1f %-1d",student.getName(),student.getAverageMark(),student.getCourse());

System.out.println();

}

System.out.println("\n");

mergeSort(students, 0, students.length - 1);

for (Comparable<Student> elem : students) {

Student student = (Student)elem;

System.out.printf("%-10s %-1f %-1d",student.getName(),student.getAverageMark(),student.getCourse());

System.out.println();

}

}

}

При запуске программы она сначала выводит информацию о студентах в массиве до сортировки, а затем выводит после сортировки. На рисунке 6.1 показан вывод программы. По нему видно, что программа работает правильно.

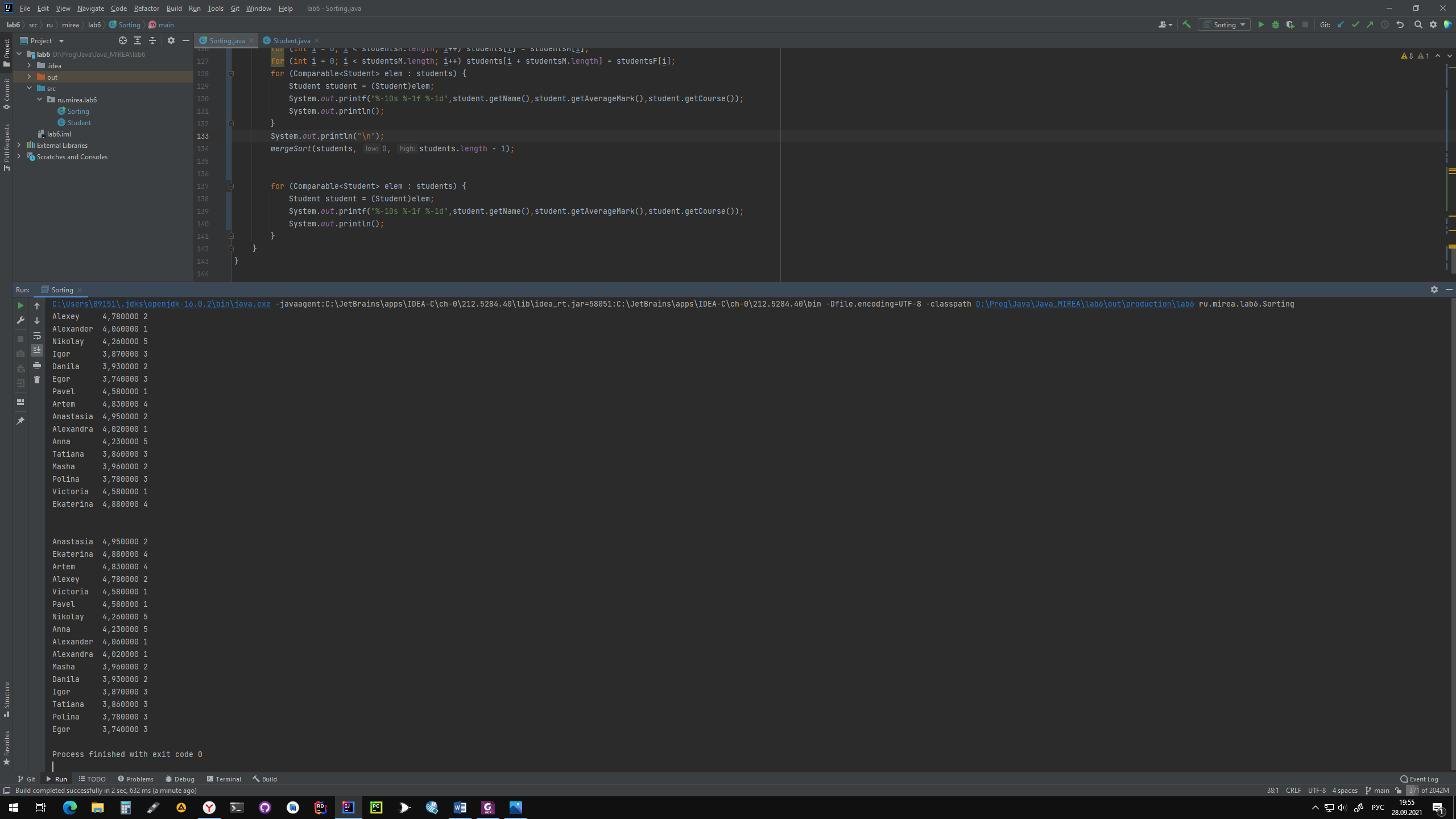


Рисунок 6.2 – Вывод программы.

**Выводы по работе:**

Выполняя данную практическую работу, я научился создавать функции сортировки.

Практическая работа № 7

Цель работы

Изучение на практике приемов работы со стандартными

контейнерными классами Java Collection Framework.

Теоретическое введение

Java Collections Framework — это набор связанных классов и интерфейсов,реализующих широко используемые структуры данных — коллекции. На вершине иерархии в Java Collection Framework располагаются 2 интерфейса: Collection и Map. Эти интерфейсы разделяют все коллекции, входящие в фреймворк на две части по типу хранения данных: простые последовательные наборы элементов и наборы пар «ключ — значение» (словари).

Vector — реализация динамического массива объектов. Позволяет хранить любые данные, включая в качестве элемента. Stack — данная коллекция является расширением коллекции Vector. Была добавлена как реализация стека LIFO (last-in-first-out).

LinkedList — вид реализации List. Позволяет хранить любые данные, включая null. Данная коллекция реализована на основе двунаправленного связного списка (каждый элемент списка имеет ссылки на предыдущий и следующий). Добавление и удаление элемента из середины, доступ по индексу, значению происходит за линейное время O(n), а из начала и конца за константное время O(1). Ввиду реализации, данную коллекцию можно использовать как абстрактный тип данных — стек или очередь. Для этого в ней реализованы соответствующие методы.

Интерфейс Set.

Представляет собой неупорядоченную коллекцию, которая не может содержать одинаковые элементы и является программной моделью математического понятия «множество».

Интерфейс Queue.

Этот интерфейс описывает коллекции с предопределённым способом вставки и извлечения элементов, а именно — очереди FIFO (first-in-first-out). Помимо методов, определённых в интерфейсе Collection, определяет дополнительные методы для извлечения и добавления элементов в очередь.

PriorityQueue — является единственной прямой реализацией интерфейса Queue (была добавлена, как и интерфейс Queue, в Java 1.5), не считая класса LinkedList, который так же реализует этот интерфейс, но был реализован намного раньше. Особенностью данной очереди является возможность управления порядком элементов. По умолчанию, элементы сортируются с использованием «natural ordering», но это поведение может быть переопределено при помощи объекта Comparator, который задаётся при создании очереди. Данная коллекция не поддерживает в качестве элементов.

ArrayDeque — реализация интерфейса Deque, который расширяет интерфейс Queue методами, позволяющими реализовать конструкцию вида LIFO (last-in-first-out). Интерфейс Deque и реализация ArrayDeque были добавлены в Java 1.6. Эта коллекция представляет собой реализацию с использованием массивов, подобно ArrayList, но не позволяет обращаться к элементам по индексу и хранение null. Как заявлено в документации, эта коллекция работает быстрее чем Stack, если используется как LIFO коллекция, а также быстрее чем LinkedList, если используется как FIFO коллекция.

Выполнение лабораторной работы

Задание:

Напишите программу в виде консольного приложения, которая моделирует карточную игру «пьяница» и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую; карта «0» побеждает карту «9».

Карточная игра “ В пьяницу”. В этой игре карточная колода раздается поровну двум игрокам. Далее они открывают по одной верхней карте, и тот, чья карта старше, забирает себе обе открытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт, - проигрывает.

Для простоты будем считать, что все карты различны по номиналу и что самая младшая карта побеждает самую старшую карту (“шестерка берет туз”).

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

Входные данные.

Программа получает на вход две строки: первая строка содержит 5 карт первого игрока, вторая - 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

Выходные данные.

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 106 ходов игра не заканчивается, программа должна вывести слово botva.

Используйте для организации хранения структуру данных Stack.

Решение:

Сначала создаём класс игральной карты. В нём определим поле значения карты, конструктор, поля для получения значения.

package ru.mirea.lab7;

public class Card {

private int value;

public Card(int value) {

this.value = value;

}

public int getValue() {

return value;

}

}

Далее создаём класс для запуска программы. В нём определяем класс main. В нём создаём два массива с картами игроков.

package ru.mirea.lab7;

import java.util.ArrayDeque;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

ArrayDeque<Card> first = new ArrayDeque<Card>(5);

ArrayDeque<Card> second = new ArrayDeque<Card>(5);

Scanner scan = new Scanner(System.in);

for (int i = 0; i < 5; i++) first.add(new Card(scan.nextInt()));

for (int i = 0; i < 5; i++) second.add(new Card(scan.nextInt()));

int actions = 0;

while (first.size() > 0 && second.size() > 0) {

if (actions >= 106) {

System.out.println("botva");

break;

} else if (first.getFirst().getValue() > second.getFirst().getValue() || ((first.getFirst().getValue() == 0) && (second.getFirst().getValue() == 9))) {

first.add(first.getFirst());

first.add(second.getFirst());

first.removeFirst();

second.removeFirst();

actions += 1;

} else if (first.getFirst().getValue() < second.getFirst().getValue() || ((first.getFirst().getValue() == 9) && (second.getFirst().getValue() == 0))) {

second.add(second.getFirst());

second.add(first.getFirst());

first.removeFirst();

second.removeFirst();

actions += 1;

}

if (first.size() == 0) System.out.println("first " + actions);

else if (second.size() == 0) System.out.println("second " + actions);

}

}

}

На рисунке 7.1 показан вывод программы. По нему можно понять, что программа работает правильно.

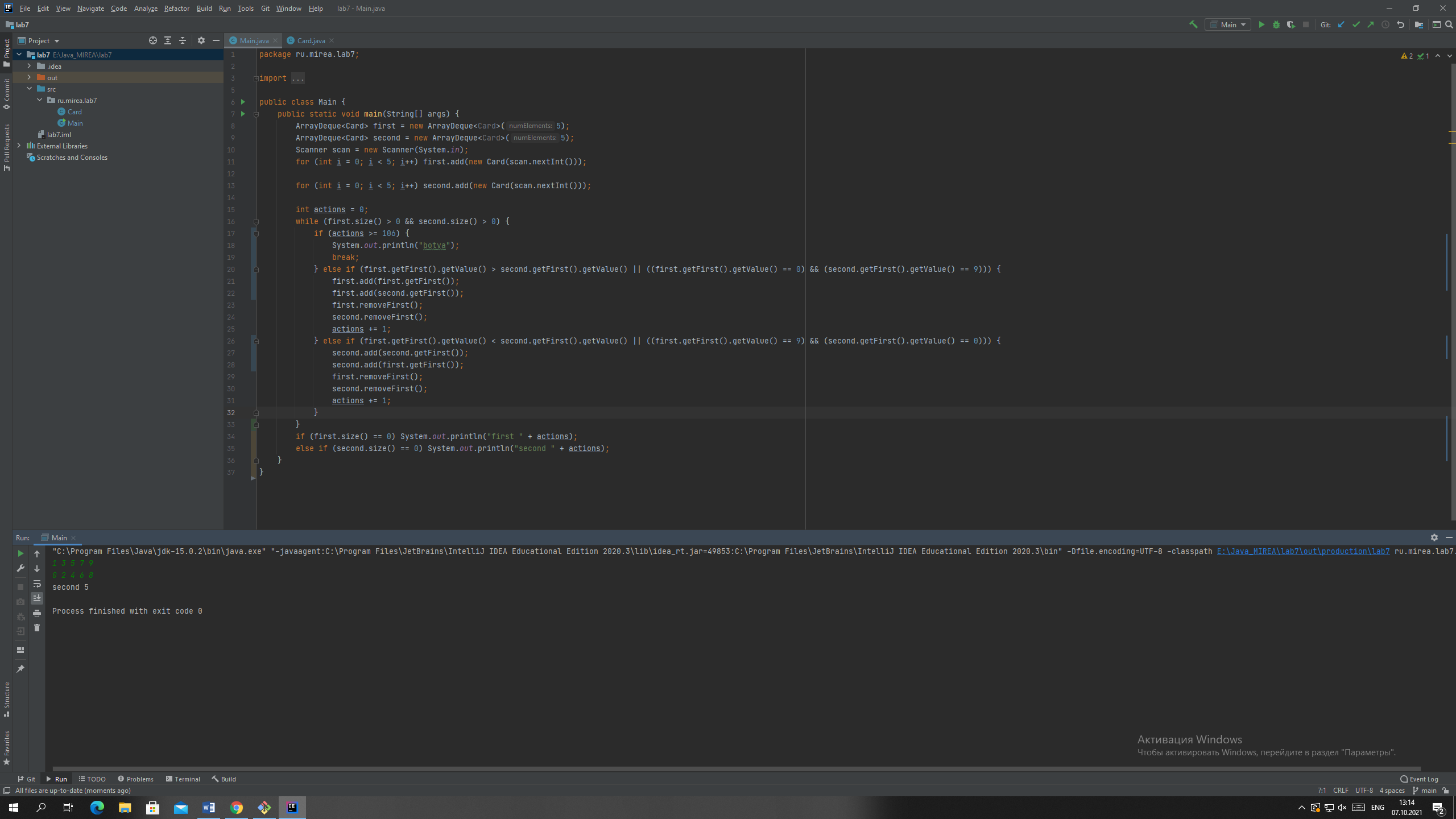


Рисунок 7.1 – Вывод программы

**Выводы по работе:**

Выполняя данную практическую работу, я научился создавать и использовать различные виды массивов.

Практическая работа № 10

Цель работы

Цель данной лабораторной работы – закрепить знания в области обработки строк, научиться применять методы класса String и других классов для обработки строк.

Теоретическое введение

Для работы с текстовыми данными в Java есть три класса: String, StringBuffer и StringBuilder.

Особенности использования строк в Java:

В Java строки представляют собой неизменяемую последовательность символов Unicode. В отличие от представления в C / C ++, где строка является просто массивом типа char, любая Java, строка является объектом класса java.lang.

Однако Java строка, представляет собой в отличие от других используемых классов особый класс, который обладает довольно специфичными характеристиками. Отличия класса строк от обычных классов:

* Cтрока в Java представляет из себя строку литералов (текст), помещенных в двойные кавычки, например:
* "Hello , World! ". Вы можете присвоить последовательность строковых литералов непосредственно переменной типа String, вместо того чтобы вызывать конструктор для создания экземпляра класса String.
* Оператор '+' является перегруженным, для объектов типа String, и всегда используется, чтобы объединить две строки операндов. В данном контексте мы говорим об операции конкатенации или сложения строк. Хотя '+' не работает как оператор сложения для любых других объектов, кроме строк, например, таких как Point и Circle.
* Строка является неизменяемой, то есть, символьной константой. Это значит, что ее содержание не может быть изменено после ее (строки как объекта) создания. Например, метод toUpperCase () – преобразования к верхнему регистру создает и возвращает новую строку вместо изменения содержания существующей строки.

Статический метод String.format() может быть использован для получения форматированного вывода, таким же образом как это делается в языке Си с использованием функции printf() и спецификаторов вывода для различных типов данных. Метод format() делает то же самое, что функция printf().

Разработчики Java все-таки решили сохранить примитивные типы в объектно-ориентированном языке вместо того, чтобы сделать вообще все в виде объектов. Нужно сказать, что сделано это в первую очередь, для того чтобы повысить производительность языка. Ведь примитивы хранятся в стеке вызовов, и, следовательно, требуют меньше пространства для хранения, и ими легче управлять. С другой стороны, объекты хранятся в области памяти, которую используют программы, и которая называется “куча” (heap), а этот механизм требуют сложного управления памятью и потребляет гораздо больше места для хранения.

По соображениям производительности, класс String в Java разработан, так, чтобы быть чем-то промежуточным между примитивными типами данных и типами данных типа класс. Как уже было отмечено выше специальные характеристики типа String включают в себя:

• '+' оператор, который выполняет сложение примитивных типов данных (таких, как int и double), и перегружен, чтобы работать на объектах String. Операция '+’ выполняет конкатенацию двух операндов типа String.

• Java не поддерживает механизма перегрузки операций по разработке программного обеспечения. В языке, который поддерживает перегрузку операций, например C ++, вы можете превратить оператор '+' (с помощью перегрузки) в оператор для выполнения сложения или вообще вычитания, например двух матриц, кстати это будет примером плохого кода. В Java оператор '+' является единственным оператором, который внутренне перегружен, чтобы поддержать конкатенацию (сложение) строк в Java. Нужно принять к сведению, что '+' не работает на любых других произвольных объектах, помимо строк, например, таких как рассмотренные нами ранее классы Point или Circle.

Выполнение лабораторной работы

Задание:

Реализуйте класс Shirt: Метод toString() выводит объяснение и значение полей построчно.

Дан также строковый массив: shirts[0] = "S001,Black Polo Shirt,Black,XL"; shirts[1] = "S002,Black Polo Shirt,Black,L"; shirts[2] = "S003,Blue Polo Shirt,Blue,XL"; shirts[3] = "S004,Blue Polo Shirt,Blue,M"; shirts[4] = "S005,Tan Polo Shirt,Tan,XL"; shirts[5] = "S006,Black T-Shirt,Black,XL"; shirts[6] = "S007,White T-Shirt,White,XL"; shirts[7] = "S008,White T-Shirt,White,L"; shirts[8] = "S009,Green T-Shirt,Green,S"; shirts[9] = "S010,Orange T-Shirt,Orange,S"; shirts[10] = "S011,Maroon Polo Shirt,Maroon,S";

Преобразуйте строковый массив в массив класса Shirt и выведите его на консоль.

Решение:

Создаём класс Shirt. В нём определяем поля для хранения номера, названия, цвета и размера товара.

package ru.mirea.lab10;

public class Shirt {

private String number;

private String name;

private String color;

private String size;

public Shirt(String number, String name, String color, String size) {

this.number = number;

this.name = name;

this.color = color;

this.size = size;

}

public String getNumber() {

return number;

}

public String getName() {

return name;

}

public String getColor() {

return color;

}

public String getSize() {

return size;

}

@Override

public String toString() {

return number + "," + name + "," + color + "," + size;

}

}

Далее создаём класс для запуска программы. В нём определяем класс main. В нём создаём массив с ассортиментом магазина. Затем создаём лист. В него для каждого товара создаём объект класса Shirt и вытаскиваем значения номера, названия, цвета и размера.

package ru.mirea.lab10;

import java.util.ArrayList;

public class Main {

public static void main(String[] args) {

String[] shirts = new String[11];

shirts[0] = "S001,Black Polo Shirt,Black,XL";

shirts[1] = "S002,Black Polo Shirt,Black,L";

shirts[2] = "S003,Blue Polo Shirt,Blue,XL";

shirts[3] = "S004,Blue Polo Shirt,Blue,M";

shirts[4] = "S005,Tan Polo Shirt,Tan,XL";

shirts[5] = "S006,Black T-Shirt,Black,XL";

shirts[6] = "S007,White T-Shirt,White,XL";

shirts[7] = "S008,White T-Shirt,White,L";

shirts[8] = "S009,Green T-Shirt,Green,S";

shirts[9] = "S010,Orange T-Shirt,Orange,S";

shirts[10] = "S011,Maroon Polo Shirt,Maroon,S";

ArrayList<Shirt> shirtArrayList = new ArrayList<>();

for (int i = 0; i < shirts.length; i++) {

String number, name, color, size;

number = shirts[i].substring(0, shirts[i].indexOf(','));

name = shirts[i].substring(5, shirts[i].indexOf(',', 6));

color = shirts[i].substring(shirts[i].indexOf(',', 10) + 1, shirts[i].lastIndexOf(','));

size = shirts[i].substring(shirts[i].lastIndexOf(',') + 1);

shirtArrayList.add(new Shirt(number, name, color, size));

}

for (Shirt shirt : shirtArrayList) {

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

}

}

}

На рисунке 10.1 показан вывод программы. По нему можно понять, что программа работает правильно.

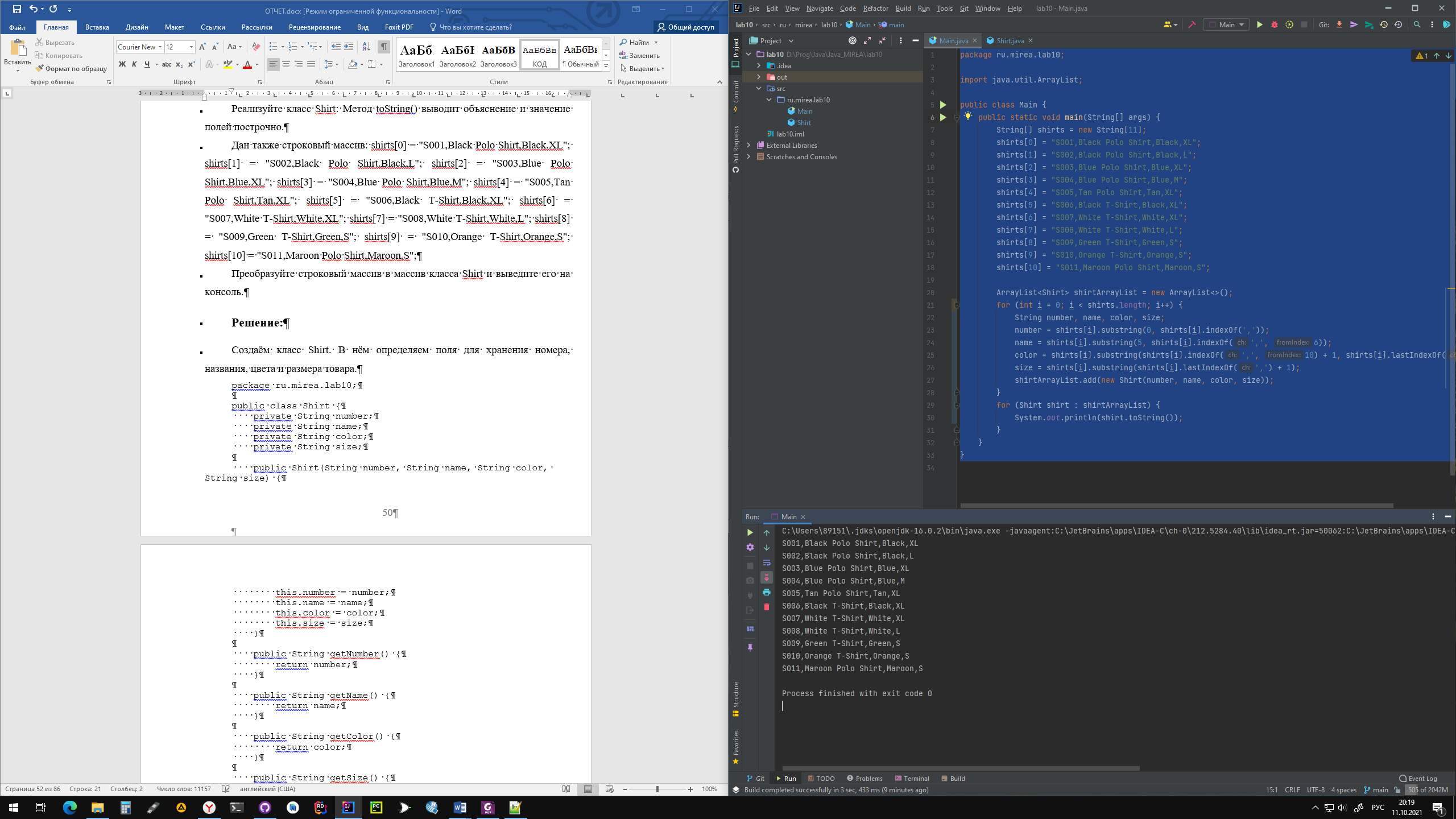


Рисунок 10.1 – Вывод программы.

**Выводы по работе:**

Выполняя данную практическую работу, я научился работать со строками.

Практическая работа № 18

Цель работы

Целями данной работы являются получение практических навыков разработки программ, изучение синтаксисаязыка Java, освоение основных конструкций языка Java (циклы, условия,создание переменных и массивов, создание методов, вызов методов), атакже научиться осуществлять стандартный ввод/вывод данных.

Теоретическое введение

Чтобы объявить переменную, необходимо указать тип переменной и ее имя. Типы переменной могут быть разные: целочисленный (long, int, short, byte), число с плавающей запятой (double, float), логический(boolean), перечисление, объектный (Object).

Целочисленным переменным можно присваивать только целые числа, а числам с плавающей запятой - дробные. Целые числа обозначаются цифрами от 0 до 9, а дробные можно записывать, отделяя целую часть от дробной с помощью точки. Переменным типа float необходимо приписывать справа букву "f", обозначающую, что данное число типа float. Без этой буквы число будет иметь тип double.

Класс String - особый класс в Java, так как ему можно присваивать значение, не создавая экземпляра класса (Java это сделает автоматически). Этот класс предназначен для представления строк. Строковое значение записывается буквами внутри двойных кавычек

Массив — это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу. Для того чтобы создать массив переменных, необходимо указать квадратные скобки при объявлении переменной массива. После чего необходимо создать массив с помощью оператора new. Необходимо указать в квадратных скобках справа размер массива.

Условие - это конструкция, позволяющая выполнять то или другое действие, в зависимости от логического значения, указанного в условии. Если логическое условие, указанное в скобках после ключевого слова if, истинно, то будет выполняться блок кода, следующий за if, иначе будет выполняться код, следующий за ключевым словом else.

Цикл - это конструкция, позволяющая выполнять определенную часть кода несколько раз. В Java есть три типа циклов for, while, do while.

Цикл for - это цикл со счетчиком, обычно используется, когда известно, сколько раз должна выполниться определенная часть кода.

Цикл while - это такой цикл, который будет выполняться, пока логическое выражение, указанное в скобках истинно.

Цикл do while - это такой цикл, тело которого выполнится хотя бы один раз. Тело выполнится более одного раза, если условие, указанное в скобках истинно.

Для ввода данных используется класс Scanner из библиотеки пакетов. Этот класс надо импортировать в той программе, где он будет использоваться. Это делается до начала открытого класса в коде программы. Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Java представлен объектом — System.in. А стандартный поток вывода (дисплей) — уже знакомым вам объектом System.out. Есть ещё стандартный поток для вывода ошибок — System.err.

Выполнение лабораторной работы

Задание:

Сгенерировать массив целых чисел случайным образом, вывести его на экран, отсортировать его, и снова вывести на экран.

Решение:

Создаём класс Main и в нём функцию main для запуска программы. Далее задаём длину массива, и заполняем его случайными числами, параллельно выводя их на экран через пробел. Создаём функцию для пузырьковой сортировки и передаём в неё массив. Затем второй строкой выводим пользователю отсортированный массив.

package ru.mirea.lab18;

public class Main {

public static void bubbleSort(int[] array) {

boolean sorted = false;

int temp;

while (!sorted) {

sorted = true;

for (int i = 0; i < array.length - 1; i++) {

if (array[i] > array[i + 1]) {

temp = array[i];

array[i] = array[i + 1];

array[i + 1] = temp;

sorted = false;

}

}

}

}

public static void main(String[] args) {

int N = 10;

int[] arr = new int[N];

for (int i = 0; i < N; i++) {

arr[i] = (int) (Math.random() \* 100);

System.out.print(arr[i] + " ");

}

System.out.println();

bubbleSort(arr);

for (int i = 0; i < N; i++) {

System.out.print(arr[i] + " ");

}

}

}

На рисунке 18.1 показан вывод программы. На первой строке выведен неотсортированный массив случайных чисел, а на второй после пузырьковой сортировки. По выводу можно понять, что программа работает правильно.

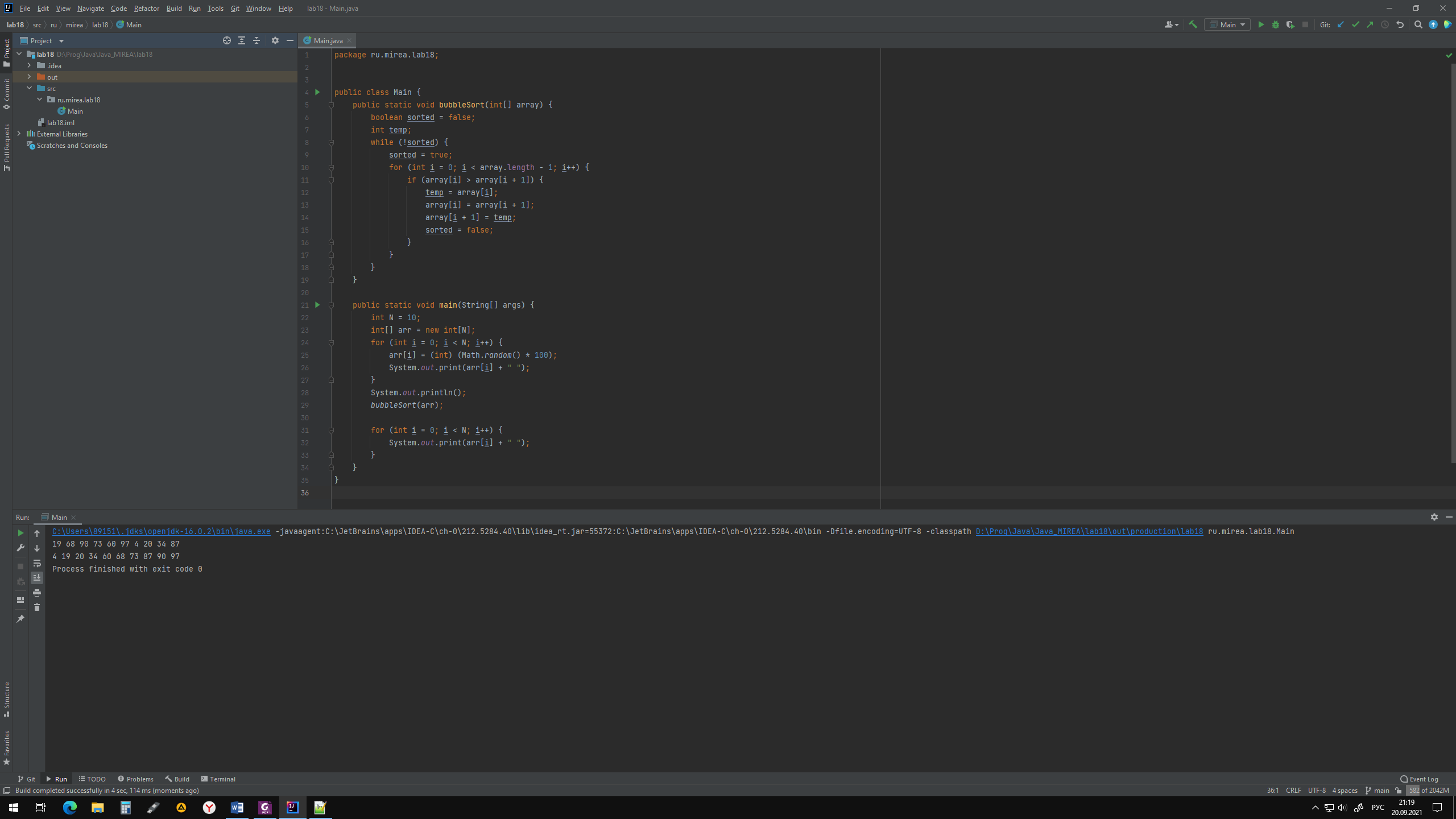


Рисунок 18.1 – Вывод программы

**Выводы по работе:**

Выполняя данную практическую работу, я изучил способы сортировки, научился создавать массивы целых чисел и передавать их в сортирующую функцию.

Практическая работа № 19

Цель работы

Цель данной работы - изучить основные концепции объектно-ориентированного программирования, изучить понятие класса и научиться создавать классы.

Теоретическое введение

Язык Java — объектно-ориентированный язык программирования. В центре ООП находится понятие объекта. Объект — это сущность, которой можно посылать сообщения, и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Класс в ООП — это в чистом виде абстрактный тип данных, создаваемый программистом. С этой точки зрения объекты являются значениями данного абстрактного типа, а определение класса задаёт внутреннюю структуру значений и набор операций, которые над этими значениями могут быть выполнены. Желательность иерархии классов (а значит, наследования) вытекает из требований к повторному использованию кода — если несколько классов имеют сходное поведение, нет смысла дублировать их описание, лучше выделить общую часть в общий родительский класс, а в описании самих этих классов оставить только различающиеся элементы.

Для того чтобы создать класс в языке Java необходимо создать файл с расширением .java. Имя файла должно быть таким же, как и имя создаваемого класса. В созданном файле должен описываться класс.

В качестве модификатора доступа можно указать ключевое слово public или private. Если указано слово public, то класс будет доступен из других пакетов. Если указано слово private, то класс будет доступен только внутри того пакета, в котором он находится. В теле класса можно описать методы, переменные, константы, конструкторы класса.

Конструктор - это специальный метод, который вызывается при создании нового объекта.

Для того чтобы создать экземпляр класса необходимо объявить переменную, тип которой соответствует имени класса или суперкласса. После чего нужно присвоить этой переменной значение, вызвав конструктор создаваемого класса с помощью оператора new.

Выполнение лабораторной работы

Задание:

Создать класс, описывающий книгу (Book). В классе должны быть описаны нужные свойства книги(автор, название, год написания и т. д.)и методы для получения, изменения этих свойств. Протестировать работу класса в классе BookTest, содержащим метод статический main(String[] args).

Решение:

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

package ru.mirea.lab19;

import java.lang.\*;

public class Book {

private String bookName;

private String authorName;

private int writingYear;

private int pages;

public Book(String bookName, String authorName, int writingYear, int pages) {

this.bookName = bookName;

this.authorName = authorName;

this.pages = pages;

this.writingYear = writingYear;

}

public Book(String bookName) {

this.bookName = bookName;

}

public String getBookName() {

return bookName;

}

public void setBookName(String bookName) {

this.bookName = bookName;

}

public String getAuthorName() {

return authorName;

}

public void setAuthorName(String authorName) {

this.authorName = authorName;

}

public int getWritingYear() {

return writingYear;

}

public void setWritingYear(int writingYear) {

this.writingYear = writingYear;

}

public int getPages() {

return pages;

}

public void setPages(int pages) {

this.pages = pages;

}

public String toString() {

return "Книга: " + bookName + "; Автор: " + authorName + "; Год написания: " + writingYear + "; Страниц: " + pages;

}

}

Создаём новый класс BookTest. 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.lab19;

public class BookTest {

public static void main(String[] args) {

Book book1 = new Book("Мертвые души", "Николай Гоголь", 1841, 352);

Book book2 = new Book("Война и мир");

book2.setAuthorName("Лев Толстой");

book2.setWritingYear(1869);

book2.setPages(1827);

System.out.println(book1.toString());

System.out.println(book2.toString());

}

}

На рисунке 19.1 показан вывод программы. По нему можно убедится, что программа работает правильно.

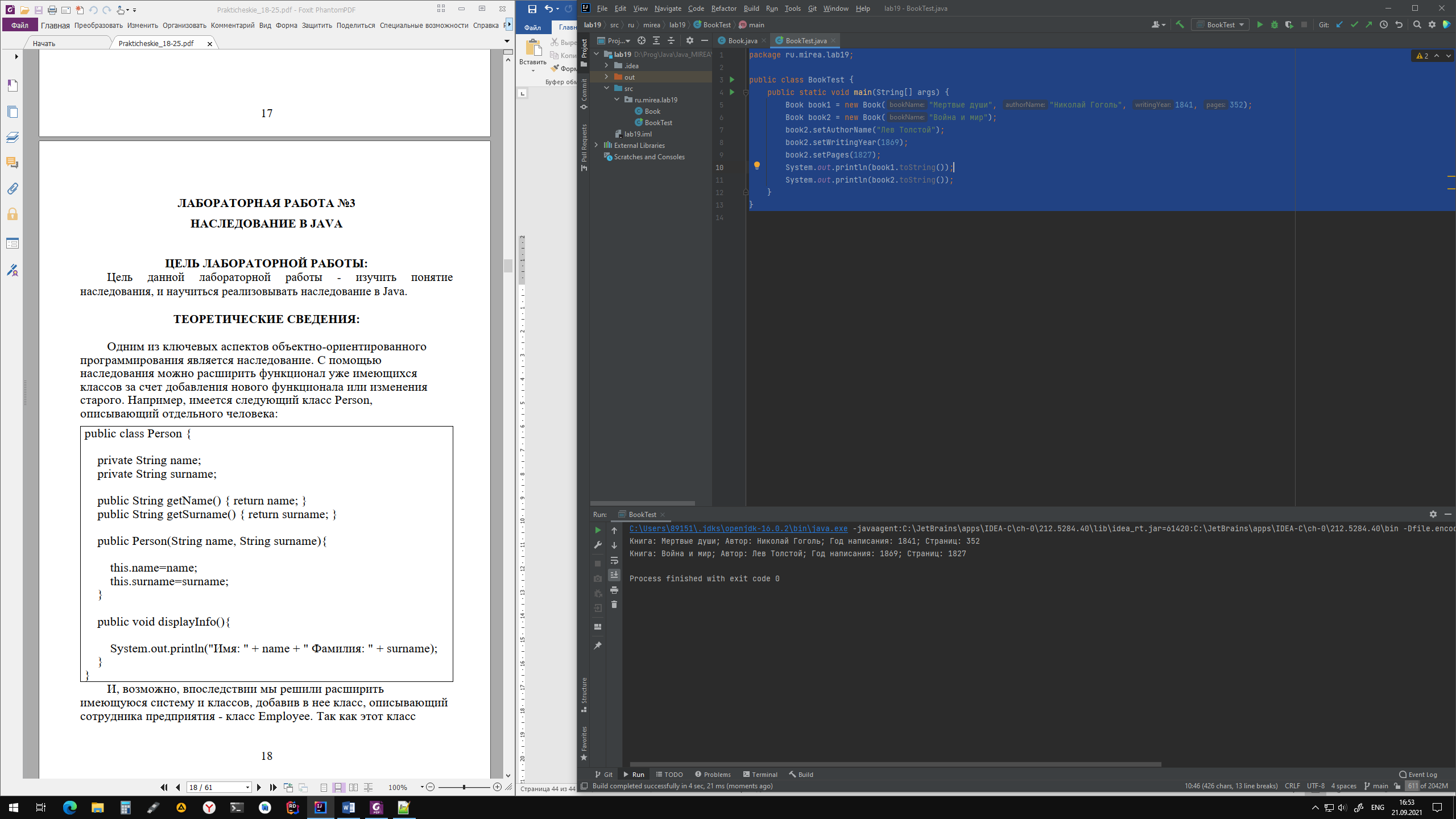


Рисунок 19.1 - Вывод программы.

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 № 20

Цель работы:

Цель данной работы — изучить понятие наследования, и научиться реализовывать наследование в Java.

Теоретическое введение

Одним из ключевых аспектов объектно-ориентированного программирования является наследование. С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого.

И, возможно, впоследствии мы решили расширить имеющуюся систему и классов, добавив в нее класс, описывающий сотрудника предприятия - класс Employee. Так как этот класс реализует тот же функционал, что и класс Person, так как сотрудник — это также и человек, то было бы рационально сделать класс Employee производным (или наследником) от класса Person, который, в свою очередь, называется базовым классом или родителем. Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово extends, после которого идет имя базового класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же поля и методы, которые есть в классе Person.

Кроме обычных классов в Java есть абстрактные классы. Абстрактный класс похож на обычный класс. В абстрактном классе также можно определить поля и методы, в то же время нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов-наследников. А производные классы уже реализуют этот функционал.

Выполнение лабораторной работы

Задание:

Создать абстрактный класс, описывающий мебель. С помощьюнаследования реализовать различные виды мебели. Также создатькласс FurnitureShop, моделирующий магазин мебели. Протестироватьработу классов.

Решение:

Создаём абстрактный класс Furniture и в нём описываем три поля для хранения материала (строковый тип), цены (целочисленный тип) и цвета (строковый тип), а также методы для получения и установки значений этих полей, и метод для получения полной информации об объекте.

package ru.mirea.lab20;

public abstract class Furniture {

private String material;

private int price;

private String color;

public Furniture(String material, int price, String color) {

this.material = material;

this.price = price;

this.color = color;

}

public String getMaterial() {

return material;

}

public int getPrice() {

return price;

}

public void setPrice(int price) {

this.price = price;

}

public String getColor() {

return color;

}

@Override

public String toString() {

return "{" +

"Материал: " + material +

", Цена: " + price +

", Цвет:" + color +

"}";

}

}

Наследуя данный абстрактный класс, создаём класс Chair. В нём определяем конструктор и метод получения информации об объекте.

package ru.mirea.lab20;

public class Chair extends Furniture {

public Chair(String material, int price, String color) {

super(material, price, color);

}

@Override

public String toString() {

return "Стул " + super.toString();

}

}

Наследуя данный абстрактный класс, создаём класс Sofa. В нём определяем конструктор и метод получения информации об объекте.

package ru.mirea.lab20;

public class Sofa extends Furniture {

public Sofa(String material, int price, String color) {

super(material, price, color);

}

@Override

public String toString() {

return "Диван " + super.toString();

}

}

Наследуя данный абстрактный класс, создаём класс Table. В нём определяем конструктор и метод получения информации об объекте.

package ru.mirea.lab20;

public class Table extends Furniture {

public Table(String material, int price, String color) {

super(material, price, color);

}

@Override

public String toString() {

return "Стол " + super.toString();

}

}

Наследуя данный абстрактный класс, создаём класс Wardrobe. В нём определяем конструктор и метод получения информации об объекте.

package ru.mirea.lab20;

public class Wardrobe extends Furniture {

public Wardrobe(String material, int price, String color) {

super(material, price, color);

}

@Override

public String toString() {

return "Шкаф " + super.toString();

}

}

Также создаём класс FurnitureShop. В нём создаем метод main для запуска программы и создаём объекты каждого класса.

package ru.mirea.lab20;

import java.util.ArrayList;

import java.util.List;

public class FurnitureShop {

private static ArrayList<Furniture> furnitures = new ArrayList<Furniture>();

public static void main(String[] args) {

furnitures.add(new Chair("Дерево", 500, "Чёрный"));

furnitures.add(new Wardrobe("Дерево", 2500, "Бежевый"));

furnitures.add(new Table("Стекло", 4500, "Прозрачный"));

furnitures.add(new Sofa("Искусственная кожа", 5000, "Янтарный"));

printAssortment();

}

private static void printAssortment() {

for (int i = 0; i < furnitures.toArray().length; i++) {

System.out.println(furnitures.toArray()[i].toString());

}

}

}

На рисунке 20.1 показан вывод программы. По нему можно убедится, что программа работает правильно.

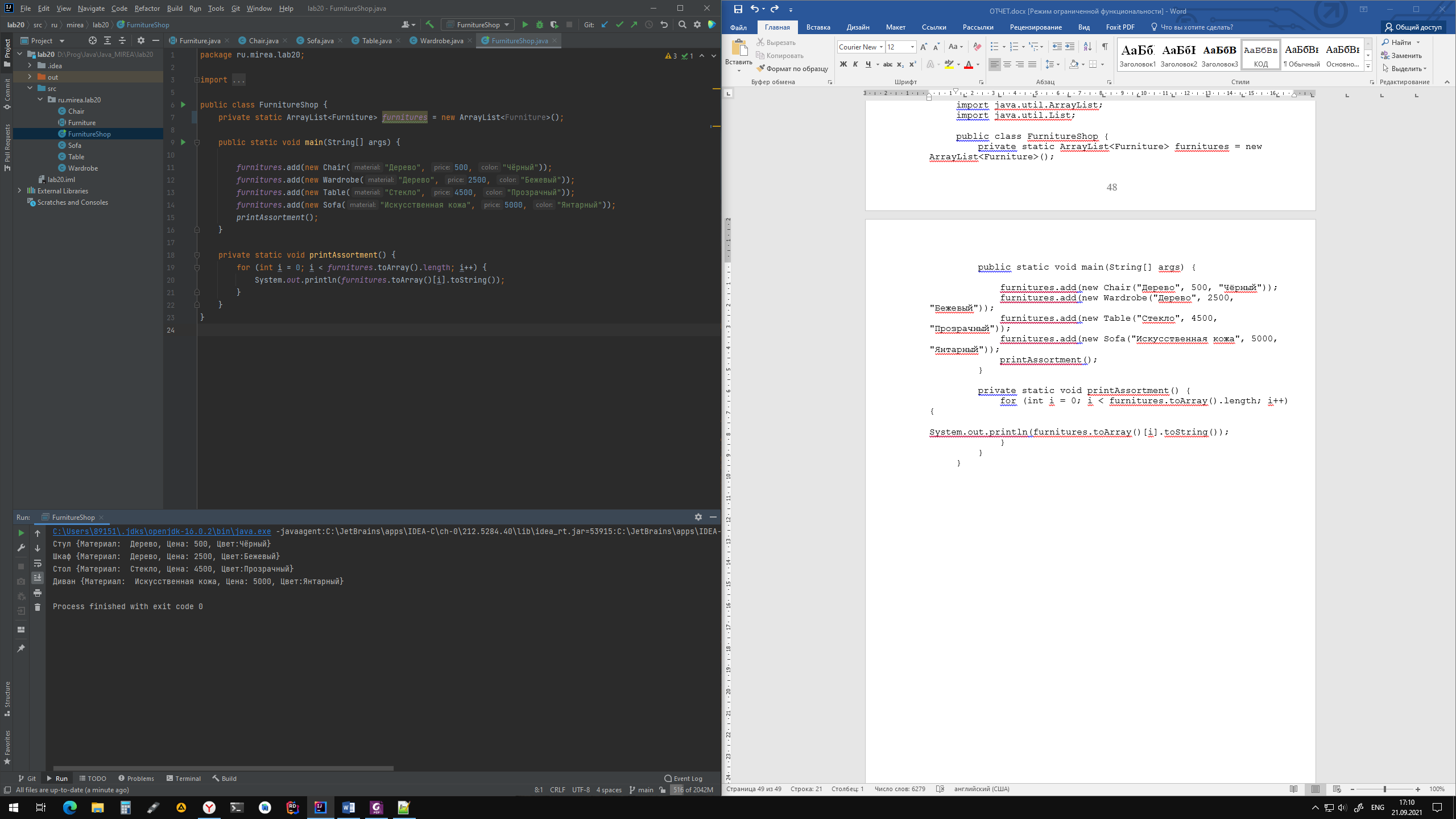


Рисунок 20.1 – Вывод программы.

Выводы по работе:

Выполняя данную практическую работу, я научился создавать абстрактный класс и наследовать от него другие классы.

Практическая работа № 21

Цель работы:

Цель данной работы - изучить понятие интерфейса, научиться создавать интерфейсы в Java и применять их в программах.

Теоретическое введение

Механизм наследования очень удобен, но он имеет свои ограничения. В частности, мы можем наследовать только от одного класса, в отличие, например, от языка С++, где имеется множественное наследование. В языке Java подобную проблему позволяют решить интерфейсы. Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов. Чтобы определить интерфейс, используется ключевое слово interface.

Интерфейс может определять различные методы, которые, так же как и абстрактные методы абстрактных классов не имеют реализации. В данном случае объявлен только один метод.

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

Чтобы класс применил интерфейс, надо использовать ключевое слово implements

Выполнение лабораторной работы

Задание:

Создать интерфейс Nameable, с методом getName(), возвращающим имя объекта, реализующего интерфейс. Проверить работу для различных объектов (например, можно создать классы, описывающие разные сущности, которые могут иметь имя: планеты, машины, животные и т. д.)

Решение:

Создадим интерфейс Nameable, в котором создадим метод возврата имени объекта.

package ru.mirea.lab21;

public interface Nameable {

String getName();

}

Далее создадим классы, реализующие интерфейс. К примеру можно создать класс машины, города и планеты. В каждом создаём конструктор и переписываем метод получения имени.

package ru.mirea.lab21;

public class Car implements Nameable {

private String name;

public Car(String name) {

this.name = name;

}

@Override

public String getName() {

return name;

}

}

package ru.mirea.lab21;

public class City implements Nameable{

private String name;

public City(String name) {

this.name = name;

}

@Override

public String getName() {

return name;

}

}

package ru.mirea.lab21;

public class Planet implements Nameable {

private String name;

public Planet(String name) {

this.name = name;

}

@Override

public String getName() {

return name;

}

}

Также создадим класс TestNameable для проверки работоспособности программы.

package ru.mirea.lab21;

public class TestNameable {

public static void main(String[] args) {

Car car = new Car("BMW");

Planet planet = new Planet("Earth");

City city = new City("Moscow");

System.out.println("Car is " + car.getName());

System.out.println("Planet is " + planet.getName());

System.out.println("City is " + city.getName());

}

}

На рисунке 21.1 показан вывод программы. По нему можно убедится, что программа работает правильно.

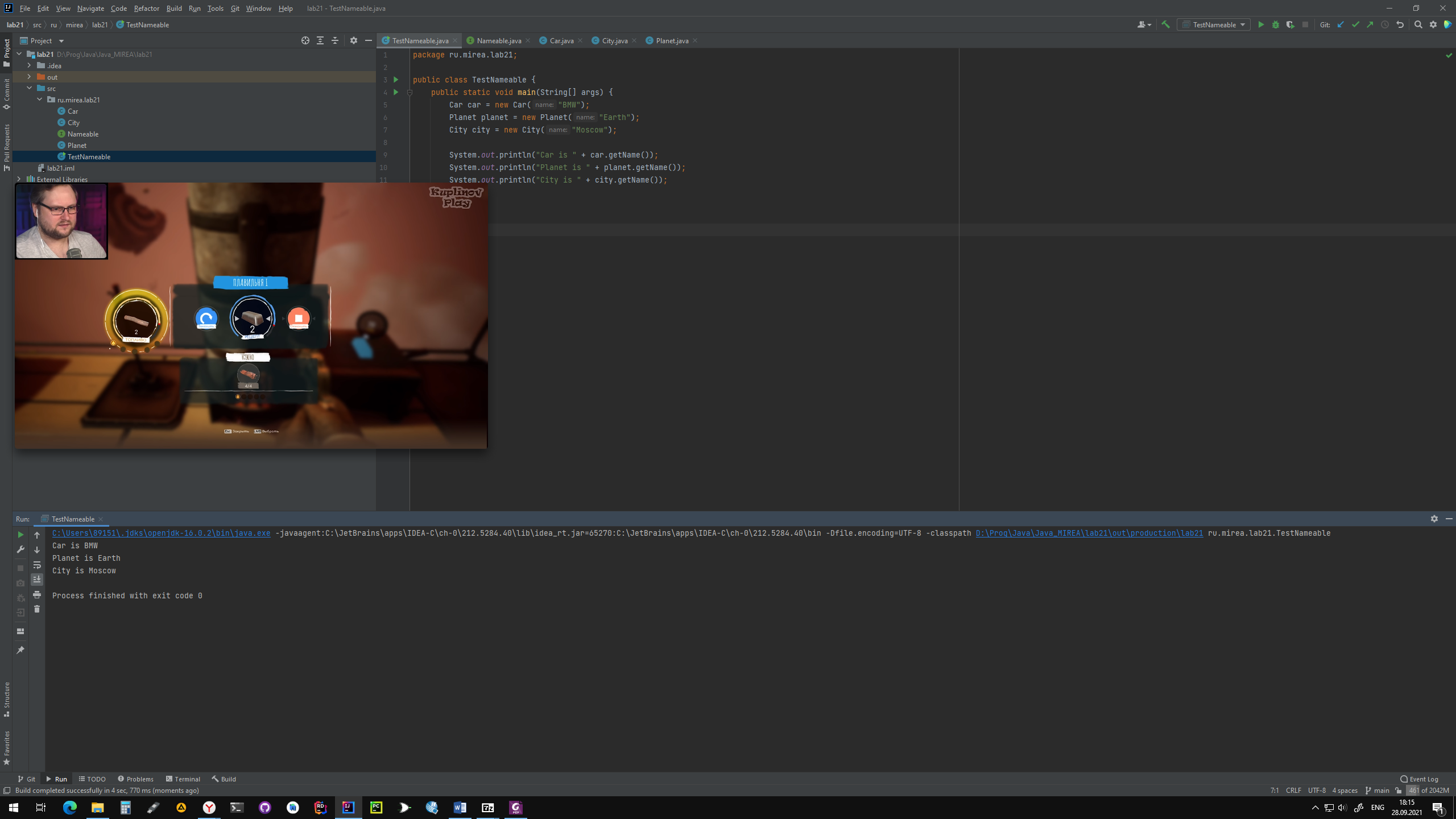


Рисунок 21.1 – Вывод программы.

Выводы по работе:

Выполняя данную практическую работу, я научился создавать интерфейсы и переопределять методы в классах.

Практическая работа № 22

Цель работы:

Цель данной работы - научиться создаватьграфический интерфейс пользователя, освоить на практике работу сразличными объектами для создания ГИП, менеджерами размещениякомпонентов.

Теоретическое введение

Для создания графического интерфейса пользователя можно использовать стандартную Java библиотеку Swing или AWT. В этих библиотеках имеются различные классы, позволяющие создавать окна, кнопки, текстовые поля, меню и другие объекты.

Text Fields - текстовое поле или поля для ввода текста (можно ввести только одну строку). Примерами текстовых полей являются поля для ввода логина и пароля, например, используемые, при входе в электронную почту.

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

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:

Создать окно, реализовать анимацию, с помощью картинки,состоящей из нескольких кадров.

Решение:

Создаём класс Frame и в его конструкторе описываем свойства окна, загружаем картинки в массив и по очереди выводим их на экран.

package ru.mirea.lab22;

import javax.imageio.ImageIO;

import javax.swing.\*;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

public class Frame extends JFrame {

Scanner scan = new Scanner(System.in);

Frame() {

super("Animation");

setVisible(true);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setSize(300, 300);

setResizable(false);

ArrayList<Image> images = new ArrayList<>();

try {

for (int i = 0; i < 6; i++) {

BufferedImage img = ImageIO.read(new File("src/dice\_images/dice\_" + (i + 1) + ".png"));

Image icon = new ImageIcon(img).getImage();

images.add(icon);

}

} catch (IOException e) {

e.printStackTrace();

}

System.out.println("Images has been uploaded!");

int ImgNumber = 1;

while (true) {

if (ImgNumber > 5) ImgNumber = 0;

drawIcon(images.get(ImgNumber));

ImgNumber++;

try {

Thread.sleep(150);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public void drawIcon(Image img) {

Graphics g = getGraphics();

g.drawImage(img, 50, 50, 200, 200, null);

}

}

Также создаем класс для проверки окна. В нём создаём экземпляр окна.

package ru.mirea.lab22;

public class TestFrame {

public static void main(String[] args) {

Frame frame = new Frame();

}

}

На рисунках 22.1 и 22.2 показано окно программы с анимацией. По нему можно убедится, что программа работает правильно.

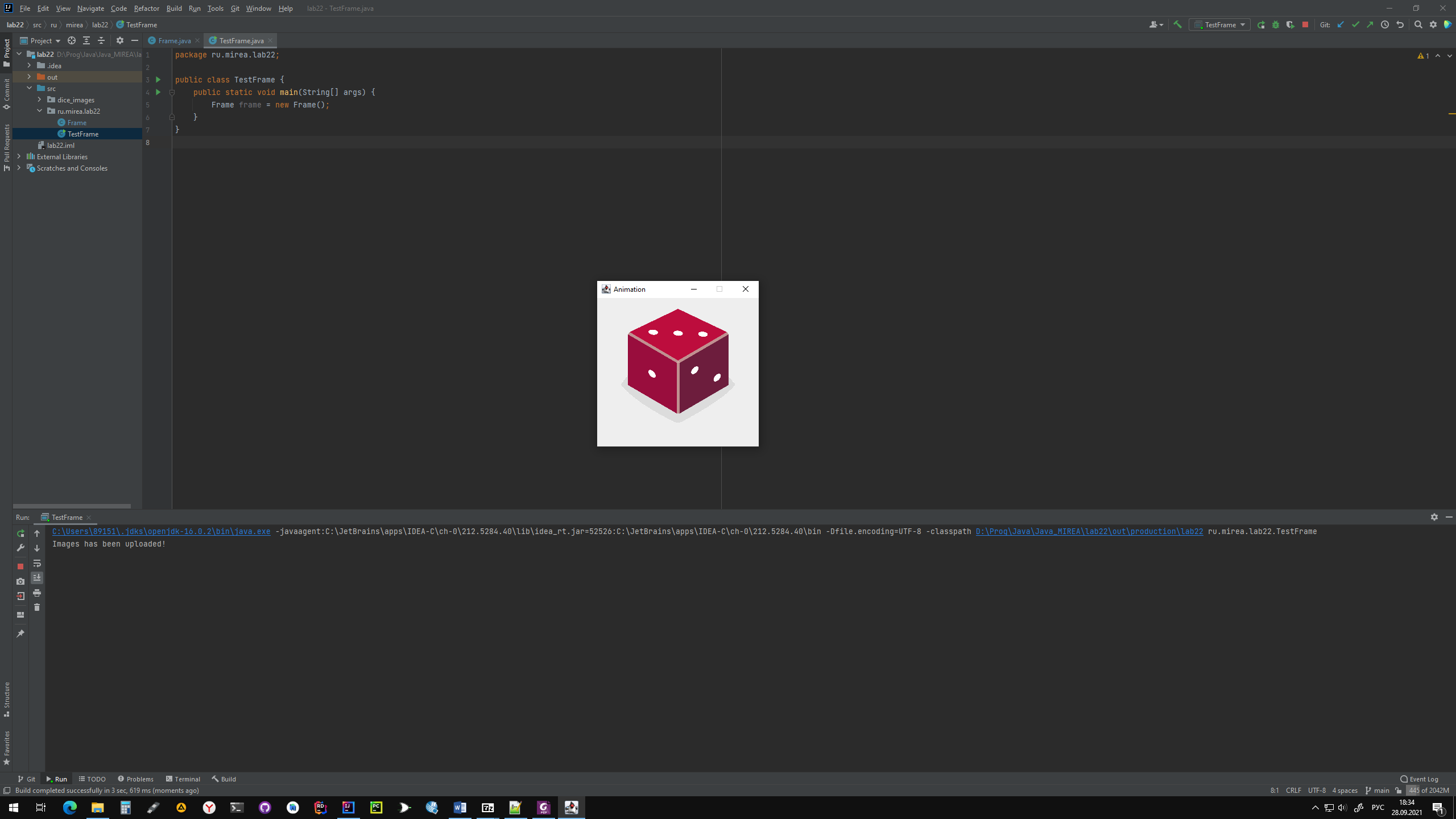


Рисунок 22.1 – Окно программы.

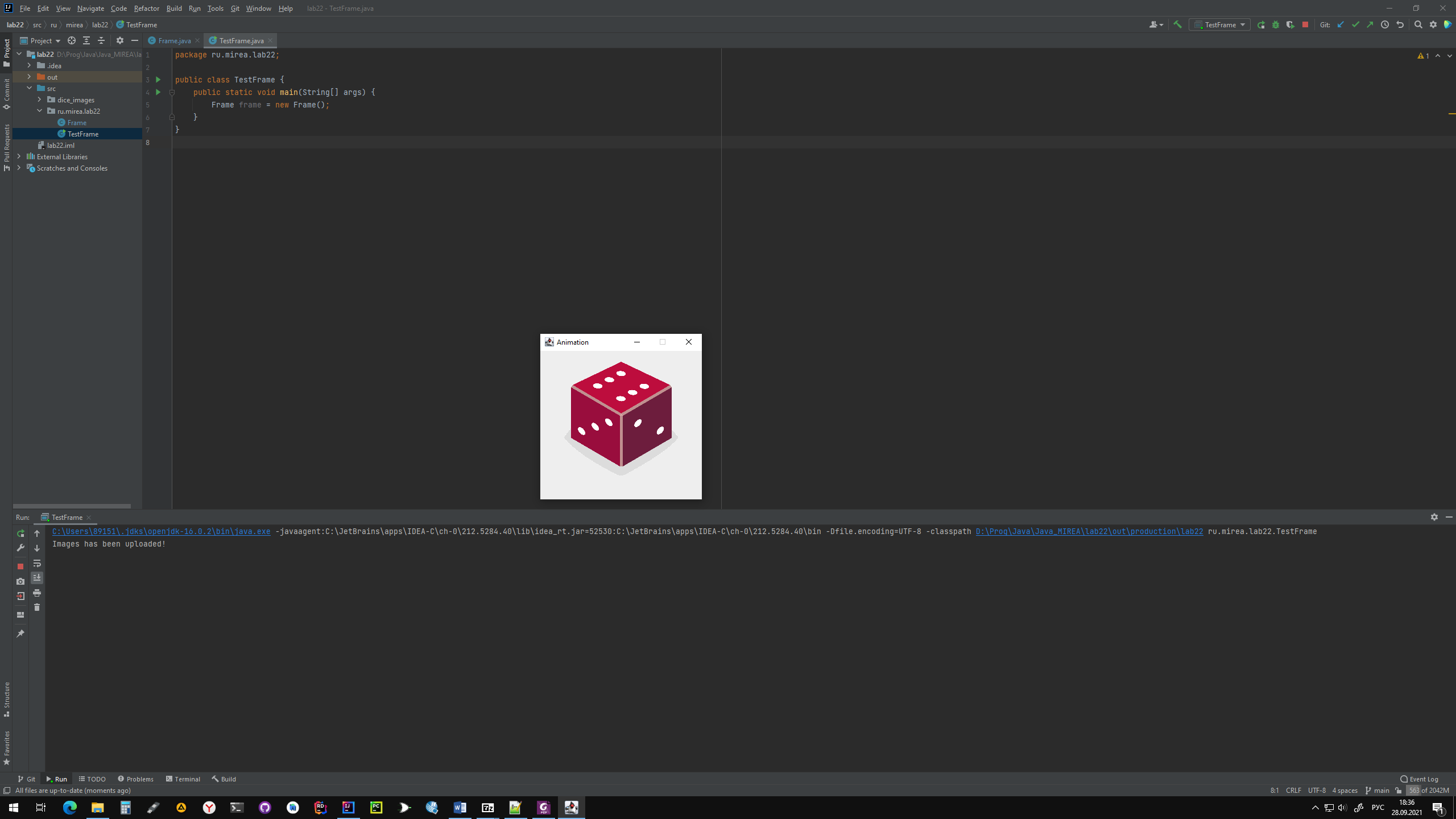


Рисунок 22.2 – Окно программы.

Выводы по работе:

Выполняя данную практическую работу, я научился создавать графические интерфейсы и работать с файловой системой.

Практическая работа № 23

Цель работы:

Цель данной лабораторной работы - научиться обрабатыватьразличные события для разных компонентов (кнопок, меню и т. д.).

Теоретическое введение

В контексте графического интерфейса пользователя наблюдаемыми объектами являются элементы управления: кнопки, флажки, меню и т.д. Они могут сообщить своим наблюдателям об определенных событиях, как элементарных (наведение мышкой, нажатие клавиши на клавиатуре), так и о высокоуровневых (изменение текста в текстовом поле, выбор нового элемента в выпадающем списке и т.д.). Наблюдателями должны являться объекты классов, поддерживающих специальные интерфейсы (в классе наблюдателя должны быть определенные методы, о которых «знает» наблюдаемый и вызывает их при наступлении события). Такие классы в терминологии Swing называются слушателями.

События от мыши — один из самых популярных типов событий. Практически любой элемент управления способен сообщить о том, что на него навели мышь, щелкнули по нему и т.д. Об этом будут оповещены все зарегистрированные слушатели событий от мыши.

Чтобы обработать нажатие на кнопку, требуется описать класс, реализующий интерфейс MouseListener. Далее необходимо создать объект этого класса и зарегистрировать его как слушателя интересующей нас кнопки. Для регистрации слушателя используется метод addMouseListener().

Выполнение лабораторной работы

Задание:

Реализация программу на Java с JTextArea и двумя меню: Цвет: который имеет возможность выбора из три возможных: синий, красный и черный Шрифт: три вида: “Times New Roman”, “MS Sans Serif”, “Courier New”.

Вы должны написать программу, которая с помощью меню, может изменять шрифт и цвет текста, написанного в JTextArea

Решение:

Создаём класс Frame для описания свойств окна. В нём также определяем кнопки, место для текста и обработчики нажатий кнопок.

package ru.mirea.lab23;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Frame extends JFrame {

private JTextArea text = new JTextArea();

private JButton blue = new JButton("Blue");

private JButton red = new JButton("Red");

private JButton black = new JButton("Black");

private JButton TNR = new JButton("Times New Roman");

private JButton Sans = new JButton("MS Sans Serif");

private JButton CN = new JButton("Courier New");

Frame() {

super("Text Editor");

setVisible(true);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(500, 500);

JPanel color = new JPanel();

color.add(blue);

color.add(red);

color.add(black);

JPanel font = new JPanel();

font.add(TNR);

font.add(Sans);

font.add(CN);

getContentPane().setLayout(new GridLayout(3, 1));

getContentPane().add(text);

getContentPane().add(color);

getContentPane().add(font);

blue.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

text.setForeground(Color.BLUE);

}

});

red.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

text.setForeground(Color.RED);

}

});

black.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

text.setForeground(Color.BLACK);

}

});

TNR.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

text.setFont(new Font("Times New Roman", Font.PLAIN, 14));

}

});

Sans.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

text.setFont(new Font("MS Sans Serif", Font.PLAIN, 14));

}

});

CN.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

text.setFont(new Font("Courier New", Font.PLAIN, 14));

}

});

}

}

Также создаем класс для проверки окна. В нём создаём экземпляр окна.

package ru.mirea.lab23;

public class TestFrame {

public static void main(String[] args) {

Frame frame = new Frame();

}

}

На рисунках 23.1,23.2 и 23.3 показано окно программы с разными состояниями. По нему можно убедится, что программа работает правильно.

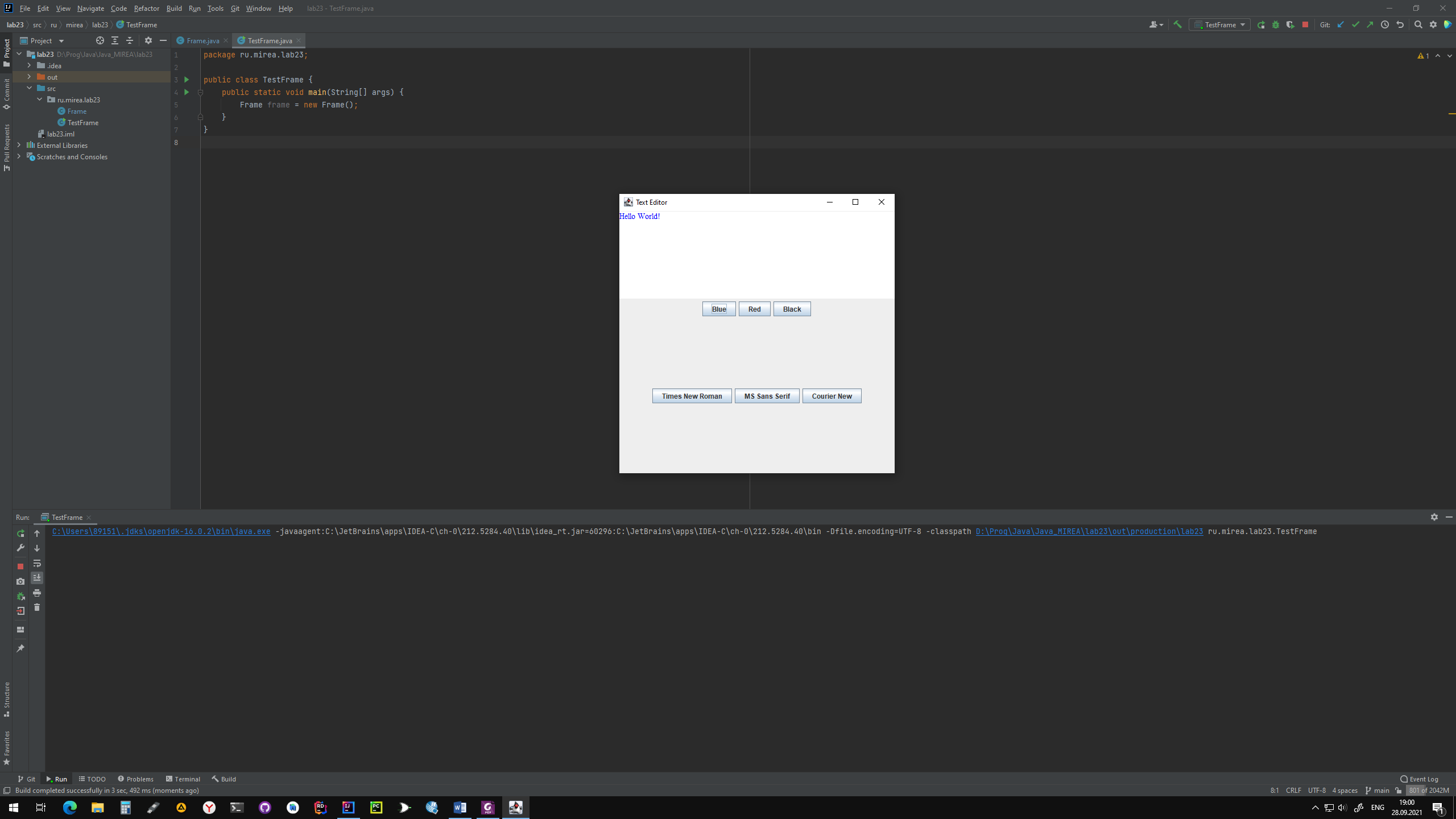


Рисунок 23.1 – Окно программы.

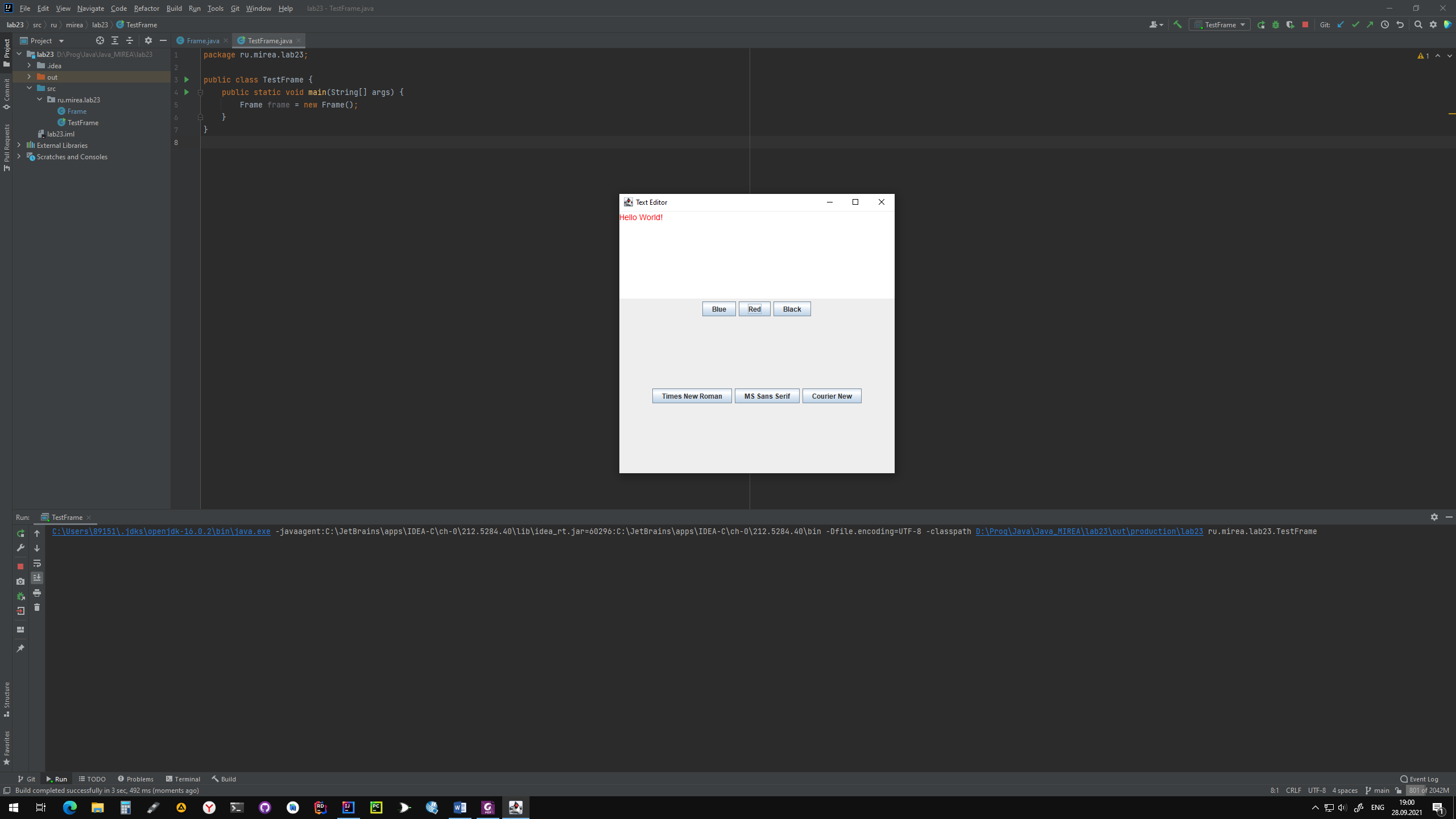


Рисунок 23.2 – Окно программы.

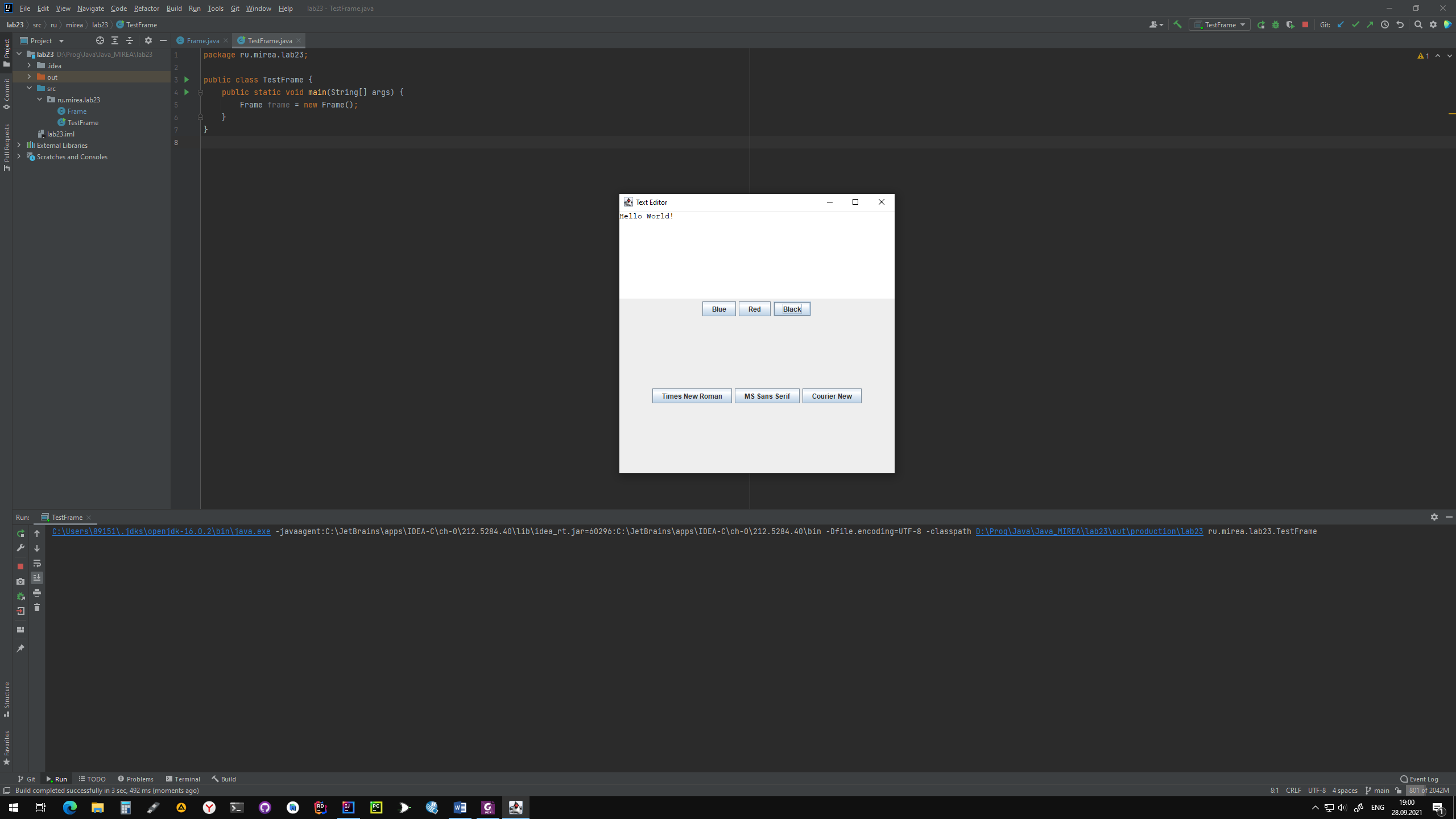


Рисунок 23.3 – Окно программы.

Выводы по работе:

Выполняя данную практическую работу, я научился создавать графические интерфейсы и работать с заданием шрифтов и цветов.

Практическая работа № 24

Цель работы:

Целью данной лабораторной работы является изучение работы с различными коллекциями в Java.

Теоретическое введение

Для хранения наборов данных в Java предназначены массивы.Однако их не всегда удобно использовать, прежде всего потому, чтоони имеют фиксированную длину. Эту проблему в Java решаютколлекции. Однако суть не только в гибких по размеру наборахобъектов, но в и том, что классы коллекций реализуют различныеалгоритмы и структуры данных, например, такие как стек, очередь,дерево и ряд других.Хотя в Java существует множество коллекций, но все ониобразуют стройную и логичную систему. Во-первых, в основе всехколлекций лежит применение того или иного интерфейса, которыйопределяет базовый функционал. Среди этих интерфейсов можновыделить следующие:

* Collection: базовый интерфейс для всех коллекций и другихинтерфейсов коллекций**.**
* Queue: наследует интерфейс Collection и представляетфункционал для структур данных в виде очереди**.**
* Deque: наследует интерфейс Queue и представляет функционалдля двунаправленных очередей.
* List: наследует интерфейс Collection и представляетфункциональность простых списков**.**
* Set: также расширяет интерфейс Collection и используется дляхранения множеств уникальных объектов**.**
* SortedSet: расширяет интерфейс Set для создания сортированныхколлекций**.**
* NavigableSet: расширяет интерфейс SortedSet для создания коллекций, в которых можно осуществлять поиск по соответствию
* Map: предназначен для созданий структур данных в виде словаря, где каждый элемент имеет определенный ключ и значение. В отличие от других интерфейсов коллекций не наследуется от интерфейса Collection

Эти интерфейсы частично реализуются абстрактными классами:

* AbstractCollection: базовый абстрактный класс для других коллекций, который применяет интерфейс Collection.
* AbstractList: расширяет класс AbstractCollection и применяет интерфейс List, предназначен для создания коллекций в виде списков.
* AbstractSet: расширяет класс AbstractCollection и применяет интерфейс Set для создания коллекций в виде множеств.
* AbstractQueue: расширяет класс AbstractCollection и применяет интерфейс Queue, предназначен для создания коллекций в виде очередей и стеков.
* AbstractMap: также расширяет класс AbstractCollection и применяет интерфейс Map, предназначен для создания наборов по типу словаря с объектами в виде пары "ключ-значение".
* AbstractSequentialList: также расширяет класс AbstractList и реализует интерфейс List. Используется для создания связанных списков.

С помощью применения вышеописанных интерфейсов и абстрактных классов в Java реализуется широкая палитра классов коллекций - списки, множества, очереди, отображения и другие, среди которых можно выделить следующие:

* ArrayList: простой список объектов
* LinkedList: представляет связанный список
* ArrayDeque: класс двунаправленной очереди, в которой мы можем произвести вставку и удаление как в начале коллекции, так и в ее конце
* HashSet: набор объектов или хеш-множество, где каждый элемент имеет ключ - уникальный хеш-код
* TreeSet: набор отсортированных объектов в виде дерева
* LinkedHashSet: связанное хеш-множество
* PriorityQueue: очередь приоритетов
* HashMap: структура данных в виде словаря, в котором каждый объект имеет уникальный ключ и некоторое значение
* TreeMap

Выполнение лабораторной работы

Задание:

Создать свою коллекцию, такую же, как и ArrayList.

Решение:

Создаём класс для коллекции MyList. В нём определяем неизменяемые поля для ёмкости по-умолчанию, текущей ёмкости, заданной ёмкости, а также два конструктора, методы для расширения массива, получения его элементов и их удаления.

package ru.mirea.lab24;

public class MyList<T> {

private final int INIT\_SIZE = 16;

private final int CUT\_RATE = 4;

private Object[] array;

private int pointer = 0;

public int size = INIT\_SIZE;

public MyList(int size) {

this.size = size;

array = new Object[size];

}

public MyList() {

array = new Object[INIT\_SIZE];

}

public void add(T item) {

if (pointer == array.length - 1)

resize(array.length \* 2);

array[pointer++] = item;

}

public T get(int index) {

return (T) array[index];

}

public void remove(int index) {

for (int i = index; i < pointer; i++)

array[i] = array[i + 1];

array[pointer] = null;

pointer--;

size--;

if (array.length > INIT\_SIZE && pointer < array.length / CUT\_RATE)

resize(array.length / 2);

}

public int Count() {

return pointer;

}

private void resize(int newLength) {

Object[] newArray = new Object[newLength];

System.arraycopy(array, 0, newArray, 0, pointer);

array = newArray;

}

}

Также создаём класс для проверки коллекции. В нём создаём метод для входа в программу, и метод печати массива на экран. Также создаём целочисленный массив для тестов и заполняем его десятью случайными числами. Затем проверяем записались ли они. Затем удаляем два первых элемента и снова выводим.

package ru.mirea.lab24;

public class TestList {

public static void main(String[] args) {

MyList<Integer> my1 = new MyList<Integer>(10);

for (int i = 0; i < my1.size; i++) {

int number = (int) (Math.random() \* 100);

my1.add(number);

System.out.print(number+"\t");

}

System.out.println();

printArray(my1);

printArray(my1);

my1.remove(0);

my1.remove(0);

printArray(my1);

}

public static void printArray(MyList<Integer> array) {

for (int i = 0; i < array.size; i++) {

System.out.print(array.get(i) + "\t");

}

System.out.println();

}

}

На рисунках 24.1 показан вывод программы. По нему можно убедится, что программа работает правильно.

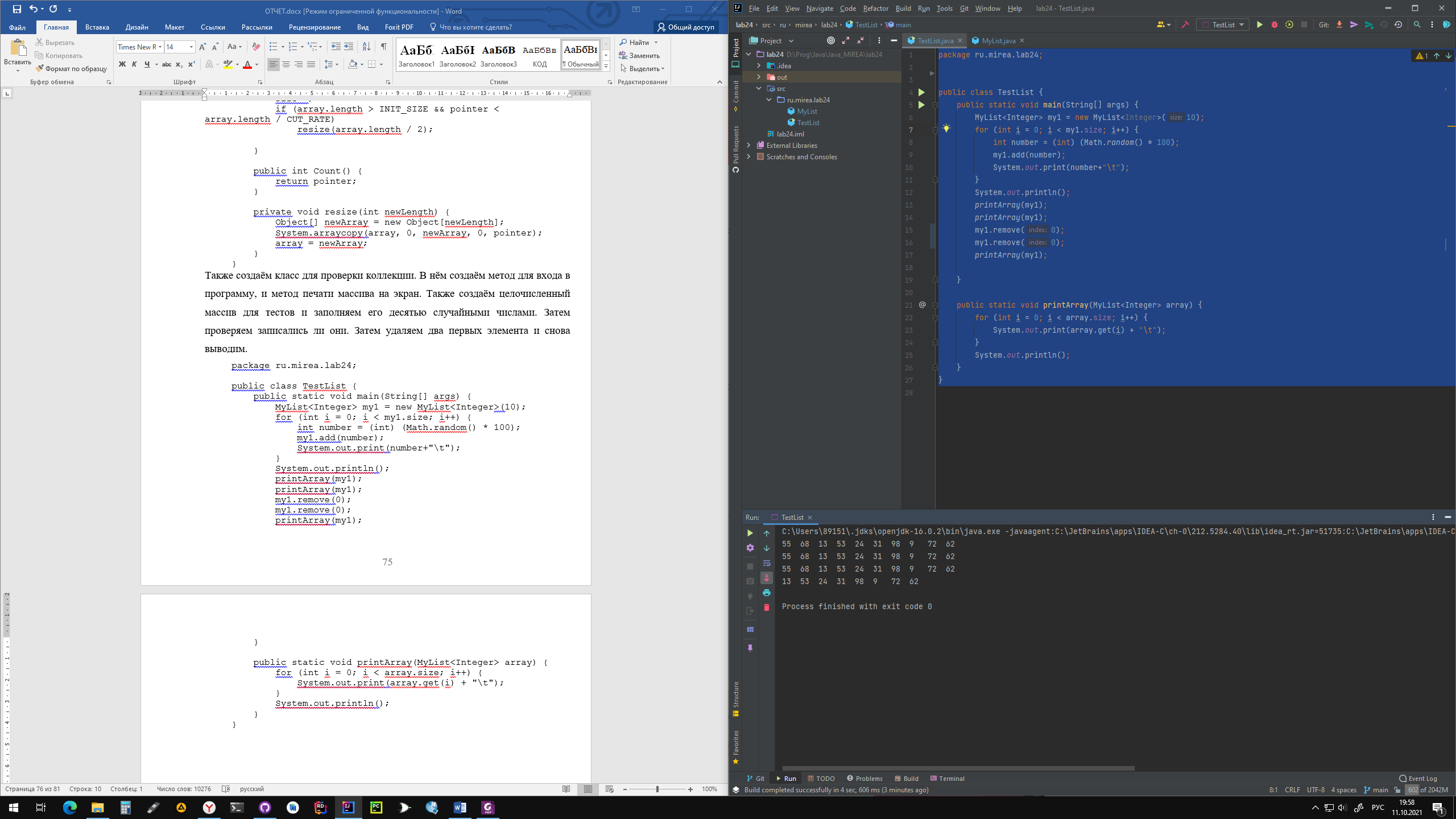


Рисунок 24.1 – Вывод программы.

Выводы по работе:

Выполняя данную практическую работу, я научился создавать собственную коллекцию.

Практическая работа № 25

Цель работы:

Освоить на практике работу с файлами на языке Java. Получить практические навыки по чтению и записи данных в файл.

Теоретическое введение

Запись файлов(Класс FileWriter)

Класс FileWriter является производным от класса Writer. Он используется для записи текстовых файлов. Чтобы создать объект FileWriter, можно использовать один из следующих конструкторов:

* FileWriter(File file)
* FileWriter(File file, boolean append)
* FileWriter(FileDescriptor fd)
* FileWriter(String fileName)
* FileWriter(String fileName, boolean append)

Так, в конструктор передается либо путь к файлу в виде строки, либо объект File, который ссылается на конкретный текстовый файл. Параметр append указывает, должны ли данные дозаписываться в конецфайла (если параметр равен true), либо файл должен перезаписываться. В конструкторе использовался параметр append со значением false - то есть файл будет перезаписываться. Затем с помощью методов, пределенных в базовом классе Writer производится запись данных.

Чтение файлов(Класс FileReader)

Для создания объекта FileReader мы можем использовать один из его конструкторов:

* FileReader(String fileName)
* FileReader(File file)
* FileReader(FileDescriptor fd)

Класс FileReader наследуется от абстрактного класса Reader и предоставляет функциональность для чтения текстовых файлов.

Выполнение лабораторной работы

Задание:

Реализовать запись в файл введённой с клавиатуры информации

Решение:

Создаём класс Main, а в нём метод для запуска программы. Открываем файл для записи в цикле считываем данные с клавиатуры и записываем в файл.

package ru.mirea.lab25;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

FileWriter writer;

Scanner scan = new Scanner(System.in);

System.out.println("Write something to save");

try {

writer = new FileWriter("src/ru/mirea/lab25/Text.txt",true);

boolean isWork = true;

while (isWork) {

String text = scan.nextLine();

if (text.equals(":q")) isWork = false;

else {

writer.write(text+"\n");

writer.flush();

}

}

writer.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

На рисунках 25.1 и 25.2 показан результат успешной работы программы.

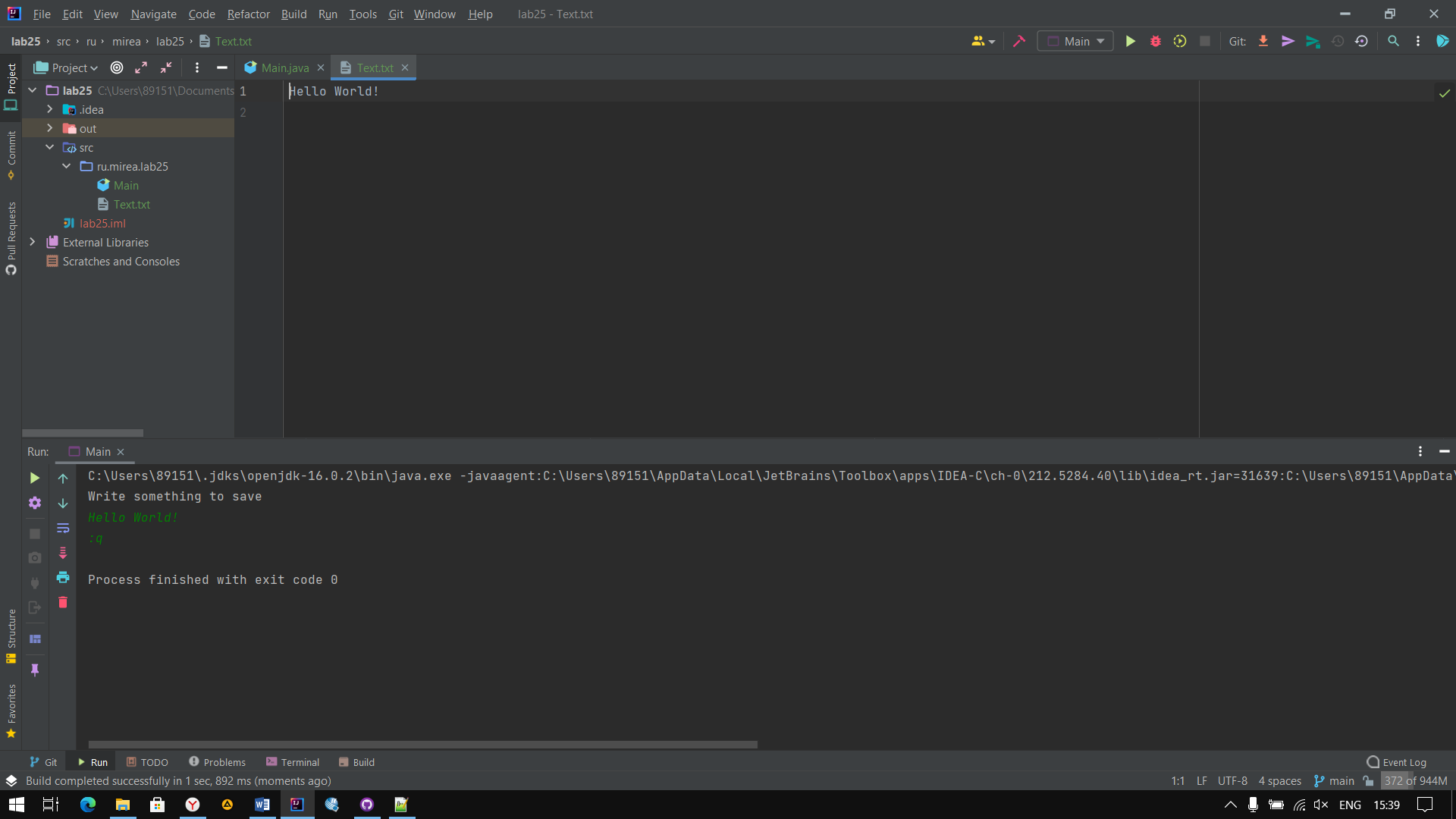


Рисунок 25.1 – Вывод программы.

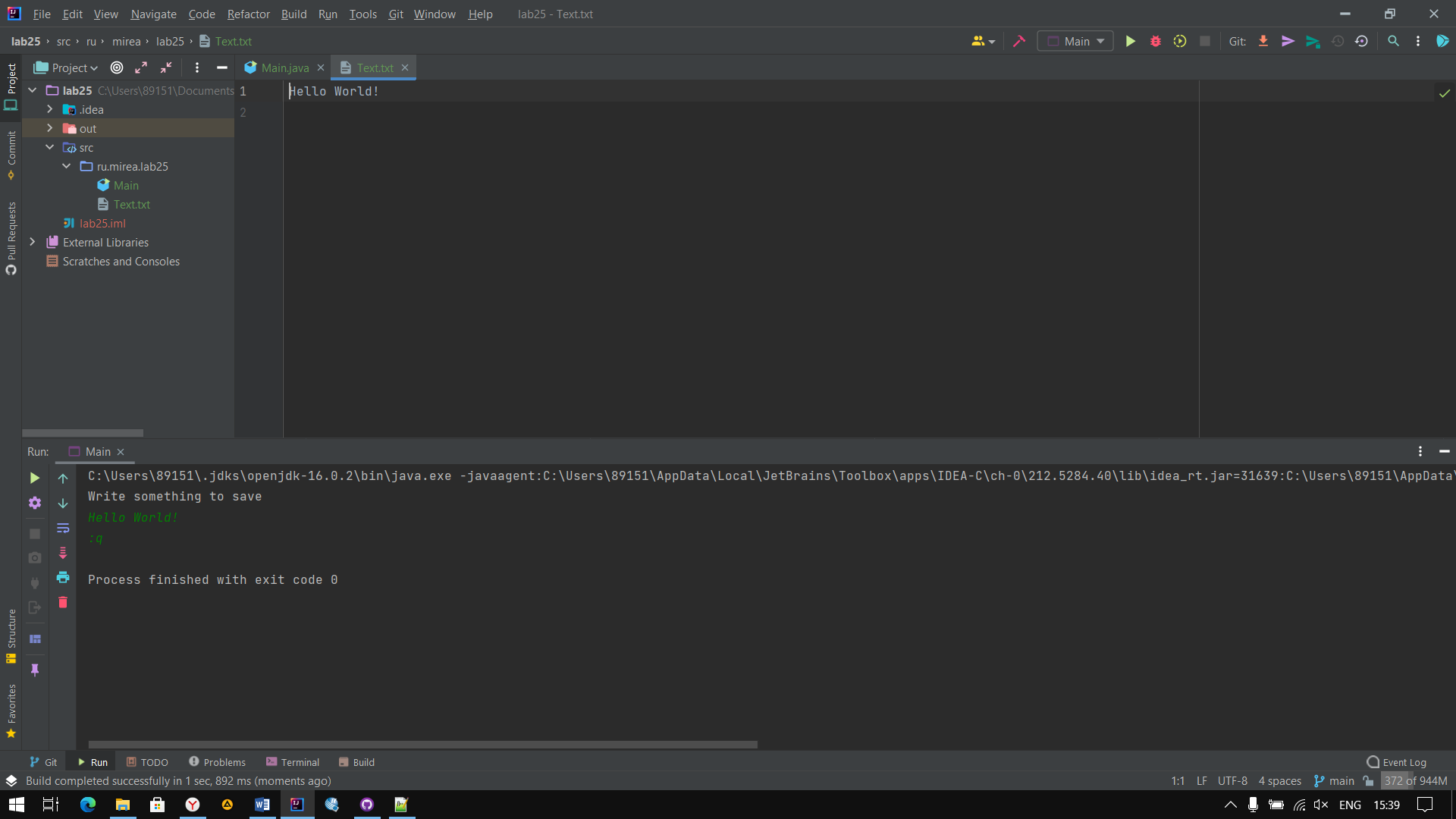


Рисунок 25.2 – Удачная запись текста в файл

Выводы по работе:

Выполняя данную практическую работу, я научился работать с файлами.