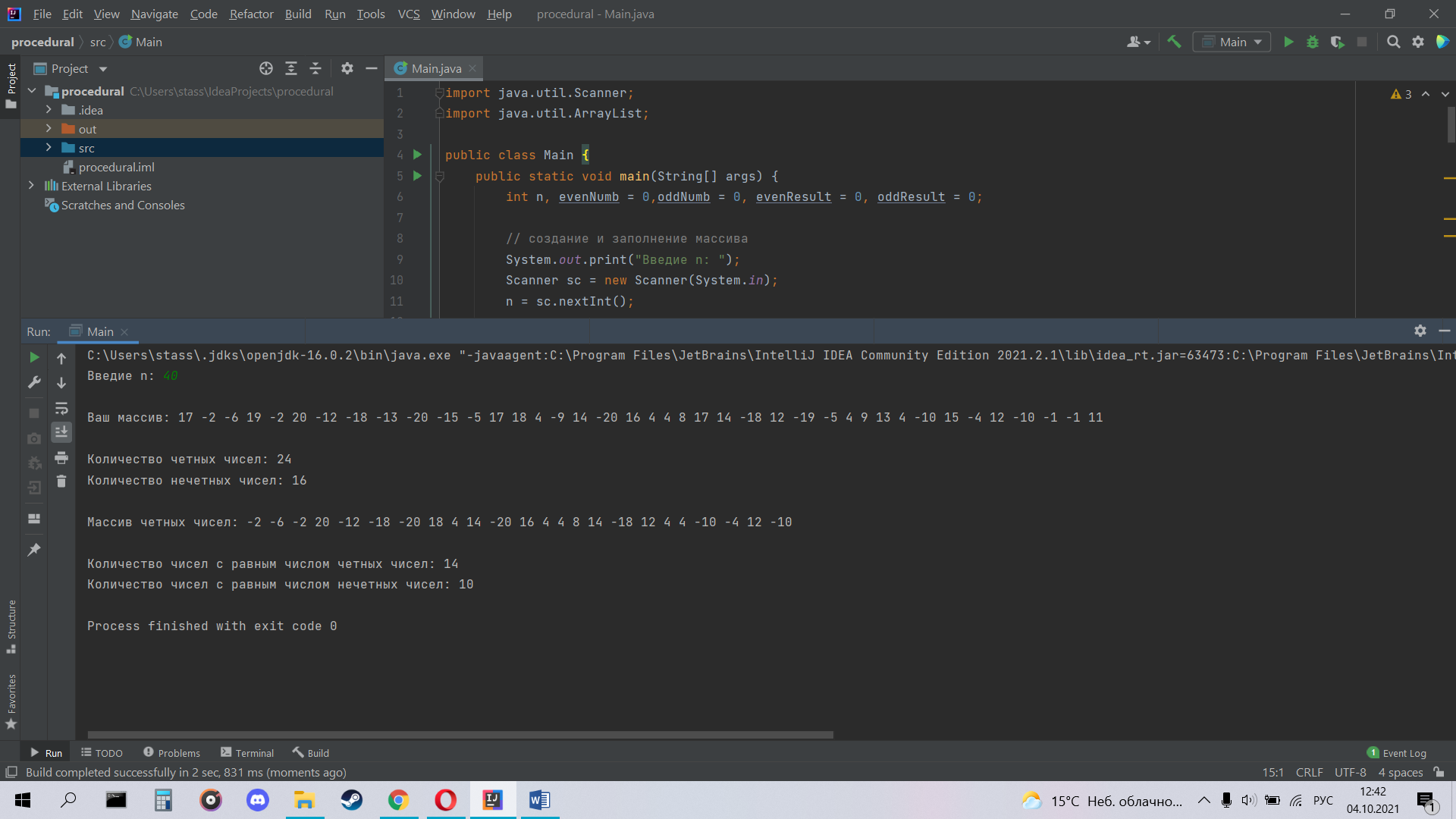
1) Ввести n чисел з консолі. Знайти кількість парних чисел, а серед них - кількість чисел з рівним числом парних і непарних чисел.

package eSForgary;

import java.util.Scanner;  
import java.util.ArrayList;  
  
public class Main {  
 public static void main(String[] args) {  
 int n, evenNumb = 0,oddNumb = 0, evenResult = 0, oddResult = 0;  
  
 // создание и заполнение массива  
 System.*out*.print("Введие n: ");  
 Scanner sc = new Scanner(System.*in*);  
 n = sc.nextInt();  
  
 System.*out*.print("\nВаш массив: ");  
 int[] array = new int[n];  
 for (int i = 0; i < array.length; i++) {  
 array[i] = (int) Math.*round*((Math.*random*() \* 40) - 20);  
 System.*out*.print(array[i] + " ");  
 }  
  
 // подсчет количества четных и не четных элементов   
 ArrayList<Integer> evenArray = new ArrayList<>();  
 for (int i = 0; i < array.length; i++) {  
 if (array[i] % 2 == 0) {  
 evenArray.add(array[i]);  
 evenNumb++;  
 }  
 else oddNumb++;  
 }  
 System.*out*.println("\n\nКоличество четных чисел: " + evenNumb);  
 System.*out*.println("Количество нечетных чисел: " + oddNumb);  
  
 // массив четных элементов   
 System.*out*.print("\nМассив четных чисел: ");  
 for (int i = 0; i < evenArray.size(); i++) {  
 System.*out*.print(evenArray.get(i) + " ");  
 }  
  
 // подсчет количество чисел с равным числом четных и нечетных цифр  
 for (int i = 0; i < evenArray.size(); i++) {  
 if (*countsDigits*(evenArray.get(i)) % 2 == 0) evenResult++;  
 else if (*countsDigits*(evenArray.get(i)) % 2 != 0) oddResult++;  
 }  
 System.*out*.println();  
 System.*out*.println("\nКоличество чисел с равным числом четных чисел: " + evenResult);  
 System.*out*.println("Количество чисел с равным числом нечетных чисел: " + oddResult);  
 }  
   
 // метод для подсчета количества цифр в числе   
 public static int countsDigits(long number) {  
 int count = (number == 0) ? 1 : 0;  
 while (number != 0) {  
 count++;  
 number /= 10;  
 }  
 return count;  
 }  
}

результат:



2) а) Визначити клас Точка на площині (в просторі) і в часі. Задати рух точки в певному напрямку. Створити методи по визначенню швидкості точки. Перевірити для двох точок можливість перетину траєкторій. Визначити відстань між двома точками в заданий момент часу.

@Main.java

package eSForgary;

public class Main {  
 public static void main(String[] args) {  
  
 Point a\_start = new Point(1,2);  
 Point a\_end = new Point(3,1);  
  
 Point b\_start = new Point(1,1);  
 Point b\_end = new Point(4,3);  
  
 Segment a = new Segment(a\_start, a\_end);  
 Segment b = new Segment(b\_start, b\_end);  
  
 System.*out*.println("- - - - - - - - - - - - - - - - - - ");  
 System.*out*.println("Determination of the speed of point A moving along its trajectory: " + Operation.*speed*(a));  
 System.*out*.println("Determination of the speed of point B moving along its trajectory: " + Operation.*speed*(b));  
  
 System.*out*.println("- - - - - - - - - - - - - - - - - - ");  
 System.*out*.println("distance between A and B: " + Operation.*distance*(a\_start, b\_start));  
  
 System.*out*.println("- - - - - - - - - - - - - - - - - - ");  
 if (Operation.*checkSegmentIntersection*(a, b)) {  
 System.*out*.println("Trajectory of point A and trajectory of point B are intersect");  
 } else {  
 System.*out*.println("Trajectory of point A and trajectory of point B aren`t intersect");  
 }  
 }  
}

@Point.java

package eSForgary;

public class Point {  
 private double x;  
 private double y;  
  
 public Point(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public double getX() { return x; }  
 public double getY() { return y; }  
  
 @Override  
 public String toString() { return "Point{" + "x=" + x + ", y=" + y +"}"; }  
}

@Vector.java

package eSForgary;

public class Vector {  
 private Point startPoint;  
 private Point endPoint;  
 private double xComponent;  
 private double yComponent;  
  
 public Vector(Point startPoint, Point endPoint) {  
 super();  
 this.startPoint = startPoint;  
 this.endPoint = endPoint;  
 calculateComponent();  
 }  
  
 public double getxComponent() {  
 return xComponent;  
 }  
 public double getyComponent() {  
 return yComponent;  
 }  
  
 private void calculateComponent() {  
 xComponent = endPoint.getX() - startPoint.getX();  
 yComponent = endPoint.getY() - startPoint.getY();  
 }  
  
 @Override  
 public String toString() {  
 return "Vector [startPoint = " + startPoint + ", endPoint = " + endPoint + "xComponent = " + xComponent + ", yComponent = " + yComponent + "]";  
 }  
}

@Segment.java

package eSForgary;

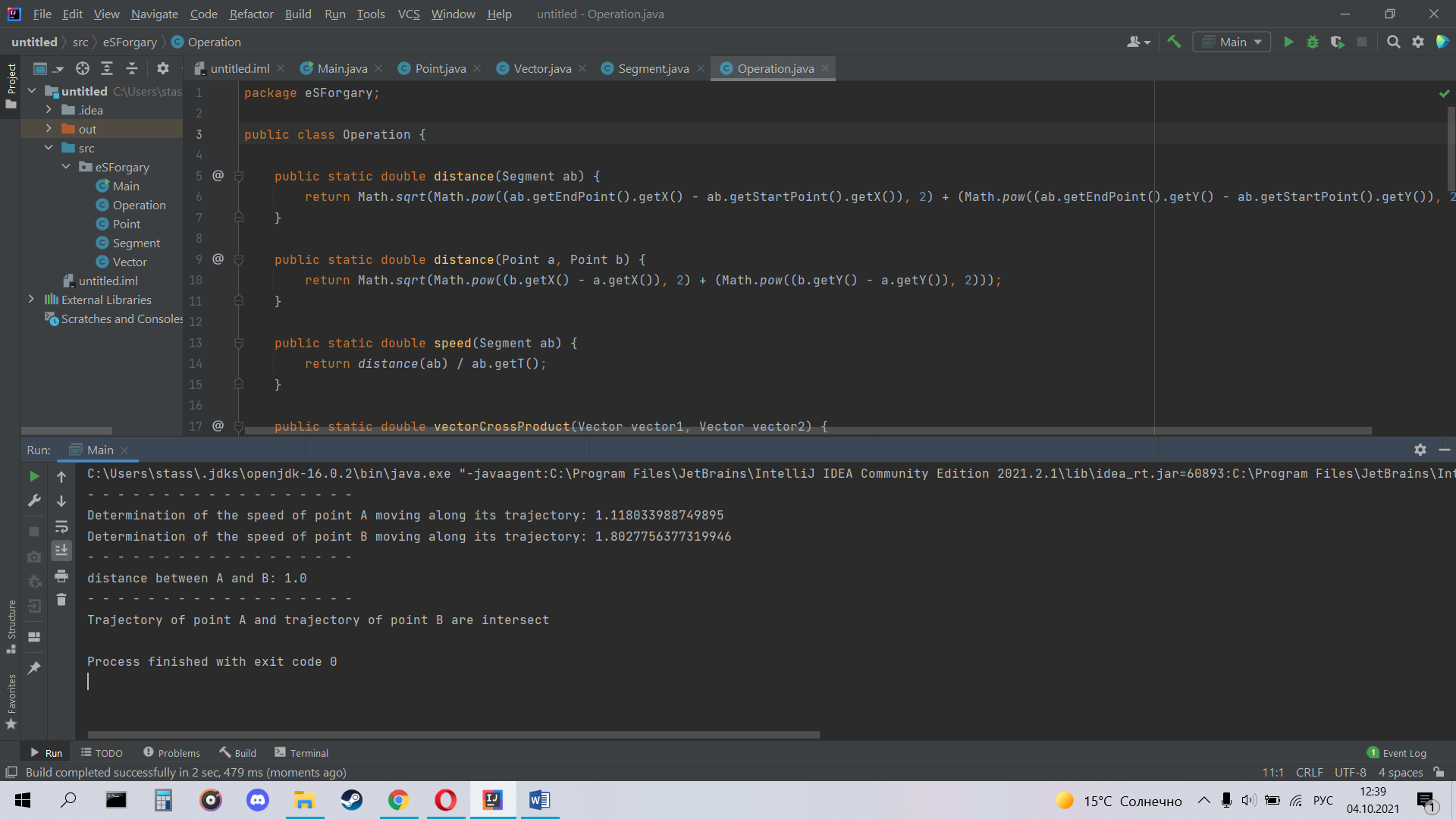
public class Segment {  
 private Point startPoint;  
 private Point endPoint;  
 private double t = 2;  
  
 public Segment(Point startPoint, Point endPoint) {  
 super();  
 this.startPoint = startPoint;  
 this.endPoint = endPoint;  
 }  
  
 public double getT() { return t; }  
 public Point getStartPoint() { return startPoint; }  
 public Point getEndPoint() { return endPoint; }  
  
 @Override  
 public String toString() {  
 return "Segment [startPoint = " + startPoint + ", endPoint = " + endPoint + "]";  
 }  
}

@Operation.java

package eSForgary;

public class Operation {  
  
 public static double distance(Segment ab) {  
 return Math.*sqrt*(Math.*pow*((ab.getEndPoint().getX() - ab.getStartPoint().getX()), 2) + (Math.*pow*((ab.getEndPoint().getY() - ab.getStartPoint().getY()), 2)));  
 }  
  
 public static double distance(Point a, Point b) {  
 return Math.*sqrt*(Math.*pow*((b.getX() - a.getX()), 2) + (Math.*pow*((b.getY() - a.getY()), 2)));  
 }  
  
 public static double speed(Segment ab) {  
 return *distance*(ab) / ab.getT();  
 }  
  
 public static double vectorCrossProduct(Vector vector1, Vector vector2) {  
 return vector1.getxComponent() \* vector2.getyComponent() - vector2.getxComponent() \* vector1.getyComponent();  
 }  
  
 public static boolean rangeIntersection(double a, double b, double c, double d) {  
 if (a > b) {  
 double temp = a;  
 a = b;  
 b = temp;  
 }  
 if (c > d) {  
 double temp = c;  
 c = d;  
 d = temp;  
 }  
 return Math.*max*(a, c) <= Math.*min*(b, d);  
 }  
  
 public static boolean boundingBox(Segment ab, Segment cd) {  
 boolean xRangeIntersection = *rangeIntersection*(ab.getStartPoint().getX(), ab.getEndPoint().getX(), cd.getStartPoint().getX(), cd.getEndPoint().getX());  
 boolean yRangeIntersection = *rangeIntersection*(ab.getStartPoint().getY(), ab.getEndPoint().getY(), cd.getStartPoint().getY(), cd.getEndPoint().getY());  
 return xRangeIntersection && yRangeIntersection;  
 }  
  
 public static boolean checkSegmentIntersection(Segment ab, Segment cd) {  
 if (!*boundingBox*(ab, cd)) { return false;}  
 Vector vAB = new Vector(ab.getStartPoint(), ab.getEndPoint());  
 Vector vAC = new Vector(ab.getStartPoint(), cd.getStartPoint());  
 Vector vAD = new Vector(ab.getStartPoint(), cd.getEndPoint());  
  
 Vector vCD = new Vector(cd.getStartPoint(), cd.getEndPoint());  
 Vector vCA = new Vector(cd.getStartPoint(), ab.getStartPoint());  
 Vector vCB = new Vector(cd.getStartPoint(), ab.getEndPoint());  
  
 double d1 = *vectorCrossProduct*(vAB, vAC);  
 double d2 = *vectorCrossProduct*(vAB, vAD);  
 double d3 = *vectorCrossProduct*(vCD, vCA);  
 double d4 = *vectorCrossProduct*(vCD, vCB);  
  
 return ((d1 <= 0 && d2 >= 0) || (d1 >= 0 && d2 <= 0)) && ((d3 <= 0 && d4 >= 0) || (d3 >= 0 && d4 <= 0));  
 }  
}

результат:



б) Визначити клас Чотирикутник на площині, вершини якого мають тип Точка. Визначити площу і периметр чотирикутника. Створити масив / список / безліч об'єктів і підрахувати кількість чотирикутників різного типу (квадрат, прямокутник, ромб, довільний). Визначити для кожної групи найбільший і найменший за площею (периметру) об'єкт.

@Main.java

package eSForgary;  
import java.util.\*;  
  
public class Main {  
  
 private List<Shape> quadrilaterals;  
  
 private void fillQuadrilateralsList() {  
 quadrilaterals = new ArrayList<>();  
 quadrilaterals.add(new Quadrilateral(new Point(1, 1), new Point(1, 4), new Point(7, 4), new Point(7, 1)));  
 quadrilaterals.add(new Quadrilateral(new Point(1, 2), new Point(3, 1), new Point(4, 2), new Point(5, 2)));  
 quadrilaterals.add(new Quadrilateral(new Point(3, 3), new Point(4, 6), new Point(1, 3), new Point(2, 6)));  
 quadrilaterals.add(new Quadrilateral(new Point(0, 0), new Point(0, 5), new Point(5, 5), new Point(5, 0)));  
 quadrilaterals.add(new Quadrilateral(new Point(1, 1), new Point(2, 4), new Point(5, 4), new Point(4, 1)));  
 quadrilaterals.add(new Quadrilateral(new Point(4, 2), new Point(2, 6), new Point(6, 8), new Point(8, 4)));  
 quadrilaterals.add(new Quadrilateral(new Point(-3, 0), new Point(0, 3), new Point(3, 0), new Point(0, -3)));  
 quadrilaterals.add(new Quadrilateral(new Point(0, 0), new Point(10, 3), new Point(15, 0), new Point(8, -3)));  
 quadrilaterals.add(new Quadrilateral(new Point(0, 0), new Point(0, 0), new Point(15, 0), new Point(8, -3)));  
 quadrilaterals.add(new Quadrilateral(new Point(1, 2), new Point(2, 4), new Point(5, 4), new Point(4, 1)));  
 }  
  
 private Map<QuadrilateralType, List<Shape>> splitByType() {  
 List<Shape> quadrilaterals = this.quadrilaterals;  
  
 Map<QuadrilateralType, List<Shape>> groups = new HashMap<>();  
  
 for (Shape shape : quadrilaterals) {  
 if (!groups.containsKey(shape.getType())) {  
 List<Shape> group = new ArrayList<>();  
 group.add(shape);  
 groups.put(shape.getType(), group);  
 } else {  
 groups.get(shape.getType()).add(shape);  
 }  
 }  
 return groups;  
 }  
  
 private void showQuadrilateralsInfo(){  
 fillQuadrilateralsList();  
 for (Map.Entry<QuadrilateralType, List<Shape>> shapes : splitByType().entrySet()) {  
  
 double maxS = 0;  
 double minS = Double.*MAX\_VALUE*;  
 double maxP = 0;  
 double minP = Double.*MAX\_VALUE*;  
 for (Shape shape: shapes.getValue()){  
 double shapeS = shape.getSquare();  
 double shapeP = shape.getPerimeter();  
 if(maxS < shapeS)  
 maxS = shapeS;  
 if(minS > shapeS)  
 minS = shapeS;  
 if(maxP < shapeP)  
 maxP = shapeP;  
 if(minP > shapeP)  
 minP = shapeP;  
 }  
 System.*out*.printf("Type: %s %nCount: %d %nmaxSquare: %.3f %nminSquare: %.3f %nmaxPerimeter: %.3f %nminPerimeter: %.3f %n", shapes.getKey(), shapes.getValue().size(), maxS, minS, maxP, minP);  
 System.*out*.println(shapes.getValue() + "\n");  
 }  
 }  
  
 public static void main(String[] args) {  
 Main main = new Main();  
 main.showQuadrilateralsInfo();  
 }  
}

@Point.java

package eSForgary;  
public class Point {  
  
 private double x;  
 private double y;  
  
 public Point(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public double getY() { return y; }  
 public double getX() { return x; }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
  
 Point point = (Point) o;  
  
 if (Double.*compare*(point.x, x) != 0) return false;  
 return Double.*compare*(point.y, y) == 0;  
 }  
  
 @Override  
 public int hashCode() {  
 int result;  
 long temp;  
 temp = Double.*doubleToLongBits*(x);  
 result = (int) (temp ^ (temp >>> 32));  
 temp = Double.*doubleToLongBits*(y);  
 result = 31 \* result + (int) (temp ^ (temp >>> 32));  
 return result;  
 }  
  
 @Override  
 public String toString() { return "V{" + x + ", " + y +'}'; }  
}

@Shape.java

package eSForgary;  
import java.util.Arrays;  
import java.util.List;  
  
public abstract class Shape {  
  
 protected List<Point> vertices;  
  
 public Shape(Point... vertices){  
 this.vertices = Arrays.*asList*(vertices);  
 }  
  
 protected double distance(Point v1, Point v2) {  
 return Math.*sqrt*(Math.*pow*(v1.getX() - v2.getX(), 2) + Math.*pow*(v1.getY() - v2.getY(), 2));  
 }  
  
 public abstract double getPerimeter();  
 public abstract double getSquare();  
 public abstract QuadrilateralType getType();  
  
 @Override  
 public String toString() {  
 return "Shape{" + "vertices=" + vertices + "}";  
 }  
}

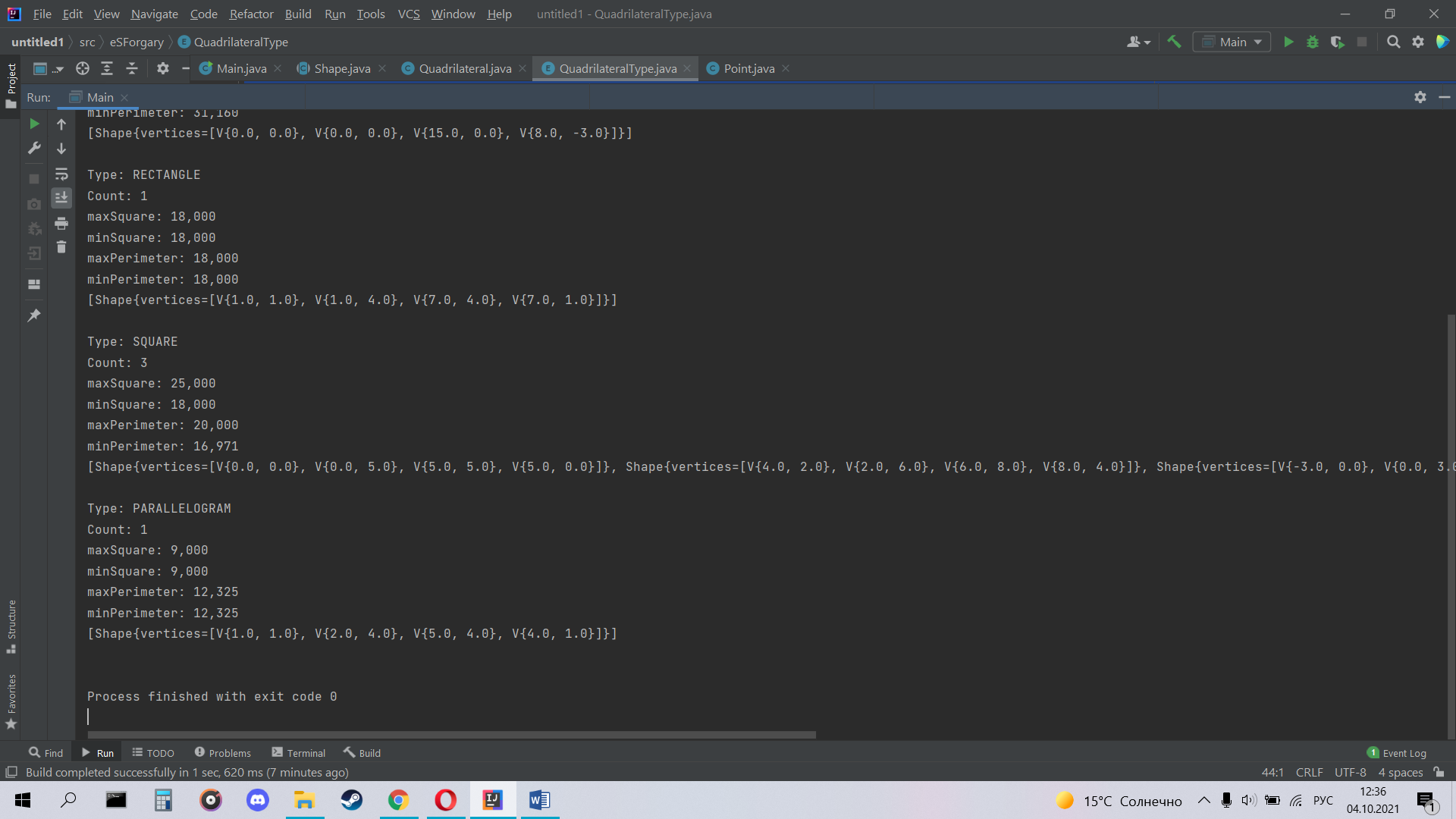
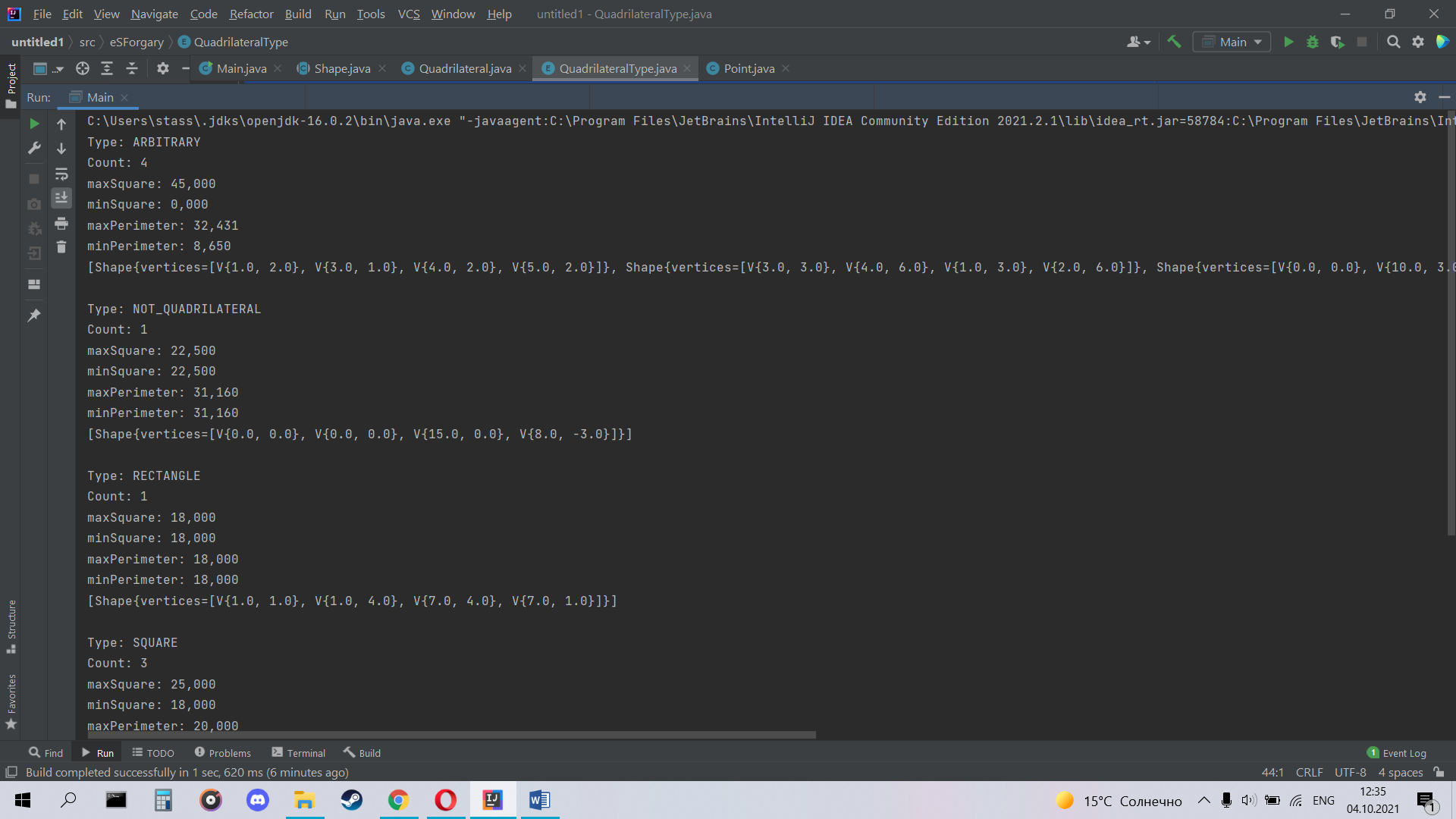
@QuadrilateralType.java

package eSForgary;  
public enum QuadrilateralType {  
 *SQUARE*, *RECTANGLE*, *RHOMBUS*, *ARBITRARY*, *PARALLELOGRAM*, *NOT\_QUADRILATERAL*}

@Quadrilateral.java

package eSForgary;  
public class Quadrilateral extends Shape {  
  
 private double side1;  
 private double side2;  
 private double side3;  
 private double side4;  
 private double diag1;  
 private double diag2;  
  
 public Quadrilateral(Point point1, Point point2, Point point3, Point point4) {  
 super(point1, point2, point3, point4);  
  
 this.side1 = distance(point1, point2);  
 this.side2 = distance(point2, point3);  
 this.side3 = distance(point3, point4);  
 this.side4 = distance(point4, point1);  
 this.diag1 = distance(point1, point3);  
 this.diag2 = distance(point2, point4);  
 }  
  
 @Override  
 public double getPerimeter() {  
 return side1 + side2 + side3 + side4;  
 }  
  
 @Override  
 public double getSquare() {  
 double square = 0;  
 for (int i = 0; i < vertices.size() - 1; i++) {  
  
 Point p1 = vertices.get(i);  
 Point p2 = vertices.get(i + 1);  
 square += ((p1.getX() - p2.getX()) \* (p1.getY() + p2.getY())) / 2;  
 }  
  
 Point p1 = vertices.get(vertices.size() - 1);  
 Point p2 = vertices.get(0);  
 square += ((p1.getX() - p2.getX()) \* (p1.getY() + p2.getY())) / 2;  
  
 return Math.*abs*(square);  
 }  
  
 @Override  
 public QuadrilateralType getType() {  
  
 if (hasIdenticalVertexes())  
 return QuadrilateralType.*NOT\_QUADRILATERAL*;  
  
 if (side1 == side2 && side1 == side3 && side1 == side4 && diag1 == diag2)  
 return QuadrilateralType.*SQUARE*;  
  
 if (side1 == side3 && side2 == side4 && diag1 == diag2)  
 return QuadrilateralType.*RECTANGLE*;  
  
 if (side1 == side2 && side1 == side3 && side1 == side4)  
 return QuadrilateralType.*RHOMBUS*;  
  
 if (side1 == side3 && side2 == side4)  
 return QuadrilateralType.*PARALLELOGRAM*;  
  
 return QuadrilateralType.*ARBITRARY*;  
 }  
  
 private boolean hasIdenticalVertexes() {  
 for (int i = 0; i < vertices.size(); i++)  
 for (int j = 0; j < vertices.size(); j++) {  
 if (i == j) continue;  
 if (vertices.get(i).equals(vertices.get(j)))  
 return true;  
 }  
 return false;  
 }  
}

результат:



3) У тексті слово "країна " замінити словом "Україна".

package eSForgary;

public class Main {  
 public static void main(String[] args) {  
 String word = "Країна", newWord = "Україна";  
 StringBuilder text = new StringBuilder("Країна — це територія з визначеними кордонами й населенням, що являє собою єдине ціле з погляду історії, культури, \nнації та в політико-географічному відношенні може бути незалежною або залежною. Країна не завжди є державою, наприклад \nУкраїна в 1900 р. була країною українців, але поділеною між Російською та Австро-Угорською імперіями.");  
 System.*out*.print("\n 1) " + text);  
  
 int i = 0;  
 while ( (i = text.indexOf(word, i)) >= 0) {  
 text.delete(i, i + word.length());  
 text.insert(i, newWord);  
 i += newWord.length();  
 }  
 System.*out*.print("\n\n 2) " + text);  
 }  
}

результат:

