Донецкий Национальный Технический Университет

Лабораторная работа № 2

«Методы функционального тестирования черным ящиком»

Выполнил:

ст. группы ИПЗ -13

Лысенко А. С.

Проверила:

Курило Е. В.

Красноармейск 2016

Задача:

В трехмерном пространстве задано множество точек. Пересекутся ли две прямые, одна из которых проходит через две ближайшие к началу координат из заданного в пространстве множества точек, вторая - через две максимально удаленные.

1. Анализ предложенной задачи. Математическую модель и метод решения задачи.

1) Задать множество точек

Мин. кол-во точек 4, макс. кол-во точек – любое.

2) Проверка ближайших и максимально удаленных точек

Проверка осуществляется сравниванием длин отрезков от точки начало координат М1(0,0) до точки указанной в пространстве по формуле , выбираем 2-е max и 2-е min удаленные точки

3) Построение векторов из пар ближайших и пар максимально удаленных точек

Вектора строим по формуле

4) Проверка на пересечение прямых из пар ближайших и пар максимально удаленных точек

Прямые (d­1,d2) пересекаются в том случае если они лежат в одной плоскости и при этом их векторы неколлинеарные. Для того чтобы определить находятся ли прямые в одной плоскости, нужно взять координаты точек (x, y, z), (x1, y1, z1), прямых (M1, M2) и найти вектор () и вектора (p1,p2) показывающие направление прямых. В сумме вектора прямых должны дать ноль, что будет означать их нахождение в одной плоскости:  (p1\*p2\*) = 0. Теперь проверим вектора прямых (p1,p2) на коллинеарность, нам нужно неколлинеарные, для этого необходимо чтобы их соответствующие координаты были непропроциональные

= =

Алгоритм решения задачи







Спецификации программных функций

Процедуры:

1) Ввод координат точек

2) Удаление точек

3) Вычисление длин векторов прямых

4) Условие пересечения прямых

5) Изображение прямых

1) Вводим координаты точек x, y, z пока переменная k != -1 после каждой координаты (x, y, z) нажимаем клавишу enter, диапазон значений от [-250, 250].

2) Вводим индекс удаляемой точки(x, y, z), индексы точек будут показаны в консоли, индекс не должен превышать макс кол-во точек.

5) Для изображения прямых, вводим координаты min точек(x, y) 1-й, 2-ой и координаты max точек(x, y) 1-й, 2-ой, диапазон значений от [-250, 250].

Анализ методов тестирования черным ящиком

Приёмы тестирования черным ящиком:

1. Эквивалентное разбиение

2. Анализ граничных условий

3. Анализ причинно-следственных связей

4. Предположение об ошибке

1. Ввод координат точек

1) Метод эквивалентного разбиения:

Процедура ввода координат точек, существует 1 правильный класс – значения лежат в диапазоне от [-250, 250] и 2 не правильных, если a <= [-250, 250] <= b, неправильный класс a, если значение меньше заданного диапазона и класс b, если значение больше заданного диапазона.

2) Анализ граничных значений:

Проверяем -250, 250, -250.000001, 250.000001

3) Анализ причинно-следственных связей:

Причина – входные данные должные находиться в диапазоне от [-250, 250]

Следствие - выходные данные должные находиться в диапазоне от [-250, 250]

|  |  |
| --- | --- |
| Причины | Следствие |
| значение буква | не подходит |
| значение цифра < указанного диапазона | не подходит |
| значение цифра > указанного диапазона | не подходит |
| значение цифра в указанном диапазоне | подходит |

4) Предположение об ошибке:

Ошибки во входных данных

2. Удаление точек

1) Метод эквивалентного разбиения:

Процедура удаления точек, существует 1 правильный класс – значения лежа в диапазоне от [0, n), где n макс кол-во точек, и 2 не правильных, если a <= (0, n) <= b, неправильный класс a, если значение меньше заданного диапазона и класс b, если значение больше заданного диапазона.

2) Анализ граничных значений:

Проверяем -1 , n + 1;

3) Анализ причинно-следственных связей:

Причина – входные данные должные находиться в диапазоне от [0, n)

Следствие - выходные данные должные находиться в диапазоне от [0, n)

|  |  |
| --- | --- |
| Причины | Следствие |
| значение буква | не подходит |
| значение цифра < указанного диапазона | не подходит |
| значение цифра > указанного диапазона | не подходит |
| значение цифра в указанном диапазоне | подходит |

4) Предположение об ошибке:

Ошибки во входных данных

3. Изображение прямых

Процедура ввода координат точек, существует 1 правильный класс – значения лежат в диапазоне от [-250, 250] и 2 не правильных, если a <= [-250, 250] <= b, неправильный класс a, если значение меньше заданного диапазона и класс b, если значение больше заданного диапазона.

3) Анализ причинно-следственных связей:

Причина – входные данные должные находиться в диапазоне от [-250, 250]

Следствие - выходные данные должные находиться в диапазоне от [-250, 250]

|  |  |
| --- | --- |
| Причины | Следствие |
| значение буква | не подходит |
| значение цифра < указанного диапазона | не подходит |
| значение цифра > указанного диапазона | не подходит |
| значение цифра в указанном диапазоне | подходит |

4) Предположение об ошибке:

Ошибки во входных данных

Тест-кейсы для тестирование программы с применением методов черного ящика

1. Ввод координат точек

1) Метод эквивалентного разбиения:

Процедура ввода координат точек, существует 1 правильный класс – значения лежат в диапазоне от [-250, 250] и 2 не правильных, если a <= [-250, 250] <= b, неправильный класс a, если значение меньше заданного диапазона и класс b, если значение больше заданного диапазона.

2) Анализ граничных значений:

Проверяем -250, 250, -250.000001, 250.000001

2. Удаление точек

3) Анализ причинно-следственных связей:

Причина – входные данные должные находиться в диапазоне от [0, n]

Следствие - выходные данные должные находиться в диапазоне от [0, n]

|  |  |
| --- | --- |
| Причины | Следствие |
| значение буква | не подходит |
| значение цифра < указанного диапазона | не подходит |
| значение цифра > указанного диапазона | не подходит |
| значение цифра в указанном диапазоне | подходит |

3. Изображение прямых

4) Предположение об ошибке:

Ошибки во входных данных

Тест-кейсы:

1. id 1 (для ввода координат точек)

Idea: значение входного параметра координаты точки должно находится в диапазоне [-250, 250].

PreCondi1tions: значение должно быть числом а не буквой

Test Case Description: ввод значение, проверка его число или буква, при условии что это число, ввести ограничения диапазона [-250, 250], при условии что буква повторно запросить значение.

Expected Result: после выполнение шагов мы увидим какие из значений являются подходящими, а какие нет.

PostCondition: повторный вызов метода.

2. id 2 (для ввода координат точек)

Idea: проверка граничных значений диапазона [-250, 250], значения следующие -250, 250, -250.000001, 250.000001,

PreConditions: значение должно быть числом а не буквой

Test Case Description: ввод значение, проверка его число или буква, при условии что это число, ввести значения -250, 250, -250.000001, 250.000001 при условии что буква повторно запросить значение.

Expected Result: после выполнение шагов мы увидим правильно ли функционирует процедура

PostCondition: повторный вызов метода.

3. id 3 (для удаления точки)

Idea: значение входного параметра номер точки должно находится в диапазоне [0, n].

PreCondiltions: значение должно быть числом а не буквой

Test Case Description: ввод значение, проверка его число или буква, при условии что это число, ввести ограничения диапазона [0, n], при условии что буква повторно запросить значение.

Expected Result: после выполнение шагов мы увидим какие из значений являются подходящими, а какие нет.

PostCondition: повторный вызов метода.

Программная реализация

Logic.java

**package** Source;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.geom.Arc2D;  
**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** Logic {  
 **int k**;  
 String **k1**;  
 **double n7**;  
 BufferedReader **reader** = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
  
 **public** Logic( ) **throws** IOException {  
 **this**.enterDots();  
 }  
  
 **public void** enterDots() **throws** IOException {  
 */\* ArrayList<Point> point = new ArrayList<>();  
  
 do {  
 point.add(new Point());  
 show("Введите -1, для завершения: ");  
 k1 = reader.readLine();  
 k = Integer.valueOf(k1);  
 } while (k != -1);  
 for (i = 0; i < point.size(); i++) {  
  
 }\*/* **new** Point();  
 *//show("Кол-во точек: " + point.size());* }  
  
 **public void** show(String text) {  
 System.***out***.print(text);  
 }  
  
 **public void** showLn(String text) {  
 System.***out***.println(text);  
 }  
  
 **class** Point {  
 **int count** = 0;  
 **int i** = 0;  
 **double x** = 0;  
 **double y** = 0;  
 **double z** = 0;  
 **double k** = 0;  
 **double n7** = 0;  
 **double n8** = 0;  
 **double n9** = 0;  
 **double n10** = 0;  
 **double n11** = 0;  
 **double n12** = 0;  
 **double n13** = 0;  
 **double n14** = 0;  
 **double n15** = 0;  
 **double n16** = 0;  
 **double n17** = 0;  
 **double n18** = 0;  
 **double d1** = 0;  
 **double d2** = 0;  
 **double d3** = 0;  
 **double d4** = 0;  
 **double d5** = 0;  
 **double d6** = 0;  
 **double d7** = 0;  
 **double d8** = 0;  
 **double d9** = 0;  
 **boolean f**;  
 **int p** = 3;  
 **double d10** = 0;  
 String **x1** = **"251"**;  
 String **y1** = **"251"**;  
 String **z1** = **"251"**;  
 String **k1** = **"251"**;  
 ArrayList<Double> **masX** = **new** ArrayList<>();  
 ArrayList<Double> **masY** = **new** ArrayList<>();  
 ArrayList<Double> **masZ** = **new** ArrayList<>();  
 ArrayList<Double> **d** = **new** ArrayList<>();  
  
 **public** Point() **throws** IOException {  
 **this**.enterPoint();  
 *//this.showPoint();  
 //this.deletePoint();  
 // this.checkPointMax();  
 // this.showSqrt();* }  
  
 **public void** enterPoint() **throws** IOException {  
  
 **do** {  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите координату точки x["** + **i** + **"]: "**);  
 **x1** = **reader**.readLine();  
 tryParseIntX(**x1**);  
 }  
 **x** = Double.*valueOf*(**x1**);  
 **masX**.add(**x**);  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите координату точки y["** + **i** + **"]: "**);  
 **y1** = **reader**.readLine();  
 tryParseIntY(**y1**);  
 }  
 **y** = Double.*valueOf*(**y1**);  
 **masY**.add(**y**);  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите координату точки z["** + **i** + **"]: "**);  
 **z1** = **reader**.readLine();  
 tryParseIntZ(**z1**);  
 }  
 **z** = Double.*valueOf*(**z1**);  
 **masZ**.add(**z**);  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите -1 для завершения: "**);  
 **k1** = **reader**.readLine();  
 tryParseIntK(**k1**);  
 }  
 **k** = Double.*valueOf*(**k1**);  
 **d**.add(**k**);  
 **i**++;  
 **count**++;  
 **f** = **false**;  
 } **while** (**k** != -1);  
 **double** n3 = **masX**.get(0);  
 **double** n4 = **masY**.get(0);  
 **double** n5 = **masZ**.get(0);  
 **double** n6 = 0;  
  
 showPoint();  
 **for** (**i** = 0; **i** < **masX**.size(); **i**++) {  
 **if** ((Math.*sqrt*(Math.*pow*(n3, 2) + Math.*pow*(n4, 2) + Math.*pow*(n5, 2))) >  
 (Math.*sqrt*(Math.*pow*(**masX**.get(**i**), 2) + Math.*pow*(**masY**.get(**i**), 2) + Math.*pow*(**masZ**.get(**i**), 2)))) {  
 n3 = **masX**.get(**i**);  
 n4 = **masY**.get(**i**);  
 n5 = **masZ**.get(**i**);  
 *// n6 = Math.sqrt(Math.pow(masX.get(i), 2) + Math.pow(masY.get(i), 2) + Math.pow(masZ.get(i), 2));  
 // cохранить точки* **n7** = **masX**.get(**i**);  
 **n8** = **masY**.get(**i**);  
 **n9** = **masZ**.get(**i**);  
 } **else** {  
 *// n6 = Math.sqrt(Math.pow(n3, 2) + Math.pow(n4, 2) + Math.pow(n5, 2));* **n7** = n3;  
 **n8** = n4;  
 **n9** = n5;  
 }  
  
 }  
 showLn(**""**);  
 showLn(**"[min] x: "** + n3 + **" y: "** + n4 + **" z: "** + n5);  
 *//showLn("sqrt: "+ n6);* **for** (**i** = 0; **i** < **masX**.size(); **i**++) {  
 **if** ((Math.*sqrt*(Math.*pow*(n3, 2) + Math.*pow*(n4, 2) + Math.*pow*(n5, 2))) <  
 (Math.*sqrt*(Math.*pow*(**masX**.get(**i**), 2) + Math.*pow*(**masY**.get(**i**), 2) + Math.*pow*(**masZ**.get(**i**), 2)))) {  
 n3 = **masX**.get(**i**);  
 n4 = **masY**.get(**i**);  
 n5 = **masZ**.get(**i**);  
 *// n6 = Math.sqrt(Math.pow(masX.get(i), 2) + Math.pow(masY.get(i), 2) + Math.pow(masZ.get(i), 2));  
 //сохранить точки* **n10** = **masX**.get(**i**);  
 **n11** = **masY**.get(**i**);  
 **n12** = **masZ**.get(**i**);  
 } **else** {  
 *// n6 = Math.sqrt(Math.pow(n3, 2) + Math.pow(n4, 2) + Math.pow(n5, 2));* **n10** = n3;  
 **n11** = n4;  
 **n12** = n5;  
  
 }  
  
 }  
 showLn(**""**);  
 showLn(**"[max] x: "** + n3 + **" y: "** + n4 + **" z: "** + n5);  
 *//showLn("sqrt: "+ n6);* showLn(**""**);  
 deletePoint();  
 showPoint();  
 deletePoint();  
 showLn(**""**);  
 showPoint();  
 n3 = **masX**.get(0);  
 n4 = **masY**.get(0);  
 n5 = **masZ**.get(0);  
 **for** (**i** = 0; **i** < **masX**.size(); **i**++) {  
 **if** ((Math.*sqrt*(Math.*pow*(n3, 2) + Math.*pow*(n4, 2) + Math.*pow*(n5, 2))) >  
 (Math.*sqrt*(Math.*pow*(**masX**.get(**i**), 2) + Math.*pow*(**masY**.get(**i**), 2) + Math.*pow*(**masZ**.get(**i**), 2)))) {  
 n3 = **masX**.get(**i**);  
 n4 = **masY**.get(**i**);  
 n5 = **masZ**.get(**i**);  
 n6 = Math.*sqrt*(Math.*pow*(**masX**.get(**i**), 2) + Math.*pow*(**masY**.get(**i**), 2) + Math.*pow*(**masZ**.get(**i**), 2));  
 *//сохранить точки* **n13** = **masX**.get(**i**);  
 **n14** = **masY**.get(**i**);  
 **n15** = **masZ**.get(**i**);  
  
 } **else** {  
 n6 = Math.*sqrt*(Math.*pow*(n3, 2) + Math.*pow*(n4, 2) + Math.*pow*(n5, 2));  
 *//cохранить точки* **n13** = n3;  
 **n14** = n4;  
 **n15** = n5;  
 }  
  
 }  
 showLn(**""**);  
 showLn(**"[min] x: "** + n3 + **" y: "** + n4 + **" z: "** + n5);  
 *//showLn("sqrt: "+ n6);* **for** (**i** = 0; **i** < **masX**.size(); **i**++) {  
 **if** ((Math.*sqrt*(Math.*pow*(n3, 2) + Math.*pow*(n4, 2) + Math.*pow*(n5, 2))) <  
 (Math.*sqrt*(Math.*pow*(**masX**.get(**i**), 2) + Math.*pow*(**masY**.get(**i**), 2) + Math.*pow*(**masZ**.get(**i**), 2)))) {  
 n3 = **masX**.get(**i**);  
 n4 = **masY**.get(**i**);  
 n5 = **masZ**.get(**i**);  
 n6 = Math.*sqrt*(Math.*pow*(**masX**.get(**i**), 2) + Math.*pow*(**masY**.get(**i**), 2) + Math.*pow*(**masZ**.get(**i**), 2));  
 **n16** = **masX**.get(**i**);  
 **n17** = **masY**.get(**i**);  
 **n18** = **masZ**.get(**i**);  
 } **else** {  
 n6 = Math.*sqrt*(Math.*pow*(n3, 2) + Math.*pow*(n4, 2) + Math.*pow*(n5, 2));  
 **n16** = n3;  
 **n17** = n4;  
 **n18** = n5;  
 }  
 }  
 showLn(**""**);  
 showLn(**"[max] x: "** + n3 + **" y: "** + n4 + **" z: "** + n5);  
 showLn(**""**);  
 lineLong();  
 }  
  
 **public void** showPoint() {  
 showLn(**""**);  
 **for** (**i** = 0; **i** < **masX**.size(); **i**++) {  
 show(**"x["** + **i** + **"]: "** + **masX**.get(**i**));  
 show(**" y["** + **i** + **"]: "** + **masY**.get(**i**));  
 show(**" z["** + **i** + **"]: "** + **masZ**.get(**i**));  
 showLn(**""**);  
 }  
 }  
  
 **int id** = 0;  
 String **id1**;  
  
 **public void** showSqrt() {  
 **for** (**i** = 0; **i** < **count**; **i**++) {  
 showLn(**"sqrt: "** + Math.*sqrt*(Math.*pow*(**masX**.get(**i**), 2) + Math.*pow*(**masY**.get(**i**), 2) + Math.*pow*(**masZ**.get(**i**), 2)));  
 }  
 }  
  
 **public void** deletePoint() **throws** IOException {  
 **f** = **false**;  
 **id** = 0;  
 **while** (!**f**) {  
 show(**"Введите индекc удаленной точки: "**);  
 **id1** = **reader**.readLine();  
 tryParseIntId(**id1**);  
 }  
 **id** = Integer.*valueOf*(**id1**);  
 **masX**.remove(**id**);  
 **masY**.remove(**id**);  
 **masZ**.remove(**id**);  
 }  
  
 **public void** lineLong() {  
 **d1** = **n7** - **n13**;  
 **d2** = **n8** - **n14**;  
 **d3** = **n9** - **n15**;  
 showLn(**"Длина вектора 2-x min отдаленных точек: "** + **"x: "** + **d1** + **" y: "** + **d2** + **" z: "** + **d3**);  
 **d4** = **n10** - **n16**;  
 **d5** = **n11** - **n17**;  
 **d6** = **n12** - **n18**;  
 showLn(**"Длина вектора 2-x max отдаленных точек: "** + **"x: "** + **d4** + **" y: "** + **d5** + **" z: "** + **d6**);  
 **d7** = **n7** - **n10**;  
 **d8** = **n8** - **n11**;  
 **d9** = **n9** - **n12**;  
 showLn(**"Длина вектора между min и max отдаленными точками: "** + **"x: "** + **d7** + **" y: "** + **d8** + **" z: "** + **d9**);  
  
 **d10** = **d1** \* **d5** \* **d9** + **d2** \* **d6** \* **d7** + **d3** \* **d4** \* **d8** - **d3** \* **d5** \* **d7** - **d1** \* **d6** \* **d8** - **d2** \* **d4** \* **d9**;  
 **if** (**d10** == 0) {  
 showLn(**"Лежат в одной плоскости."** + **d10**);  
 *// условие* **if** (**d1** / **d4** != **d2** / **d5** && **d1** / **d4** != **d3** / **d6** && **d2** / **d5** != **d3** / **d6**) {  
 showLn(**"Прямые пересекаются."**);  
 }  
 } **else** {  
 showLn(**"Не лежат в одной плоскости (определитель): "** + **d10**);  
 showLn(**"Не пересекаются"**);  
 }  
 }  
  
 Double tryParseIntX(String s) **throws** IOException {  
 {  
 **try** {  
 **f** = **true**;  
 **return new** Double(s);  
 } **catch** (NumberFormatException e) {  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите координату точки x["** + **i** + **"]: "**);  
 **x1** = **reader**.readLine();  
 tryParseIntX(**x1**);  
 **x** = Double.*valueOf*(**x1**);  
 **while** (**x** <-250 || **x** > 250)  
 {  
 show(**"Введите координату точки x["** + **i** + **"]: "**);  
 **x1** = **reader**.readLine();  
 tryParseIntX(**x1**);  
 **x** = Double.*valueOf*(**x1**);  
 }  
  
 }  
 **return null**;  
 }  
 }  
 }  
  
 Double tryParseIntY(String s) **throws** IOException {  
 {  
 **try** {  
 **f** = **true**;  
 **return new** Double(s);  
 } **catch** (NumberFormatException e) {  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите координату точки y["** + **i** + **"]: "**);  
 **y1** = **reader**.readLine();  
 tryParseIntY(**y1**);  
 **y** = Double.*valueOf*(**y1**);  
 **while** (**y** <-250 || **y** > 250)  
 {  
 show(**"Введите координату точки y["** + **i** + **"]: "**);  
 **y1** = **reader**.readLine();  
 tryParseIntY(**y1**);  
 **y** = Double.*valueOf*(**x1**);  
 }  
 }  
 **return null**;  
 }  
 }  
 }  
 Double tryParseIntZ(String s) **throws** IOException {  
 {  
 **try** {  
 **f** = **true**;  
 **return new** Double(s);  
 } **catch** (NumberFormatException e) {  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите координату точки z["** + **i** + **"]: "**);  
 **z1** = **reader**.readLine();  
 tryParseIntZ(**z1**);  
 **z** = Double.*valueOf*(**z1**);  
 **while** (**z** <-250 || **z** > 250)  
 {  
 show(**"Введите координату точки z["** + **i** + **"]: "**);  
 **z1** = **reader**.readLine();  
 tryParseIntZ(**z1**);  
 **z** = Double.*valueOf*(**x1**);  
 }  
 }  
 **return null**;  
 }  
 }  
 }  
 Double tryParseIntK(String s) **throws** IOException {  
 {  
 **try** {  
 **f** = **true**;  
 **return new** Double(s);  
 } **catch** (NumberFormatException e) {  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите -1 для завершения: "**);  
 **k1** = **reader**.readLine();  
 tryParseIntK(**k1**);  
 **k** = Double.*valueOf*(**z1**);  
 }  
 **return null**;  
 }  
 }  
 }  
 Integer tryParseIntId(String s) **throws** IOException {  
 {  
 **try** {  
 **f** = **true**;  
 **return new** Integer(s);  
 } **catch** (NumberFormatException e) {  
 **f** = **false**;  
 **while** (!**f**) {  
 show(**"Введите индекc удаленной точки: "**);  
 **id1** = **reader**.readLine();  
 tryParseIntId(**id1**);  
 **id** = Integer.*valueOf*(**id1**);  
 **while** (**id** <0 || **id** > **count** - 1)  
 {  
 show(**"Введите индекc удаленной точки: "**);  
 **id1** = **reader**.readLine();  
 tryParseIntId(**id1**);  
 **id** = Integer.*valueOf*(**id1**);  
 }  
 }  
 **return null**;  
 }  
 }  
 }  
 }  
}

Main.java

**package** Source;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.util.ArrayList;  
  
  
**public class** Main **extends** JPanel  
{  
 *//static int x1x = 50, y1y = 35, x2x = 10, y2y = 40; // координаты точек 1-го отрезка  
 //static int x3x = 70, y3y = 30, x4x = 15, y4y = 40; // координаты точек 2-го отрезка* **static int** *X* = 0, *Y* = 0;  
 **static** String *X1*;  
 **static int** *count* = 0;  
 **static int** *i* = 0;  
 **static double** *x4x* = 0;  
 **static double** *y4y* = 0;  
 **static double** *x1x* = 0;  
 **static double** *y1y* = 0;  
 **static double** *x2x* = 0;  
 **static double** *y2y* = 0;  
  
 **static double** *x3x* = 0;  
 **static double** *y3y* = 0;  
 **static double** *z* = 0;  
 **static double** *k* = 0;  
  
 **static** String *x1*;  
 **static** String *y1*;  
 **static** String *x2*;  
 **static** String *y2*;  
 **static** String *x3*;  
 **static** String *y3*;  
 **static** String *x4*;  
 **static** String *y4*;  
 **static** String *z1*;  
 **static** String *k1*;  
 **static** ArrayList<Double> *masX* = **new** ArrayList<>();  
 **static** ArrayList<Double> *masY* = **new** ArrayList<>();  
  
 **static** BufferedReader *reader* = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
  
 **public static void** show(String text) {  
 System.***out***.print(text);  
 }  
  
 **public static void** showLn(String text) {  
 System.***out***.println(text);  
 }  
  
 **public static void** main(String[] args) **throws** IOException  
 {  
 **new** Logic();  
 *show*(**"Введите координату точки x["** + *i* + **"](min): "**);  
 *x1* = *reader*.readLine();  
 *x1x* = Double.*valueOf*(*x1*);  
 *masX*.add(*x1x*);  
  
 *show*(**"Введите координату точки y["** + *i* + **"](min): "**);  
 *y1* = *reader*.readLine();  
 *y1y* = Double.*valueOf*(*y1*);  
 *masY*.add(*y1y*);  
 *i*++;  
 *show*(**"Введите координату точки x["** + *i* + **"](min): "**);  
 *x2* = *reader*.readLine();  
 *x2x* = Double.*valueOf*(*x2*);  
 *masX*.add(*x2x*);  
  
 *show*(**"Введите координату точки y["** + *i* + **"](min): "**);  
 *y2* = *reader*.readLine();  
 *y2y* = Double.*valueOf*(*y2*);  
 *masY*.add(*y2y*);  
 *i*++;  
 *show*(**"Введите координату точки x["** + *i* + **"](max): "**);  
 *x3* = *reader*.readLine();  
 *x3x* = Double.*valueOf*(*x3*);  
 *masX*.add(*x3x*);  
  
 *show*(**"Введите координату точки y["** + *i* + **"](max): "**);  
 *y3* = *reader*.readLine();  
 *y3y* = Double.*valueOf*(*y3*);  
 *masY*.add(*y3y*);  
 *i*++;  
 *show*(**"Введите координату точки x["** + *i* + **"](max): "**);  
 *x4* = *reader*.readLine();  
 *x4x* = Double.*valueOf*(*x4*);  
 *masX*.add(*x4x*);  
  
 *show*(**"Введите координату точки y["** + *i* + **"](max): "**);  
 *y4* = *reader*.readLine();  
 *y4y* = Double.*valueOf*(*y4*);  
 *masY*.add(*y4y*);  
 *i*++;  
 Main dr = **new** Main();  
 *//построение векторов* **double** v1, v2, v3, v4;  
 **double** xv12, xv13, xv14, xv31, xv32, xv34, yv12, yv13, yv14, yv31, yv32, yv34;  
 *//нахождение координат векторов* xv12 = *x2x* - *x1x*; xv13 = *x3x* - *x1x*; xv14 = *x4x* - *x1x*;  
 yv12 = *y2y* - *y1y*; yv13 = *y3y* - *y1y*; yv14 = *y4y* - *y1y*;  
  
 xv31 = *x1x* - *x3x*; xv32 = *x2x* - *x3x*; xv34 = *x4x* - *x3x*;  
 yv31 = *y1y* - *y3y*; yv32 = *y2y* - *y3y*; yv34 = *y4y* - *y3y*;  
  
  
 *// векторные произведения* v1 = xv34 \* yv31 - yv34 \* xv31;  
 v2 = xv34 \* yv32 - yv34 \* xv32;  
 v3 = xv12 \* yv13 - yv12 \* xv13;  
 v4 = xv12 \* yv14 - yv12 \* xv14;  
  
 **if**((v1 \* v2) < 0 && (v3 \* v4) < 0){  
 **double** A1, B1, C1, A2, B2, C2;  
 A1 = *y2y* - *y1y*;  
 A2 = *y4y* - *y3y*;  
 B1 = *x1x* - *x2x*;  
 B2 = *x3x* - *x4x*;  
 C1 = (*x1x* \* (*y1y* - *y2y*) + *y1y* \* (*x2x*- *x1x*)) \* (-1);  
 C2 = (*x3x* \* (*y3y* - *y4y*) + *y3y* \* (*x4x* - *x3x*)) \* (-1);  
  
  
 **double** D = (**double**) ((A1 \* B2) - (B1 \* A2));  
 **double** Dx = (**double**) ((C1 \* B2) - (B1 \* C2));  
 **double** Dy = (**double**) ((A1 \* C2) - (C1 \* A2));  
  
 **if**(D != 0) *// !=* {  
 *X* = (**int**) (Dx / D);  
 *Y* = (**int**) (Dy / D);  
 *// X1 = "Отрезки пересекаются в точке: " + X + ", " + Y;* dr.repaint();  
 }  
 }  
 *// else X1 = "Отрезки не пересекаются ";* }  
 **public** Main() **throws** IOException  
 {  
 JFrame jf = **new** JFrame();  
 jf.setSize(500, 500);  
 jf.setDefaultCloseOperation(jf.***EXIT\_ON\_CLOSE***);  
 jf.setLocationRelativeTo(**null**);  
  
 jf.getContentPane().add(**this**);  
  
 jf.setVisible(**true**);  
 jf.repaint();  
  
 }  
 **protected void** paintComponent(Graphics g){  
 **super**.paintComponent(g);  
  
 g.drawLine((**int**)*x1x*, (**int**)*y1y*, (**int**)*x2x*, (**int**)*y2y*);  
  
 g.drawLine((**int**)*x3x*, (**int**)*y3y*, (**int**)*x4x*, (**int**)*y4y*);  
  
 g.setColor(Color.***black***);  
 *// g.drawString(X1, 10, 10);* }  
 }

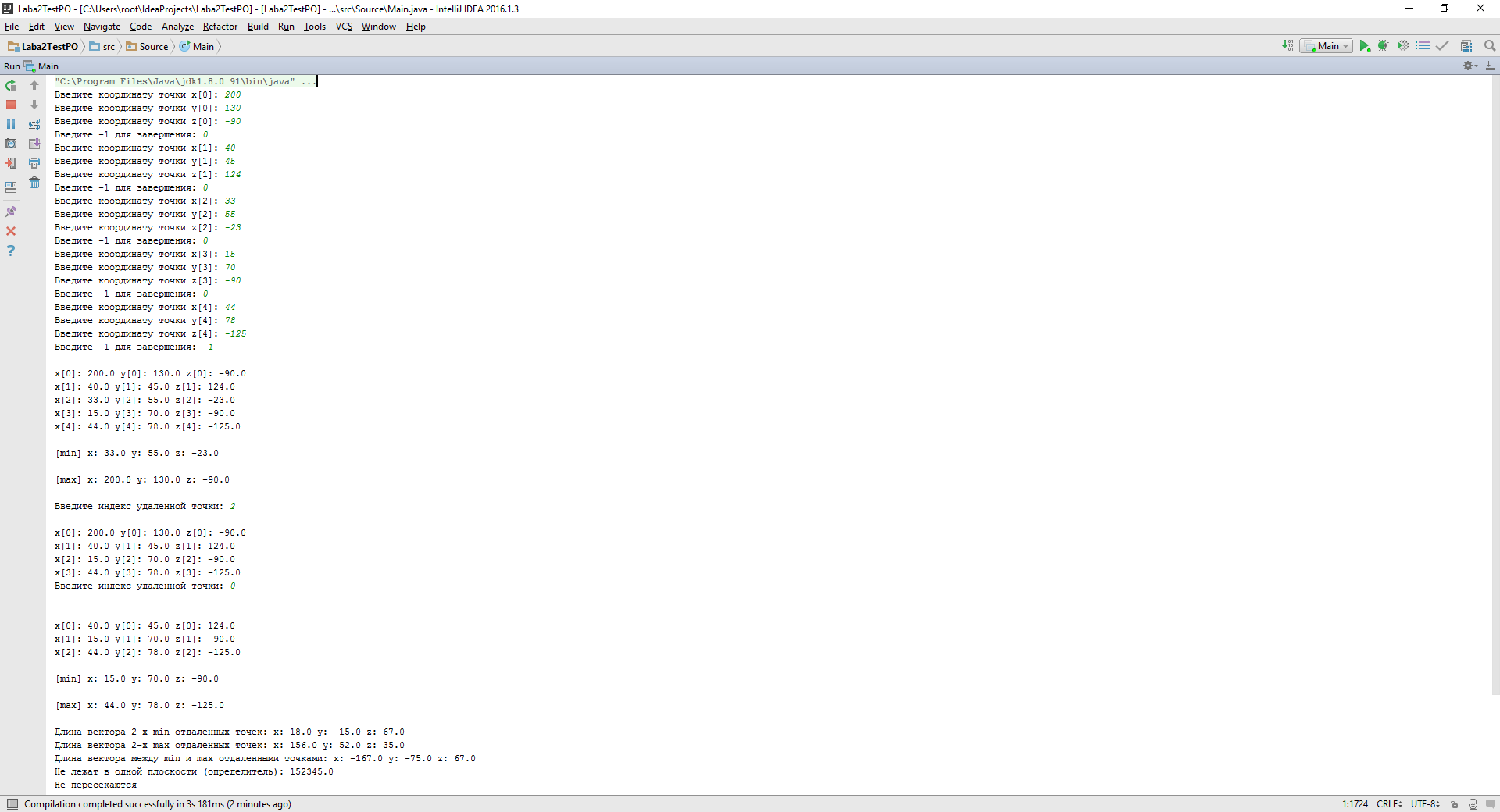


Рис 1.

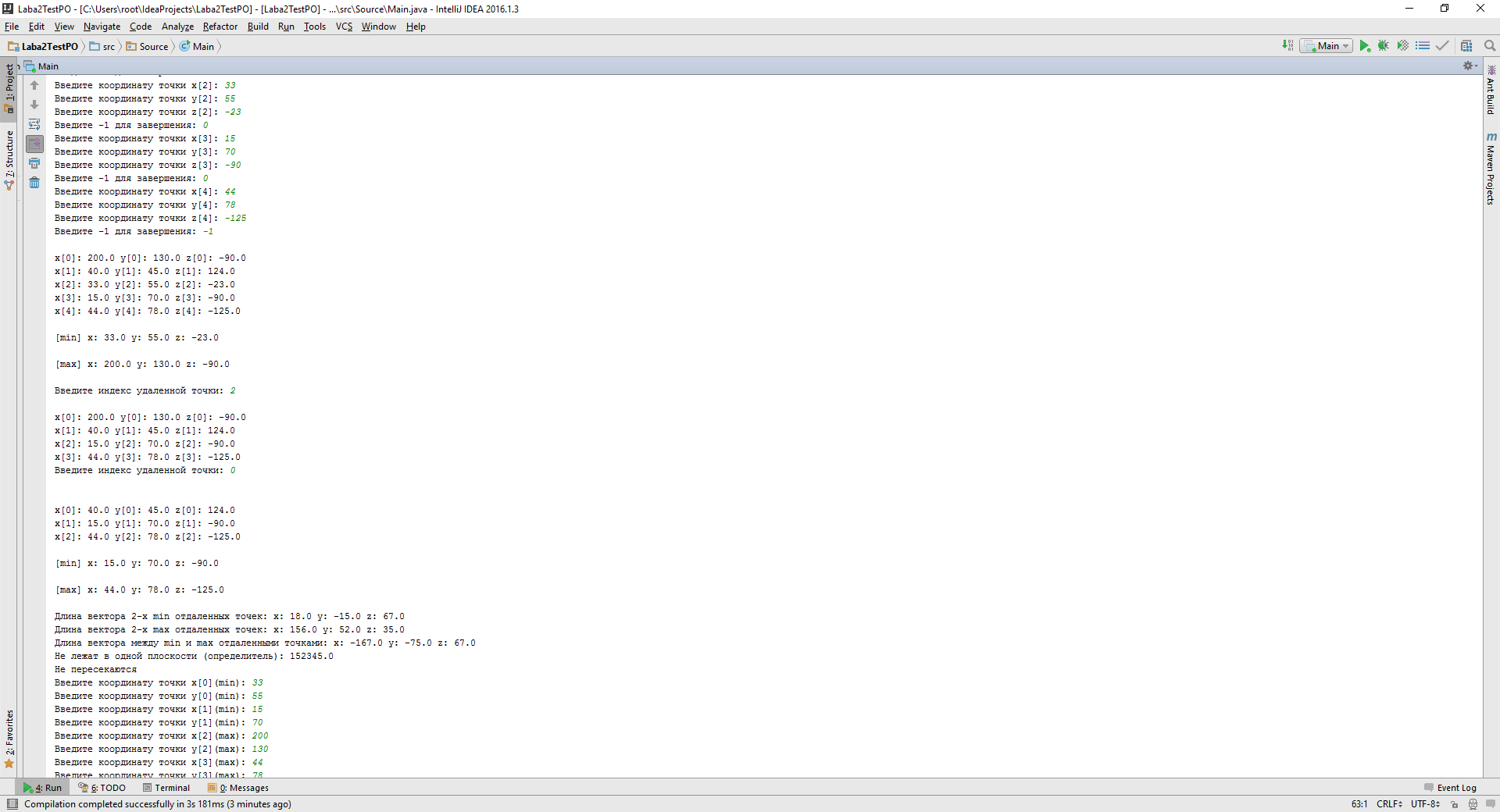


Рис 2.

