```java
package ir.shayandaneshvar;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class Main extends Application {

    public static void main(String[] args) { launch(args); }

    @Override
    public void start(Stage stage) throws Exception {
        BorderPane root = FXMLLoader.load(getClass() Class<capture of ? extends Main>
                .getResource( name: "/main.fxml") URL
                .toURI() URI
                .toURL());
        Scene scene = new Scene(root);
        stage.setTitle("Polygon Triangulation");
        stage.getIcons().add(new Image(getClass().getResource(
                name: "/images/triangle.png").toURI().toURL().toString()));
        stage.setScene(scene);
        stage.show();
    }
}

```

Class Main:

در اینجا صفحه اولیه برنامه از فایل main.fxml خوانده می شود و برنامه بالا می آید، ریشه فرم اصلی برنامه یک borderpane است که با خواندن فایل fxml این ریشه به ما داده می شود تا آن را درون یک scene گذاشته و sceneرا درون stage و بعد برنامه را اجرا کنیم.

Main.fxml:

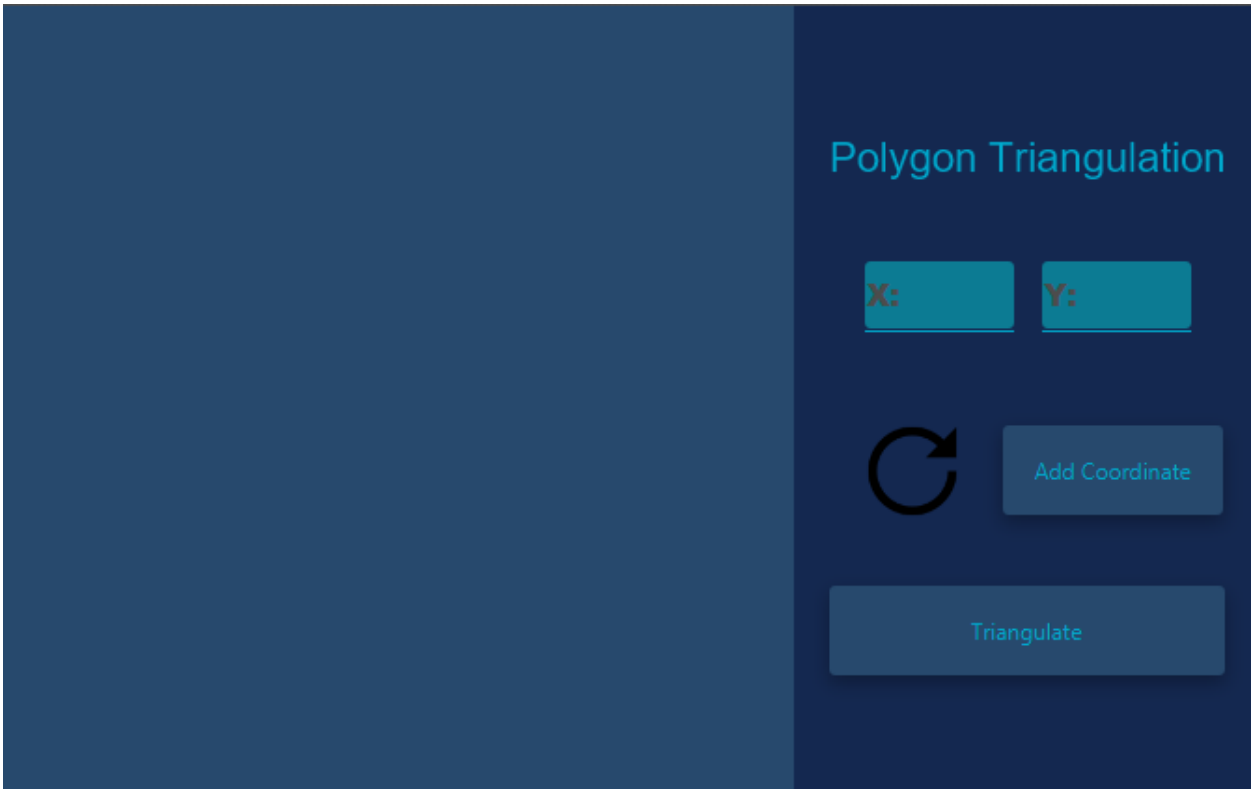
این فایل به کمک Scene Builder طراحی شده که کد آن قابل مشاهده است و فرم نهایی آن نیز در زیر عکس main.fxml قابل مشاهده است.

نمایش نقاط و چند ضلعی د وقطر ها در بخش سمت چپ فرم خواهد بود وسمت راست می توان به صورت ساعت گرد یا پاد ساعت گرد نقاط چند ضلعی را به ترتیب به برنامه وارد کرد و با زدن Triangulate قطر ها کشیده و مینیموم هزینه محاسبه می شوند، به کمک دکمه ریست نیز می توان برنامه را به حالت اولیه خود بازگرداند و از ابتدا رسم کرد.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXTextField?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="421.0" prefWidth="673.0" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="ir.shayandaneshvar.Controller">
   <right>
      <AnchorPane prefHeight="421.0" prefWidth="249.0" style="-fx-background-color: #142850;" BorderPane.alignment="CENTER">
         <children>
            <Text fill="#00a8cc" layoutX="19.0" layoutY="89.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Polygon Triangulation" textAlignment="CENTER">
               <font>
                  <Font name="Arial" size="22.0" />
               </font>
            </Text>
            <JFXTextField focusColor="#27496d" layoutX="38.0" layoutY="137.0" onKeyPressed="#handleKeyPressed" prefHeight="31.0" prefWidth="80.0" promptText="X:" style="-fx-background-color: #0c7b93;" unFocusColor="#00a8cc" fx:id="xField">
            <font>
               <Font name="Arial Black" size="16.0" />
            </font></JFXTextField>
            <JFXTextField fx:id="yField" focusColor="#27496d" layoutX="133.0" layoutY="137.0" onKeyPressed="#handleKeyPressed" prefHeight="31.0" prefWidth="80.0" promptText="Y:" style="-fx-background-color: #0c7b93;" unFocusColor="#00a8cc">
            <font>
               <Font name="Arial Black" size="16.0" />
            </font></JFXTextField>
            <JFXButton buttonType="RAISED" layoutX="112.0" layoutY="225.0" onMouseClicked="#addCoordinate" prefHeight="48.0" prefWidth="118.0" ripplerFill="#0c7b93" style="-fx-background-color: #27496d;" text="Add Coordinate" textFill="#00a8cc" />
            <JFXButton buttonType="RAISED" layoutX="19.0" layoutY="311.0" onMouseClicked="#triangulate" prefHeight="48.0" prefWidth="212.0" ripplerFill="#0c7b93" style="-fx-background-color: #27496d;" text="Triangulate" textFill="#00a8cc" />
            <ImageView fitHeight="57.0" fitWidth="61.0" layoutX="35.0" layoutY="221.0" onMouseClicked="#reset" pickOnBounds="true" preserveRatio="true">
               <image>
                  <Image url="@images/reset.png" />
               </image>
            </ImageView>
         </children>
      </AnchorPane>
   </right>
   <center>
      <AnchorPane fx:id="drawPane" prefHeight="200.0" prefWidth="200.0" style="-fx-background-color: #27496d;" BorderPane.alignment="CENTER" />
   </center>
</BorderPane>
```



Class Controller:

این کلاس هسته اصلی برنامه و کنترلر فرمی است که طراحی شده است. X و YField همان تکستباکس هایی اند که مختصات در آن ها وارد می شوند، drawPane بخش سمت چپ فرم است که چندضلعی و قطر هایش در آن رسم می شوند.

```java
package ir.shayandaneshvar;

import ...

public class Controller implements Initializable {
    private List<Pair<Double, Double>> coordinates;
    @FXML private JFXTextField yField;
    @FXML private JFXTextField xField;
    @FXML private AnchorPane drawPane;
    @FXML void addCoordinate() {
        var x = Double.parseDouble(xField.getText());
        var y = Double.parseDouble(yField.getText());
        var pair = new Pair<>(x, y);
        coordinates.add(pair);
        xField.setText("");
        yField.setText("");
        drawPolygon(coordinates);
    }

    @FXML void handleKeyPressed(KeyEvent event) {
        if (event.getCode() == KeyCode.ENTER) {
            xField.requestFocus();
            addCoordinate();
        }
    }
    @FXML void reset() {
        coordinates.clear();
        drawPane.getChildren().clear();
    }
```

coordinates نیز مختصات تمام نقاط چند ضلعی که در برنامه وارد می شوند را در خود نگاه می دارد، متد AddCoordinate نیز که به دکمه متناظر آن bind شده اطلاعات را استخراج می کند داخل یک دوتایی مرتب (Pair) می ریزد آن را به لیست نقاط اضافه می کند و در نهایت متد draw Polygon را فراخوانی میکند که نقاط داده شده را می گیرد و چند ظلعی رسم میکند.

متد handleKeyPressed برای user friendly تر شدن برنامه است تا اگر هنگام وارد کردن مختصات دکمه Enter هم فشار دهید مشابه کلیک کردن روی دکمه add Coordinate عمل کند و فرایند ورودی دادن را تسریع کند.

متد ریست که به تصویر ریست بایند شده است، لیست نقاط را خالی می کند و همچنین طرف عناصر موجود در سمت چپ فرم را پاک می کند.

```java
@FXML void triangulate() {
    List<Pair<Pair<Double, Double>, Pair<Double, Double>>> result =
            new ArrayList<>();
    double cost = getLines(result);
    drawLines(result);
    Alert alert = new Alert(Alert.AlertType.INFORMATION,
            s: "The Minimum Cost is: " + cost, ButtonType.OK);
    alert.setTitle("Done!");
    alert.setHeaderText("Triangulated Successfully!");
    alert.show();
}
private double distance(Pair<Double, Double> p1,
                        Pair<Double, Double> p2) {
    return Math.sqrt(Math.pow(p1.getKey() - p2.getKey(), 2) +
            Math.pow(p1.getValue() - p2.getValue(), 2));
}
private void drawLines(List<Pair<Pair<Double, Double>, Pair<Double, Double>>> result) {
    result.forEach(z -> {
        var from = z.getKey();
        var to = z.getValue();
        Line line = new Line( v: from.getKey() * 10, v1: 420 - from.getValue() * 10,
                v2: to.getKey() * 10, v3: 420 - to.getValue() * 10);
        line.setStrokeLineCap(StrokeLineCap.ROUND);
        line.setStroke(Color.YELLOWGREEN);
        line.setStrokeWidth(3);
        drawPane.getChildren().add(line);
    });
}
```

متد triangulate که به دکمه متناظر آن بایند شده لیستی را که معادل نقطه شروع و نقطه پایان است را به متد getLines می دهد تا مجموع قطر هایی که باید رسم شوند را داخل این لیست بریزد و مقدار هزینه را نیز به صورت یک double باز گرداند و بعد از آن پنجره ای باز شود و مقدار هزینه مینیموم را نمایش دهد.

متد distance متد ساده ای که فاصله هندسی دو نقطه را محاسبه می کند و بر می گرداند.

متد drawLines : متدی که نتیجه خطوط یعنی قطر هارا به صورت ورودی می گیرد و آن ها را به رنگ yellowgreen رسم میکند. برای بزرگ تر شدن اشکال همه مختصات ها هنگام رسم در 10 ضرب شده اند، چون ساختار گرافیکی javafx محور عرض های آن برعکس است، ارتفاع بخش سمت چپ را از هر مختصات کم میکنیم (که برابر 420 است) تا به صورت مختصات کارتزین معمولی در آیند و رسم شوند.

متد drawPolygon : همان متدی است که نقاط ورودی را میگیرد و به ترتیب بین هر دو نقطه خطی آبی می کشد ، در نهایت بین آخرین نقطه و اولین نقطه خط دیگری می کشد تا چند ضلعی کامل رسم شود.

متد initialize : متدی برای initialize کردن اجزای غیر گرافیکی، که مشابه سازنده عمل می کند، در اینجا فرقی نمیکرد که از این متد استفاده کنیم یا سازنده.در بدنه این متد لیست نقاط را new میکنیم.

```java
    private void drawPolygon(
            List<Pair<Double, Double>> coords) {
        drawPane.getChildren().clear();
        Pair<Double, Double> current, prev = coords.get(0);
        for (int i = 1; i < coords.size(); i++) {
            current = coords.get(i);
            var line = new Line( v: prev.getKey() * 10,
                    v1: 420 - prev.getValue() * 10,
                    v2: current.getKey() * 10, v3: 420 - current.getValue() * 10);
            prev = current;
            line.setStrokeWidth(4);
            line.setStrokeLineCap(StrokeLineCap.ROUND);
            line.setStroke(Color.CYAN);
            drawPane.getChildren().add(line);
        }
        Line line = new Line( v: coords.get(0).getKey() * 10,
                v1: 420 - coords.get(0).getValue() * 10,
                v2: prev.getKey() * 10, v3: 420 - prev.getValue() * 10);
        line.setStrokeLineCap(StrokeLineCap.ROUND);
        line.setStrokeWidth(4);
        line.setStroke(Color.CYAN);
        drawPane.getChildren().add(line);
    }

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        coordinates = new ArrayList<>();
    }
```

متد getLines :

این متد که بخش اصلی الگوریتم است، لیستی را برای ریختن جواب در آن دریافت می کند و مقدار هزینه مینیموم را برمیگرداند.

توضیح الگوریتم: مشابه توضیح داده شده و با توجه به اینکه زیر مسئله تکراری زیاد داریم از روش پویا استفاده می کنیم، به این صورت که ابتدا چند ظلعی را از 0 ضلعی شروع کرده و تا n می رویم و در هر مرحله حالت مینیموم را پیدا میکنیم، برای اینکه به ازای هر m ضلعی مقدار کمترین هزینه را بدست آوریم Kه های مختلف را تست می کنیم به این صورت که یک اشاره گر i و یک اشاره گر j میگیریم که بیانگر راس شروع و اتمام چند ضلعی است و راس k راسی است که میخواهیم با استفاده از آن مسئله را دوقسمت کنیم به طوری که از i تا k یک چند ضلعی و از k تا j یک چند ضلعی دیگر داریم حال برای این دوقسمت کردن نیاز به رسم قطر داریم که این قطر ها از i به k و از k به j خواهند بود، تنها باید چک کنیم که اگر یکی از این قطر ها روی چند ضلعی بود آن را در نظر میگریم که این اتفاق زمانی میفتد که اختلاف شماره دو راس کمتر از 2 باشد، زیرا اگر 1 باشد خطی که رسم میکنیم دقیقا روی چند ضلعی میفتد و اگر صفر باشد اصلا قطری نمی توان رسم کرد. به همین دلیل با فرض اینکه هموار مقدار j بزرگتر از i است و k نیز همیشه بین این دو است پس برای محاسبه هزینه ماتریس دوبعدی که بخش پایین اش همه صفر اند کفایت میکند، همچنین چون مقدار k بین I و J است و هیچ گاه مساوی نیست پس قطر اول ماتریس نیز همواره صفر است، همچنین چون چند ضلعی نداریم که نقطه شماره راس شروع و پایان آن یکی باشد (یعنی یک راس داشته باشد) پس در خانه هایی از این ماتریس که j = I است نیز مقدار همواره برابر صفر است، مابقی خانه ها را فعلا بی نهایت قرار می دهیم و هر گاه هزینه مثلث بندی کمتری

بدست آمد جایگزین می کنیم. طرز محاسبه نیز برابر هزینه مثلث بندی چند ضلعی از راس i تا k است به علاوه هزینه مثلث بندی چند ضلعی از k تا j که آن نیز به علاوه هزینه جدا کردن این دو چند ضلعی می شود که در متد getCost محاسبه میشود.

```
116     double getLines(List<Pair<Pair<Double, Double>, Pair<Double, Double>>>
117                             result) {
118         double[][] lengths = new double[coordinates.size()][coordinates.size()];
119         int[][] answer = new int[coordinates.size()][coordinates.size()];
120         for (int i = 0; i < coordinates.size(); i++) {
121             for (int j = 0; j < coordinates.size(); j++) {
122                 answer[i][j] = 0;
123             }
124         }
125         for (int diagonal = 0; diagonal < coordinates.size(); diagonal++) {
126             for (int i = 0, j = diagonal; j < coordinates.size(); i++, j++) {
127                 if (j - 2 < i) {
128                     lengths[i][j] = 0.0;
129                 } else {
130                     lengths[i][j] = Double.MAX_VALUE;
131                     for (int k = i + 1; k < j; k++) {
132                         double val = lengths[i][k] + lengths[k][j] +
133                                 getCost(coordinates, i, j, k);
134                         if (lengths[i][j] > val) {
135                             lengths[i][j] = val;
136                             answer[i][j] = k;
137                         }
138                     }
139                 }
140             }
141         }
142         getAnswer(result, coordinates, answer, i: 0, j: coordinates.size() - 1);
143         return lengths[0][coordinates.size() - 1] / 2;
144     }
```

همچنین در حین محاسبه ، هرگاه مقدار خانه ای کمتر شد، مقدار k آن درماتریس answer که ماتریسی است که مقدار k بهینه را در آن ذخیره میکنیم تا بعدا قطر های بهینه را از آن استخراج کنیم، در ابتدا همه خانه های این ماتریس را نیز صفر میکنیم، چرا که صفر به این معنی است که k بهینه وجود نداشته است و دلیل آن اینست که مقدار k بین i و j است یعنی از 1 تا n-1.

در انتها چون در هر بار جدا سازی چند ضلعی هزینه قطر های مرزی یک بار در مرحله قبل حساب شده و بار دیگر درمرحله فعلی محسابه می شود پس دوبار محاسبه می شود و هزینه کمینه برای مثلث بندی بین رئوس 0 تا n-1 دوبرابر حساب می شود، به همین دلیل در انتها آن را در 2 تقسیم میکنیم.

تابع getAnswer نیز قطر هارا استخراج می کند.

متد getAnswer : در این متد از خانه [0][n-1] شروع میکنیم که k بهینه در آن موجود است حال به صورت بازگشتی چند ضلعی را میشکنیم تا در هر بار شکستن قطر بهینه را پیدا کنیم، این فرایند تا جایی پیش می رود که ر ئوس از i تا j تشکیل چند ضلعی بدهند. همچنین در صورتی باید این قطر در نظر گرفته شود که هزینه داشته باشد که این در صورتی است که روی چند ضلعی اصلی

نیفتند شرط بزرگتر از یک بودن برای این است، اما چون در نهایت راس انتها باید به ابتدا وصل شود ولی رابطه اختلاف برای آن برقرار نیست چراکه به جای برابر یک بودن اختلاف، اختلاف برابر تعداد کل رئوس می شود که این حالت نیز نباید قطر رسم شود.

```
146     private void getAnswer(List<Pair<Pair<Double, Double>, Pair<Double, Double>
147             >> out, List<Pair<Double, Double>> coordinates,
148                     int[][] answer, int i, int j) {
149         if (i < 0 || j < 0 || Math.abs(i - j) < 2) {
150             return;
151         }
152         int k = answer[i][j];
153         if (Math.abs(i - j) > 1 && Math.abs(i - j) < coordinates.size() - 1) {
154             out.add(new Pair<>(coordinates.get(i), coordinates.get(j)));
155         }if (Math.abs(j - k) > 1 && Math.abs(j - k) < coordinates.size() - 1) {
156             out.add(new Pair<>(coordinates.get(k), coordinates.get(j)));
157         }if (Math.abs(i - k) > 1 && Math.abs(i - k) < coordinates.size() - 1) {
158             out.add(new Pair<>(coordinates.get(k), coordinates.get(i)));
159         }
160         getAnswer(out, coordinates, answer, i, k);
161         getAnswer(out, coordinates, answer, k, j);
162     }
163     private double getCost(List<Pair<Double, Double>> pairs, int i, int j, int k) {
164         var p1 = pairs.get(i);
165         var p2 = pairs.get(j);
166         var pk = pairs.get(k);
167         var c12 = Math.abs(i - j) > 1 && Math.abs(i - j) < coordinates.size()
168                 - 1 ? distance(p1, p2) : 0;
169         var c2k = Math.abs(j - k) > 1 && Math.abs(j - k) < coordinates.size()
170                 - 1 ? distance(p2, pk) : 0;
171         var pk1 = Math.abs(i - k) > 1 && Math.abs(i - k) < coordinates.size()
172                 - 1 ? distance(pk, p1) : 0;
173         return c12 + c2k + pk1;
174     }
```

کلیت الگوریتم و همه متد ها توضیح داده شدند، دو عکس نیز در پوشه resources وجود دارد که icon برنامه و دکمه ریست هستند.برای اجرا در ویندوز از فایل batch نوشته شده می توان استفاده کرد.(نیازمند maven)

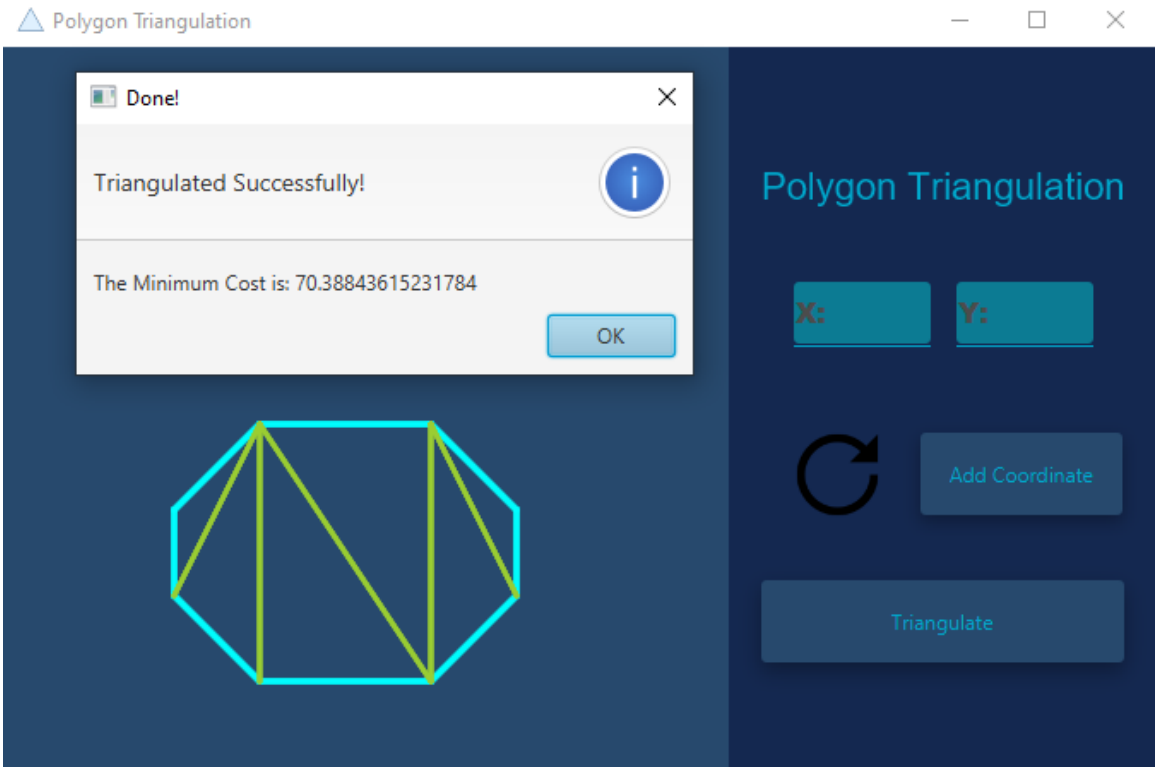
برای اجرا نیاز به Java 11 و Maven دارید، برای java های جدیدتر باید نسخه javafx را نیز در فایل pom.xml عوض کنید و نسخه سازگار با ورژن java خودتان را نصب کنید.

برای جاوا ها قدیمی تر باید فایل module-info را پاک کنید که اطلاعات این ماژول است و در جاوا های قبل از 9 نیاز نیست.

همچنین ورژن کتابخانه Jfoenix برای جاوا های قبل 9 باید به ورژن 8 تقلیل یابد ولی برای جاوا های جدیدتر موردی ندارد.

نمونه اجرا برای ورودی های:

(15,5), (25,5), (30,10), (30,15), (25,20), (15,20), (10,15), (10,10)



نمونه اجرا برای ورودی های:

(10,10) (10,20) (20,20) (20,10)