به نام خدا

پروژه درس کامپایلر

اعضای تیم:

مهران کامرانی

مهران محمدی

علی محمدی

فرشاد حسین پور

شامل توضیحات MYVISITOR و گرافیک برنامه

شرح کلاس MyVisitor

در ابتدا گرامر مربوطه را با توضیحات داده شده مینویسیم

یک hashmap به اسم memory که مقادیر متغیرها درون آن گذاشته میشود

```
private HashMap<String, String> memory = new HashMap<>();
```

****

و یک arrayList دیگر به اسم answer که برای print استفاده میشه:

```
private static ArrayList<String> answer = new ArrayList<>();
```

****

تابع isAcceptInput در ورودی کد را گرفته و بررسی میکند که آیا ورودی از نظر تحلیل گرامری و تحلیل لغوی درست است یا نه

```
public static ArrayList<String> isAcceptInput(ANTLRInputStream input) {
    textResult.clear();
    DemoLexer lexer = new DemoLexer(input);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    DemoParser parser = new DemoParser(tokens);
    DemoParser.ProgramContext tree = parser.program();
    DemoBaseVisitor<String> visitor = new MyVisitor();
    tree.accept(visitor);
    return textResult;
}
```

که DemoLexer و DemoParser و DemoBaseVisitor ورودی را بررسی میکنند و در آخر جواب چاپ میشود

****

در تابع visitProgram چود در گرامر این طور نوشته شده:

```
program: statements ;
```

بنابراین باید visit(statements) صدا زده شود :

```
@Override
public String visitProgram(DemoParser.ProgramContext ctx) {
   visit(ctx.statements());
   return null;
}
```

****

در تابع **visitStatements()** چون در گرامر اینطور است:

```
statements: (statement)+ ;
```

یعنی تا هرچقدر که جمله statements وجود دارد بخواند یعنی visit() کند:

```
@Override
public String visitStatements(DemoParser.StatementsContext ctx) {
   for (DemoParser.StatementContext temp : ctx.statement()) {
      visit(temp);
   }
return null;}
```

در این تابع در حلقه for هرچقدر که statements وجود دارد میخواند (visit())

****

در تابع **visitAssignment()** باید طبق گرامر:

```
statement: identifier equal expr semicolon
```

این گرامر ریختن عدد دریک متغیر را بررسی میکند که identifier همان متغیر هست و expr میتواند عدد,مقدار true, مقدار false, و... داشته باشد

پس باید در این تابع از دو visit() استفاده کنیم یکی برای identifier و دیگری برای expr :

```
@Override
public String visitAssignment(DemoParser.AssignmentContext ctx) {
   String variable = visit(ctx.identifier());
   String value = visit(ctx.expr());
   memory.put(variable, value);
   return null;
}
```

که در memory متغیر همراه با مقدارش ریخته میشود

در تابع **visitBegined()** برای گرامر زیر نوشته شده:

```
statement: openBracket statements closeBracket
```

که فقط باید statements را بخوانیم یعنی visite() کنیم بنابرین میشود:

```
@Override
public String visitBeginend(DemoParser.BeginendContext ctx) {
   visit(ctx.statements());
   return null;
}
```

****

درتابع **visitIf()** که گرامرش این است:

```
statement: 'if' opneParan expr closeParan statement
```

که ابتدا فقط باید expr را visit() کنیم اگر مقدارش true بود وارد مرحله بعدی میشویم یعنی که باید statement را نیز بخوانیم در غیر اینصورت هیچ

همان if در زبان های برنامه نویسی است اگر عبارت داخل if برابر true باشد وارد شرط داخل میشود

```
@Override
public String visitIf(DemoParser.IfContext ctx) {
   boolean result = visit(ctx.expr()).equals("true");
   if (result)
      visit(ctx.statement());
   return null;
}
```

متغیر result همان بررسی شرط داخل if است اگر true بود وارد شرط داخل if میشود

****

تابع **visitIfElse()** مانند تابع قبلی عمل میکند با این تفاوت که اگر شرط داخل if درست نباشد به else برود و اجرا کند

```
@Override
public String visitIfelse(DemoParser.IfelseContext ctx) {
   boolean result = visit(ctx.expr()).equals("true");
```

```
    if (result)
        visit(ctx.statement(0));
    else
        visit(ctx.statement(1));
    return null;
}
```

****

تابع **visitWhile()** با این گرامر:

```
statement:  'while' opneParan expr closeParan statement
```

اگر شرط داخل while درست باشد که برای این کار با استفاده از visit(expr) متوجه میشویم اگرtrue باشد وارد داخل while میشود که باید visit(statement) را صدا بزند البته باید این کار داخل یک حلقه انجام شود هر بار که این حلقه اجرا شود با توجه به شرط داخل آن اجرا میشود

```
@Override
public String visitWhile(DemoParser.WhileContext ctx) {
    while (visit(ctx.expr()).equals("true")) {
        visit(ctx.statement());
    }
    return null;
}
```

در داخل شرط while هر بار چک میشود که visit(ctx.expr) برابرtrue است اگر بود به داخل حلقه رفته و statement را visit میکند

****

تابع **visitFor()** با این گرامر:

```
statement:   'for' opneParan identifier equal number colon number closeParan statement
```

در ابتدا یه متغیر برای داخل شرط for ایجاد میکنیم که همان identifier است باید آن را visit() کنیم و خروجی را در یک متغیر ذخیره میکنیم

سپس باید عدد را visit (number) کنیم و عدد به دست را در یک متغیر دیگر ذخیره میکنیم

در memory مقدار متغیر و مقدار عدد را باهم ذخیره میکنیم

```
memory.put(variable, value);
```

و حالا باید عددی که متغیر تعریف شده در for باید تا آن عدد برود را بررسی کنیم با visit(number) که در یک متغیر دیگر داخل کد نوشته ریخته میشود

و حالا باید در داخل یک حلقه بگوییم که تا وقتی متغیر اولیه که در داخل memory ذخیره کردیم برابر با متغیری که بعد از to آمده برابر نشده همان طور visit(statement) کند و مقدار متغیر ذخیره شده داخل memory یکی افزایش و دوباره داخل آن قرار میگیرد

```
@Override
public String visitFor(DemoParser.ForContext ctx) {
    //assigning variable
    String variable = visit(ctx.identifier());
    String value = visit(ctx.number(0));
    memory.put(variable, value);

    //for loop
    double startValue = Double.parseDouble(value);
    double targetValue = Double.parseDouble(visit(ctx.number(1)));
    while (startValue <= targetValue) {
        visit(ctx.statement());
        startValue++;
        memory.put(variable, String.valueOf(startValue));
    }
    return null;
}
```

****

در تابع **visitPrint()** با این گرامر:

```
Statement :    'print' identifier semicolon
```

باید ابتدا identifier را مورد بررسی قرار دهیم که visit(identifier) است سپس خروجی را در داخل textResult قرار میدهیم تا در آخر به یک دفعه در خروجی چاپ شود

```
@Override
public String visitPrint(DemoParser.PrintContext ctx) {
    String variable = visit(ctx.identifier());
    answer.add(memory.get(variable));
```

```
    return null;
}
```

****

تابع **visitNot()** با گرامر:

```
expr: '!' expr
```

در این تابع باید ابتدا expr را مورد بررسی قرار دهیم این تابع چون فقط باید بر روی عدد عملیات انجام دهد بنابر این باید در regex عدد را پیدا کنیم

```
matches("\\d+(\\.\\d+)*")
```

اگر خروجی عدد بود بنابر این اگر عدد صفر باشد مقدارش true میشود و عدد دیگری جز صفر باشد false برمیگرداند و اگر به صورت boolean باشد اگر true باشد مقدارش false و برعکس

```
@Override
public String visitNot(DemoParser.NotContext ctx) {
    boolean result;
    String value = visit(ctx.expr());
    if (value.matches("\\d+(\\.\\d+)*")) {
        result = Double.parseDouble(value) != 0;
    } else result = value.equals("true");
    if (result)
        return "false";
    return "true";
}
```

****

در تابع **visitId()** که باید بررسی کنیم همچین متغیری در memory وجود دارد و سپس باید آن را با مقدارش return کنیم

****

تابع **visitBinopr()**: با این گرامر:

```
expr: expr binop expr
```

باید در این تابع سه visit() انجام اولی باید برای binop باشد که خروجی یکی از حالت های زیر است

```
binop      : '<' | '>' | '<=' | '>='  | '==' | '!=' ;
```

و داخل یک متغیر به نوع رشته ذخیره میکنیم

سپس باید دو visit() دیگر برای expr انجام دهیم و داخل یک متغیر ذخیره کنیم و بعد در داخل یک switch بگوییم مثلا اگر binop برابر > باشد بیاید مقدار آن را در یک متغیر از جنس boolean بریزد طبیعی است که اگر مقدار اولی در این مثال کوچک تر از مقدار دومی باشد باید در آن متغیر مقدار true برگرداند

```
switch (operator) {
   case "<":
      result = firstExpr < SecondExpr;
      break;
```

برای دیگر operator ها نیز به همین شیوه عمل میکنیم

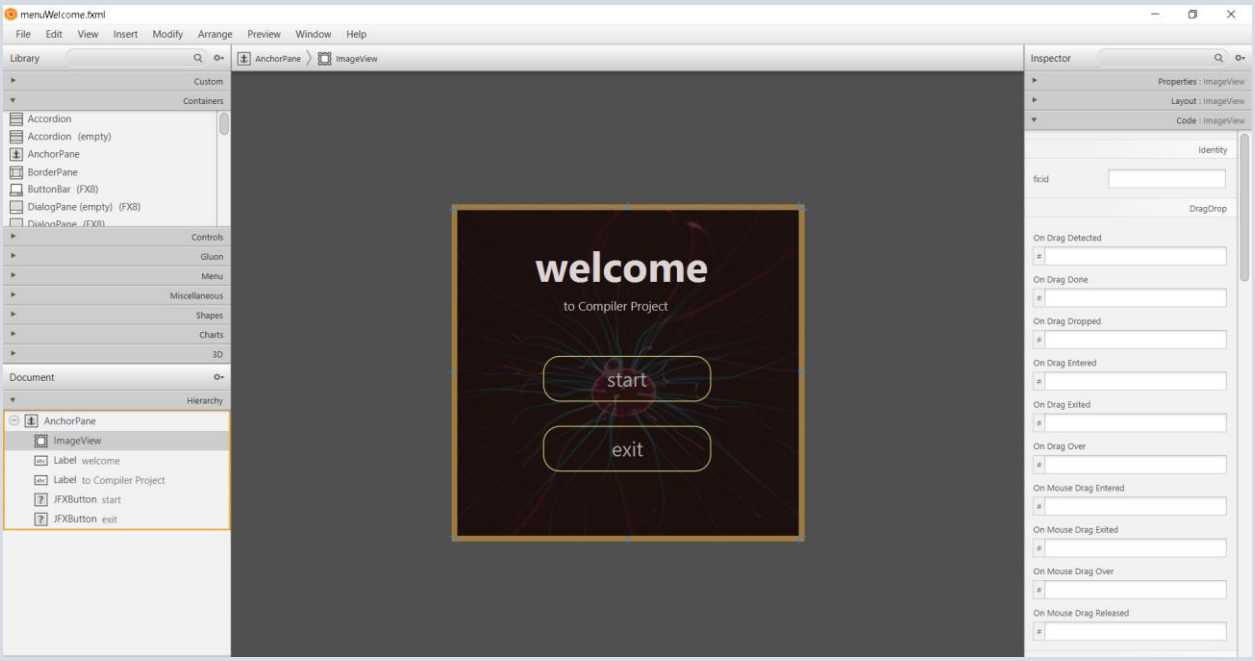
در بقیه تابع ها باید ctx.gettext را return کنیم چون لازم نیست شرطی را بررسی کنند

شرح گرافیک پروژه

گرافیک پروژه از سه قسمت تشکیل شده است

- **صفحه ورود**
- **صفحه وارد کردن کد**
- **صفحه نتیجه کد**

صفحه ورود:



برای welcome از lable استفاده شده و برای دکمه های موجود در صفحه از کتاب خانه JFonix که باید کتابخانه را دانلود و در داخل پروژه import کنیم استفاده شده است

برای background از یک تصویر که در imageviwe قرار دادیم استفاده کردیم

اگر دکمه start را فشار دهیم با استفاده از کد زیر:

```
Parent parent = FXMLLoader.load(getClass().getResource("/resource/fxmlfile/inputfile.fxml"));
```

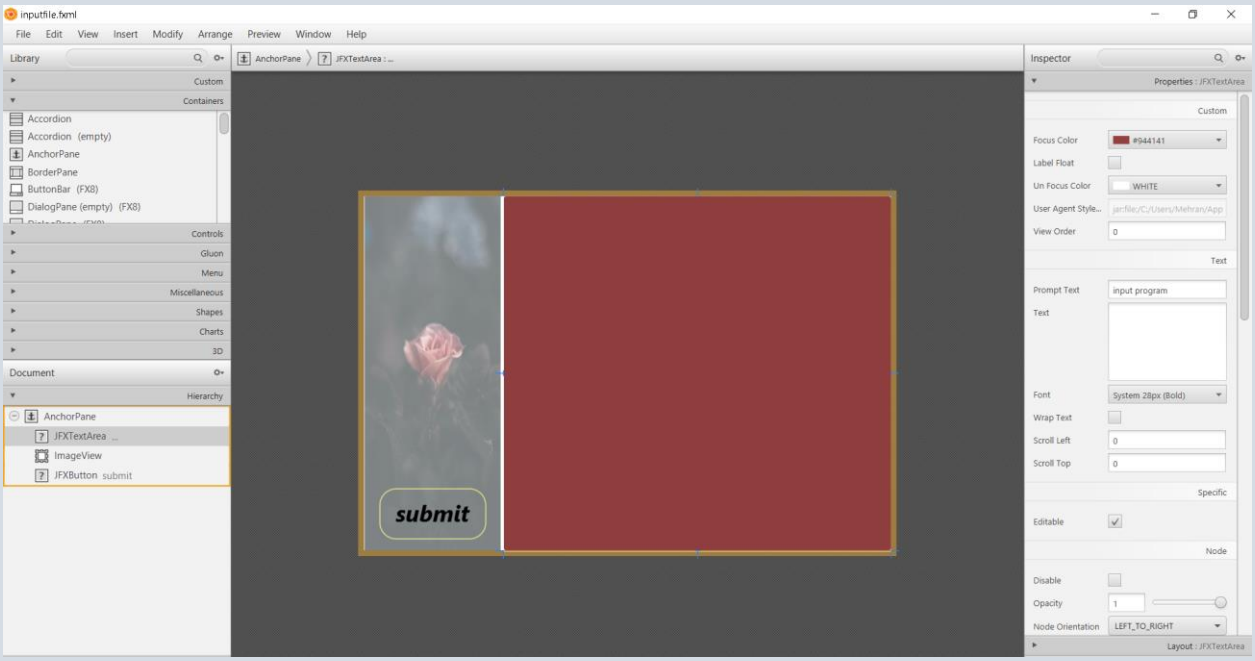
که در تابع loadMain() قرار دارد صفحه بعدی که برای وارد کردن کد ساخته شده باز میشود

اگر دکمه exit() زده شود با کد زیر:

```
System.exit(1);
```

بعد 1 ثانیه پنجره بسته خواهد شد

پنجره وارد کردن کد:



این پنجره از یک دکمه JFXButoum و یک JFXText area و همچنین یک imageviwe ساخته شده است

کاربر کد خود را در بخش text area وارد میکند و سپس دکمه submit را میفشارد در صورتی که کد نوشته شده توسط کاربر از لحاظ گرامر مورد قبول باشد صفحه result نمایش داده میشود

برای پرداز در جواب کد کاربر باید متن کاربر در یک متغیر رشته به اسمs ذخیره میشود

```
public static String s;
```

وقتی در دکمه submit کلیک میکند اول بررسی میکند که در textarea خالی نباشد

سپس با استفاده از کتابخانه antlr بر کد تحلیل گرامری و معنایی انجام میشود

```
ANTLRInputStream input = new
ANTLRInputStream(new
ByteArrayInputStream(intputii.getText().getBytes(Standa
rdCharsets.UTF_8)));
ArrayList<String> result =
MyVisitor.isAcceptInput(input);
```

سپس پاسخ را در یک لیست میریزد که در یک حلقه باید تمام جواب های قرار گرفته در آن لیست را در همان متغیر s میریزیم

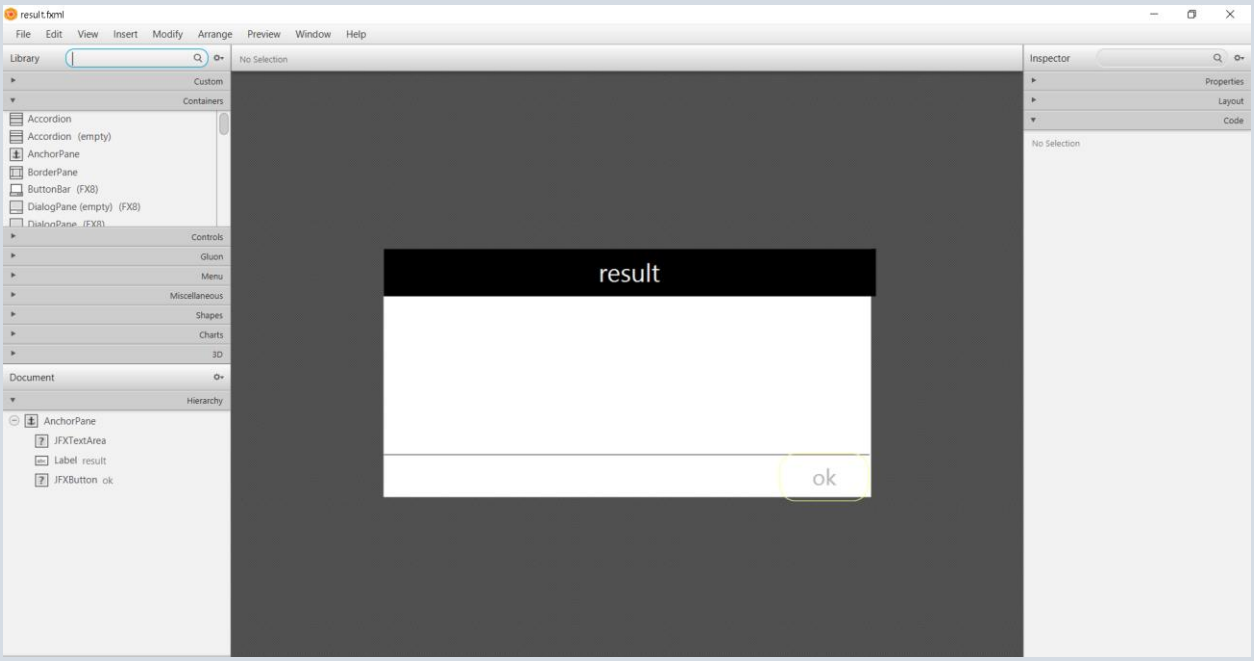
این متغیر از جنس static است که میتوانیم در کلاس بعدی که برای چاپ نتایج است استفاده میشود

```
Text bodyText = new Text(tempText.toString());
s=bodyText.getText();
```

نکته :

در این پروژه قسط داشتم که از controlfx برای اینکه وقتی کاربر به طور مثال حرف i را وارد کرد اتوماتیک برنامه به او کلمه if را نمایش دهد به طور کلی میخواستم که کامل کننده کلمه در textarea قرار دهم ولی این کتابخانه برای textareaاین کار را انجام نمیدهد و برای textField است که قطعا راه حلی برای این مشکل پیدا خواهم کرد و در گیتهاب قرار خواهم داد

صفحه result :



این صفحه از textarea برای نمایش خروجی به کاربر و یک دکمه ok برای تایید ورودی و برگشت به صفحه نوشتن کد توسط کاربر میرود

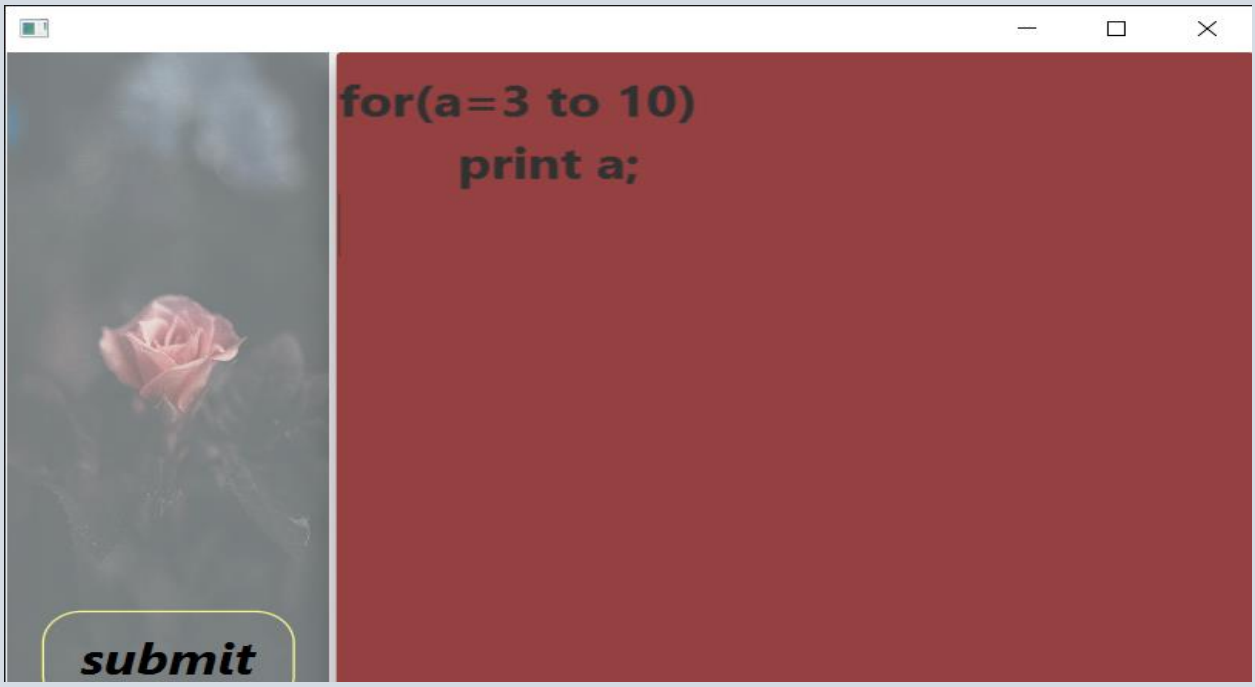
```
result.setText(ControlInputFIle.s);
```

با این کد محتویات داخل رشته s که در کلاس قبلی ایجاد کردیم و خروجی در آن قرار دارد درون این textarea قرار میگیرد

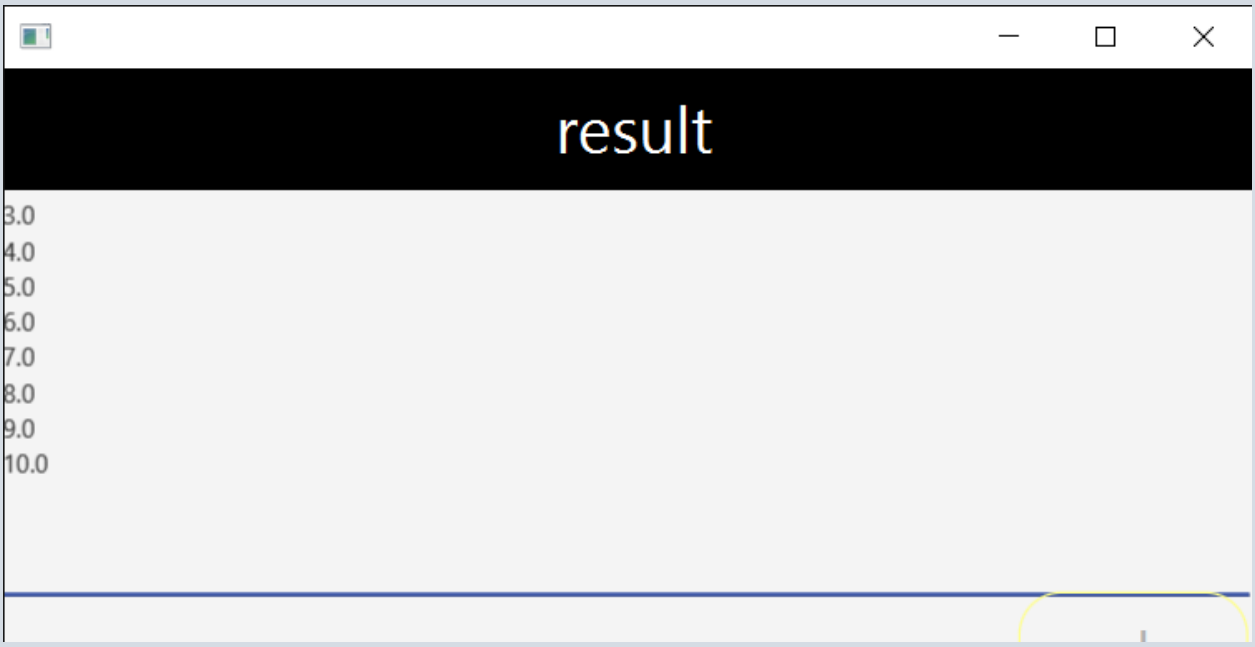
اگر ok زده شود با کد زیر:

```
((Stage) anchor.getScene().getWindow()).close();
```

پنجره بسته میشود و به بخش کد کاربر برمیگردیم



و جواب بعد کلیک دکمه submit:



نکته: گرافیک صفحه ها توسط fxml ساخته شده . ایده این طرح جداسازی گرافیک برنامه از کد اصلی است که خیلی بهتر است