МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С. П. КОРОЛЕВА»

(САМАРСКИЙ УНИВЕРСИТЕТ)

Отчёт по лабораторной работе

по курсу «Теория формальных языков и грамматик»

Вариант №12.

Выполнил:

Никитин А.А.

гр.6303

Проверил: Литвинов В. Г.

Самара 2016

**Задание:**

Написать программу синтаксического анализа автоматного языка вызова процедур языка Fortran - 77; цепочки языка имеют вид:

**CALL** <идентификатор>[**(**<список фактических параметров>**)**]

<список фактических параметров> :: = <параметр>[**,**<список фактических параметров>]

⎧<константа любая>

<параметр> :: = ⎨ ⎡**⎛**⎧<константа целая>⎫**⎞**⎤

⎪<идентификатор> ⎢**⎜**⎨ ⎬**⎟**⎥

⎩ ⎣**⎝**⎩<идентификатор> ⎭**⎠**⎦

<идентификатор> - идентификатор, начинается с буквы, включает последовательность букв, цифр, не допускает пробелы и специальные символы, ввести ограничение на длину (не более 8 символов) и не может быть зарезервированным словом (CALL, FORMAT, FOR, TO);

<константа целая> - целое число в диапазоне 1 ÷ +32767;

<константа любая> - целое число в диапазоне -32768 ÷ +32767, число с фиксированной точкой, число с плавающей точкой. Число цифр порядка ≤ 2.

Семантика:

Построить таблицу идентификаторов и констант с указанием номера параметра. Учесть перечисленные выше ограничения на идентификаторы и константы. Не допускать дублирование идентификаторов.

Сообщать об ошибках при анализе, указывая курсором место ошибки и ее содержание.

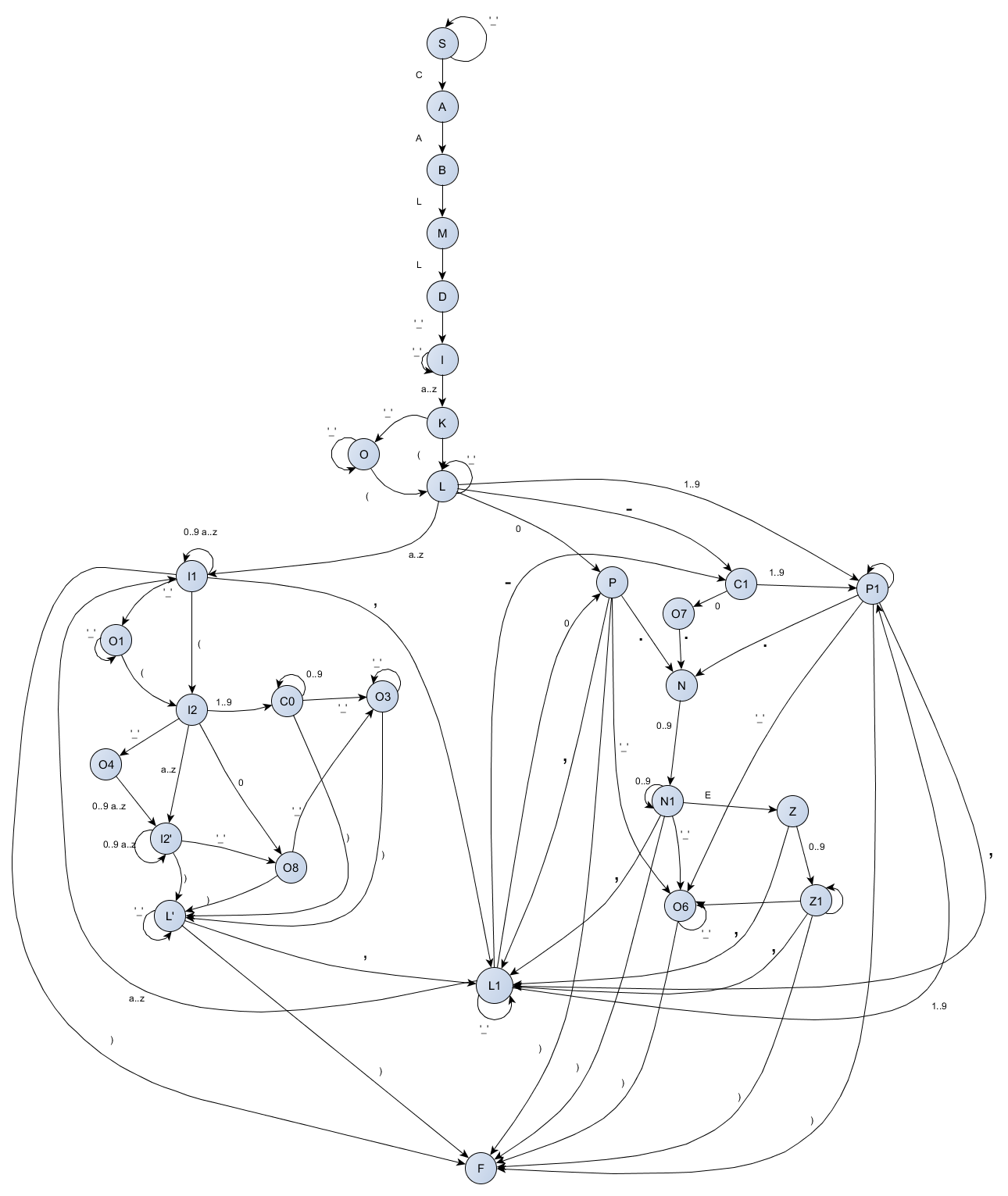
Примеры правильных цепочек:

CALL SINUS

CALL PRIMER1 ( ABC, 3.14, 0.25E4, 15 )

CALL PRIMER2 ( A ( B ), C ( 35 ), 34E-2, AB ( KB ) )

**Детерминированный конечный автомат языка:**



**Листинг программы:**

Класс анализатора:

package monitor.service;  
  
  
import monitor.service.entity.DataItem;  
import monitor.service.entity.Result;  
import monitor.service.entity.State;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.function.Function;  
  
import static monitor.service.entity.State.\*;  
  
  
public class Analyzer {  
  
 private State state;  
  
 private char current;  
  
 private String currentIdentificator;  
 private String currentConstant;  
  
 private List<DataItem> constants;  
 private List<DataItem> identificators;  
  
 private int currentId;  
  
 private String message;  
  
 private int i;  
  
 @SuppressWarnings("all")  
 public Result analyse(String inputString) {  
 constants = new ArrayList<>();  
 identificators = new ArrayList<>();  
 char[] symbols = inputString.toCharArray();  
 i = 0;  
 state = *S*;  
 currentIdentificator = "";  
 currentConstant = "";  
 currentId = 0;  
 while (state != *E* && state != *F* && i < symbols.length) {  
 current = symbols[i];  
 switch (state) {  
 case *S*:  
 if (current == 'C') {  
 state = *A*;  
 } else state = *E*;  
 break;  
  
 case *A*:  
 if (current == 'A') {  
 state = *B*;  
 } else state = *E*;  
 break;  
  
 case *B*:  
 if (current == 'L') {  
 state = *M*;  
 } else state = *E*;  
 break;  
  
 case *M*:  
 if (current == 'L') {  
 state = *D*;  
 } else state = *E*;  
 break;  
  
 case *D*:  
 if (current == ' ') {  
 state = *I*;  
 } else state = *E*;  
 break;  
  
 case *I*:  
 if (isIn('a', 'z')) {  
 state = *K*;  
 currentIdentificator += current;  
 } else {  
 if (current != ' ') {  
 state = *E*;  
 }  
 }  
 break;  
  
 case *K*:  
 if (current == '(') {  
 state = *L*;  
 addIdentificator();  
 } else if (current == ' ') {  
 state = *O*;  
 } else if (isIn('0', '9', 'a', 'z')) {  
 currentIdentificator += current;  
 } else state = *E*;  
 break;  
  
 case *L*:  
 if (isIn('1', '9')) {  
 state = *P1*;  
 currentConstant += current;  
 } else if (current == '-') {  
 state = *C1*;  
 currentConstant += current;  
 } else if (current == '0') {  
 state = *P*;  
 currentConstant += current;  
 } else if (isIn('a', 'z')) {  
 state = *I1*;  
 currentIdentificator += current;  
 } else if (current != ' ') {  
 state = *E*;  
 }  
 break;  
  
 case *P*:  
 if (current == ',') {  
 state = *L1*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (current == ')') {  
 state = *F*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (current == ' ') {  
 state = *O6*;  
 } else if (current == '.') {  
 state = *N*;  
 currentConstant += current;  
 } else state = *E*;  
 break;  
  
 case *O*:  
 if (current == '(') {  
 state = *L*;  
 addIdentificator();  
 } else if (current != ' ') {  
 state = *E*;  
 }  
 break;  
  
 case *I1*:  
 if (current == '(') {  
 state = *I2*;  
 addIdentificator();  
 } else if (current == ' ') {  
 state = *O1*;  
 } else if (current == ')') {  
 state = *F*;  
 addIdentificator();  
 } else if (current == ',') {  
 state = *L1*;  
 addIdentificator();  
 } else if (isIn('0', '9', 'a', 'z')) {  
 currentIdentificator += current;  
 } else state = *E*;  
 break;  
  
 case *O1*:  
 if (current == '(') {  
 state = *I2*;  
 addIdentificator();  
 } else if (current == ',') {  
 state = *L1*;  
 addIdentificator();  
 } else if (current != ' ') {  
 state = *E*;  
 }  
 break;  
  
 case *I2*:  
 if (current == ' ') {  
 state = *O4*;  
 } else if (isIn('1', '9')) {  
 state = *C0*;  
 currentConstant += current;  
 /\*} else if (current == '-') {  
 state = C0o;\*/  
 } else if (isIn('a', 'z')) {  
 state = *I2o*;  
 currentIdentificator += current;  
 } else if (current == '0') {  
 state = *O8*;  
 currentConstant += current;  
 } else {  
 state = *E*;  
 }  
 break;  
  
 case *C0*:  
 if (current == ' ') {  
 state = *O3*;  
 } else if (current == ')') {  
 state = *Lo*;  
 addConstant(Analyzer::*isInteger*);  
 } else if (isIn('0', '9')) {  
 currentConstant += current;  
 } else state = *E*;  
 break;  
  
 case *O3*:  
 if (current == ')') {  
 state = *Lo*;  
 addConstant(Analyzer::*isInteger*);  
 } else if (current != ' ') {  
 state = *E*;  
 }  
 break;  
  
  
 case *O8*:  
 if (current == ' ') {  
 state = *O3*;  
 } else if (current == ')') {  
 state = *Lo*;  
 addConstant(Analyzer::*isInteger*);  
 addIdentificator();  
 } else state = *E*;  
 break;  
  
 case *O4*:  
 if (isIn('a', 'z')) {  
 state = *I2o*;  
 currentIdentificator += current;  
 /\*} else if (current == '-') {  
 state = C0o;\*/  
 } else if (isIn('1', '9')) {  
 state = *C0*;  
 currentConstant += current;  
 } else if (current != ' ') {  
 state = *E*;  
 }  
 break;  
  
 case *I2o*:  
 if (current == ' ') {  
 state = *O8*;  
 } else if (current == ')') {  
 state = *Lo*;  
 addIdentificator();  
 } else if (isIn('0', '9', 'a', 'z')) {  
 currentIdentificator += current;  
 } else state = *E*;  
 break;  
  
 case *Lo*:  
 if (current == ' ') {  
 state = *O5*;  
 } else if (current == ',') {  
 state = *L1*;  
 } else if (current == ')') {  
 state = *F*;  
 } else state = *E*;  
 break;  
  
 case *O5*:  
 if (current == ',') {  
 state = *L1*;  
 } else if (current == ')') {  
 state = *F*;  
 } else if (current != ' ') {  
 state = *E*;  
 }  
 break;  
  
 case *C1*:  
 if (current == '0') {  
 state = *O7*;  
 currentConstant += current;  
 } else if (isIn('1', '9')) {  
 state = *P1*;  
 currentConstant += current;  
 } else state = *E*;  
 break;  
  
 case *P1*:  
 if (current == ')') {  
 state = *F*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (current == 'E') {  
 state = *Z*;  
 currentConstant += current;  
 } else if (current == ' ') {  
 state = *O6*;  
 } else if (current == ',') {  
 addConstant(Analyzer::*isAnyConstant*);  
 state = *L1*;  
 } else if (current == '.') {  
 state = *N*;  
 currentConstant += current;  
 } else if (isIn('0', '9')) {  
 currentConstant += current;  
 } else state = *E*;  
 break;  
  
  
 case *O7*:  
 if (current == '.') {  
 state = *N*;  
 currentConstant += current;  
 } else state = *E*;  
 break;  
  
  
 case *N*:  
 if (isIn('0', '9')) {  
 state = *N1*;  
 currentConstant += current;  
 } else state = *E*;  
  
 break;  
  
  
 case *N1*:  
 if (current == 'E') {  
 state = *Z*;  
 currentConstant += current;  
 } else if (current == ' ') {  
 state = *O6*;  
 } else if (current == ')') {  
 state = *F*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (current == ',') {  
 state = *L1*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (isIn('0', '9')) {  
 currentConstant += current;  
 } else state = *E*;  
 break;  
  
 case *O6*:  
 if (current == ',') {  
 state = *L1*;  
 } else if (current == ')') {  
 state = *F*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (current != ' ') {  
 state = *E*;  
 }  
 break;  
  
 case *Z*:  
 if (isIn('0', '9')) {  
 state = *Z1*;  
 currentConstant += current;  
 } else if (current == ' ') {  
 state = *O6*;  
 } else if (current == ',') {  
 state = *L1*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (current == '-') {  
 state = *Z2*;  
 currentConstant += current;  
 }  
 break;  
  
 case *Z2*:  
 if (isIn('0', '9')) {  
 state = *Z1*;  
 currentConstant += current;  
 } else state = *E*;  
 break;  
  
 case *Z1*:  
 if (current == ')') {  
 state = *F*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (current == ' ') {  
 state = *O6*;  
 } else if (current == ',') {  
 state = *L1*;  
 addConstant(Analyzer::*isAnyConstant*);  
 } else if (isIn('0', '9')) {  
 currentConstant += current;  
 } else state = *E*;  
 break;  
  
 case *L1*:  
 if (current == '0') {  
 state = *P*;  
 currentConstant += current;  
 } else if (isIn('1', '9')) {  
 state = *P1*;  
 currentConstant += current;  
 } else if (current == '-') {  
 state = *C1*;  
 currentConstant += current;  
 } else if (isIn('a', 'z')) {  
 state = *I1*;  
 currentIdentificator += current;  
 } else if (current != ' ') {  
 state = *E*;  
 }  
 break;  
 }  
 i++;  
 }  
 addIdentificator();  
 Result result = new Result();  
 if (i < symbols.length) {  
 result.setSuccess(false);  
 if (message != null) {  
 result.setMessage(message);  
 } else result.setMessage("Синтаксическая ошибка!");  
 result.setPosition(i-1);  
 } else {  
 switch (state) {  
 case *E*:  
 result.setSuccess(false);  
 if (message != null) {  
 result.setMessage(message);  
  
 } else result.setMessage("Синтаксическая ошибка!");  
 result.setPosition(i - 1);  
 break;  
 case *F*:  
 case *K*:  
 case *O*:  
 result.setSuccess(true);  
 result.setConstants(constants);  
 result.setIdentificators(identificators);  
 break;  
 default:  
 result.setSuccess(false);  
 result.setMessage("Ошибка! Неожиданный конец строки!");  
 result.setPosition(i - 1);  
 }  
 }  
 return result;  
 }  
  
 private void addIdentificator() {  
 if (!currentIdentificator.isEmpty()) {  
 if (*isIdentificator*(currentIdentificator)) {  
 if (!identificators.stream().anyMatch(dataItem -> dataItem.getData().equals(currentIdentificator))) {  
 identificators.add(new DataItem(currentId++, currentIdentificator));  
 currentIdentificator = "";  
 } else {  
 state = *E*;  
 message = "Повторяющийся идентификатор: " + currentIdentificator;  
 }  
  
 } else {  
 state = *E*;  
 message = "Некорректный идентификатор: " + currentIdentificator;  
 }  
 }  
 }  
  
 private void addConstant(Function<String, Boolean> function) {  
 if (!currentConstant.isEmpty()) {  
 if (function.apply(currentConstant)) {  
 constants.add(new DataItem(currentId++, currentConstant));  
 currentConstant = "";  
 } else {  
 state = *E*;  
 message = "Некорректная константа: " + currentConstant;  
 i--;  
 }  
 }  
 }  
  
  
 private boolean isIn(char... chars) {  
 if (chars.length == 1) {  
 return Character.*toLowerCase*(chars[0]) == Character.*toLowerCase*(current);  
 }  
 boolean condition = false;  
 for (int i = 0; i < chars.length; i += 2) {  
 condition = condition || Character.*toLowerCase*(current) >= Character.*toLowerCase*(chars[i]) && Character.*toLowerCase*(current) <= Character.*toLowerCase*(chars[i + 1]);  
 }  
 return condition;  
 }  
  
  
 public static boolean isInteger(String string) {  
 try {  
 int val = Integer.*parseInt*(string);  
 return val >= 1 && val <= 32767;  
 } catch (NumberFormatException nfe) {  
 return false;  
 }  
 }  
  
 public static boolean isAnyConstant(String string) {  
  
 if (string.contains(".") || string.contains("E") || string.contains("e")) {  
 int index = string.indexOf('E') != -1 ? string.indexOf('E') : string.indexOf('e');  
 if (index != -1) {  
 String intVal = string.substring(0, index);  
 try {  
 Double.*parseDouble*(intVal);  
 } catch (NumberFormatException nfe) {  
 return false;  
 }  
 if (index == string.length()) {  
 return true;  
 }  
 String doubleVal = string.substring(index + 1, string.length());  
 return doubleVal.length() <= 2;  
 } else {  
 try {  
 Double.*parseDouble*(string);  
 return true;  
 } catch (NumberFormatException nfe) {  
 return false;  
 }  
 }  
 } else {  
 try {  
 int val = Integer.*parseInt*(string);  
 return val >= -32768 && val <= 32767;  
 }catch (NumberFormatException n){  
 return false;  
 }  
 }  
 }  
  
 public static boolean isIdentificator(String string) {  
 return string.length() <= 8  
 && !string.equalsIgnoreCase("CALL")  
 && !string.equalsIgnoreCase("FORMAT")  
 && !string.equalsIgnoreCase("FOR")  
 && !string.equalsIgnoreCase("TO");  
 }  
}

Класс графического интерфейса:

package monitor.view;  
  
import javafx.application.Application;  
import javafx.application.Platform;  
import javafx.collections.FXCollections;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.fxml.Initializable;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Label;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.TextField;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.stage.Stage;  
import monitor.service.Analyzer;  
import monitor.service.entity.DataItem;  
import monitor.service.entity.Result;  
  
import java.net.URL;  
import java.util.ResourceBundle;  
  
  
public class AnalyseView extends Application implements Initializable {  
  
 public TableColumn<DataItem, Integer> id1;  
 public TableColumn<DataItem, Integer> id2;  
 @FXML  
 private TableColumn<DataItem, String> identificators;  
 @FXML  
 private TableColumn<DataItem, String> constants;  
 @FXML  
 private TableView<DataItem> constantsTable;  
 @FXML  
 private TableView<DataItem> identificatorsTable;  
  
 @FXML  
 private TextField input;  
 @FXML  
 private Label output;  
  
 @Override  
 public void start(Stage stage) throws Exception {  
 stage.setTitle("Analyzer");  
 Parent root = FXMLLoader.*load*(getClass().getClassLoader().getResource("main.fxml"));  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.show();  
 }  
  
  
 @FXML  
 private void analyze(ActionEvent event) throws Exception {  
 new Thread(() -> {  
 Analyzer analyzer = new Analyzer();  
 Result result = analyzer.analyse(input.getText());  
 if (result.isSuccess()) {  
 Platform.*runLater*(() -> {  
 output.setText("Успешно!");  
 constantsTable.setItems(FXCollections.*observableList*(result.getConstants()));  
 identificatorsTable.setItems(FXCollections.*observableList*(result.getIdentificators()));  
 });  
 }else {  
 Platform.*runLater*(()->{  
 output.setText(result.getMessage() + " Позиция:" + result.getPosition());  
 input.requestFocus();  
 input.selectRange(result.getPosition(), result.getPosition()+1);  
 });  
  
 }  
 }).start();  
 }  
  
 @Override  
 public void initialize(URL url, ResourceBundle rb) {  
 identificators.setCellValueFactory(new PropertyValueFactory<DataItem, String>("data"));  
 constants.setCellValueFactory(new PropertyValueFactory<DataItem, String>("data"));  
 id1.setCellValueFactory(new PropertyValueFactory<DataItem, Integer>("id"));  
 id2.setCellValueFactory(new PropertyValueFactory<DataItem, Integer>("id"));  
 }  
}

**Результат выполнения программы:**

