Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

Кафедра «САПР ВС»

Пояснительная записка

к курсовой работе

по дисциплине

«Компьютерная графика»

На тему «Использование JavaFX для разработки графического приложения»

Выполнил:

студент группы 143

Кузнецов Д.А.

Проверил:

доц. Митрошин А. А.

Рязань 2023

# Введение

Разработка графического приложения в данное время занимает одно из главных мест в разработке ПО. В современных приложениях без графического интерфейса никуда, так как пользователю практически всегда необходимо вести диалог с программой. На языке Java существуют библиотеки для разработки графических интерфейсов, такие как Swing, JavaFX, JOGL, LWJGL, JME, Ogre4i, Java 3D, OpenCV и множество других.

# Описание JavaFX

JavaFX используется для создания разнообразных приложений с графическим интерфейсом, которые охватывают различные области. Вот несколько примеров приложений, для которых JavaFX часто выбирается в качестве технологии разработки:

**Управление Проектами:** Приложения для управления проектами и задачами, где графический интерфейс играет важную роль. JavaFX обеспечивает возможность создания удобных панелей инструментов, интерактивных диаграмм и динамичных списков для эффективного управления задачами.

**Образовательные Приложения:** Создание образовательных программ с интерактивными элементами, такими как визуализации, викторины и обучающие игры. JavaFX обеспечивает инструменты для создания интересных и вовлекающих образовательных приложений.

**Инструменты Визуализации Данных:** JavaFX широко используется для создания приложений визуализации данных, таких как графики, диаграммы, графы и карты. Это может быть полезно в областях анализа данных, бизнес-интеллекта и научных исследований.

**Мультимедийные Плееры:** Разработка мультимедийных плееров, поддерживающих воспроизведение аудио и видео файлов. JavaFX предоставляет удобные средства для работы с мультимедийным контентом.

**Игры:** JavaFX может использоваться для создания 2D и 3D игр с привлекательной графикой и анимациями. Это позволяет разработчикам легко интегрировать графические ресурсы и управлять игровым циклом.

**Системы Управления Базами Данных:** Создание приложений для управления базами данных с использованием графического интерфейса для добавления, редактирования и просмотра данных. JavaFX обеспечивает средства для создания форм ввода и отображения данных в удобной форме.

**Приложения для Работы с Графикой:** Разработка графических приложений для обработки изображений, рисования и редактирования графики. JavaFX предоставляет API для работы с графическим контентом.

# Задача

В задачах данной курсовой работы включены пункты:

1. Разработка программного обеспечения построения диаграмм по данным формата csv
2. ПО должно импортировать данные из файла csv и помещать их в некую структуру
3. ПО должно уметь строить диаграммы по импортируемым данным

# Подключение JavaFX к проекту (в eclipse)

Для начала необходимо скачать JavaFX SDK с официального сайта JavaFX [1]. Распаковать и поместить в любое место на диске. После нужно скачать плагин для eclipse для работы с JavaFX под названием e(fx)clipse (рисунок 1).

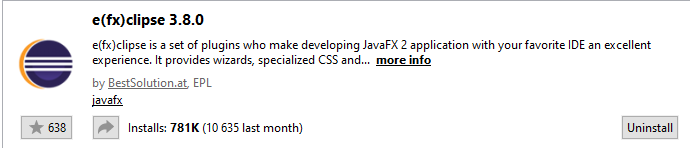


Рисунок 1 - Плагин e(fx)clipse

После в настройка необходимо указать путь к JavaFX SDK (рисунок 2).

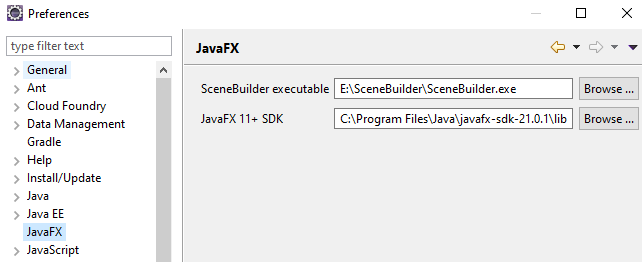


Рисунок 2 - Настройки плагина для JavaFX

После чего можно уже создавать проект. Сделать это можно с помощью File -> New -> Project... -> JavaFX -> JavaFX Project. После 1 запуска проекта в консоле выводится ошибка, представленная на рисунке 3.

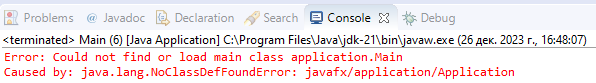


Рисунок 3 - Ошибка при запуске проекта JavaFX

Исправить её можно следующим способом: ПКМ по проекту -> Run as ->   
Run Configurations -> Arguments. В данном окне в поле VM arguments необходимо ввести следующую строку – «--module-path "Путь к JavaFX SDK" --add-modules javafx.controls,javafx.fxml» (рисунок 4).

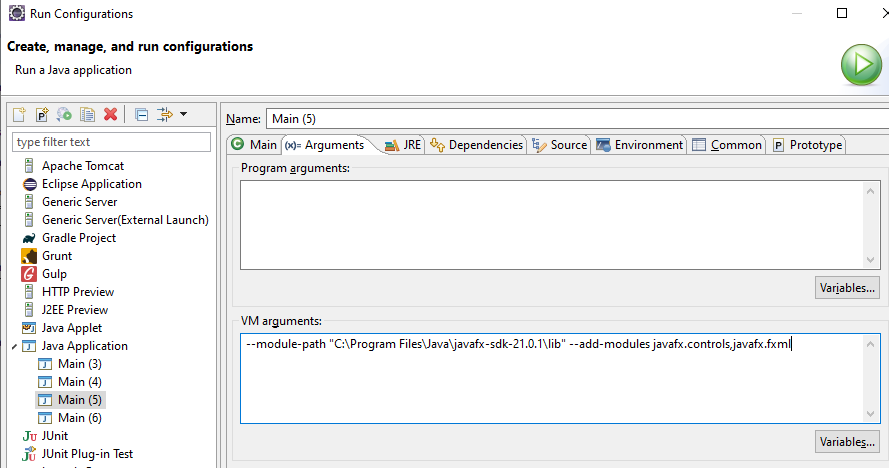


Рисунок 4 - Настройка конфигурации запуска проекта

# Разработка ПО

Для начала была создана структура, которая будет хранить данных из файла csv и будет неким посредником между данными и построением диаграмм. Данная структура была названа как DataTable, и далее будет использоваться это обозначение. DataTable имеет 3 свойства, такие как Список столбцов, список строк, а также файл, который хранит данные. Каждый столбец характеризуется именем. Каждая строка имеет следующее поле – список значений в виде строк.

ПО состоит из 1 окна, это окно имеет боковое меню для перехода между страницами. Сами страницы следующие:

1. Главная – начальная страница пользователя.
2. Данные – страница для работы с данными.
3. Диаграммы – страница для построения диаграмм.
4. Справка – страница для получения информации о ПО.

Cтилевое оформление бокового меню можно рассмотреть на рисунке 5.

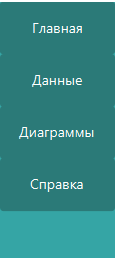


Рисунок 5 - Боковое меню

При наведении на одну из кнопок, стиль кнопки меняется (рисунок 6).

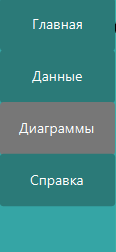


Рисунок 6 - Боковое меню при наведении на кнопку "Диаграммы"

В дополнение к этому была создана функция закрытия данного бокового меню. Данная кнопка была размещена справа от бокового меню и сделана в виде «кнопки-бургера» в обведенном круге (рисунок 7).

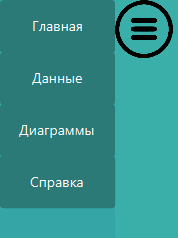


Рисунок 7 - Боковое меню с "кнопкой-бургером"

При нажатии на данную кнопку боковое меню выходит за пределы окна плавно в виде анимации. Если нажать на «кнопку-бургер» еще раз, то окно также плавно выдвинется. Сравнить окно с видимым и невидимым боковым окном можно на рисунке 8.

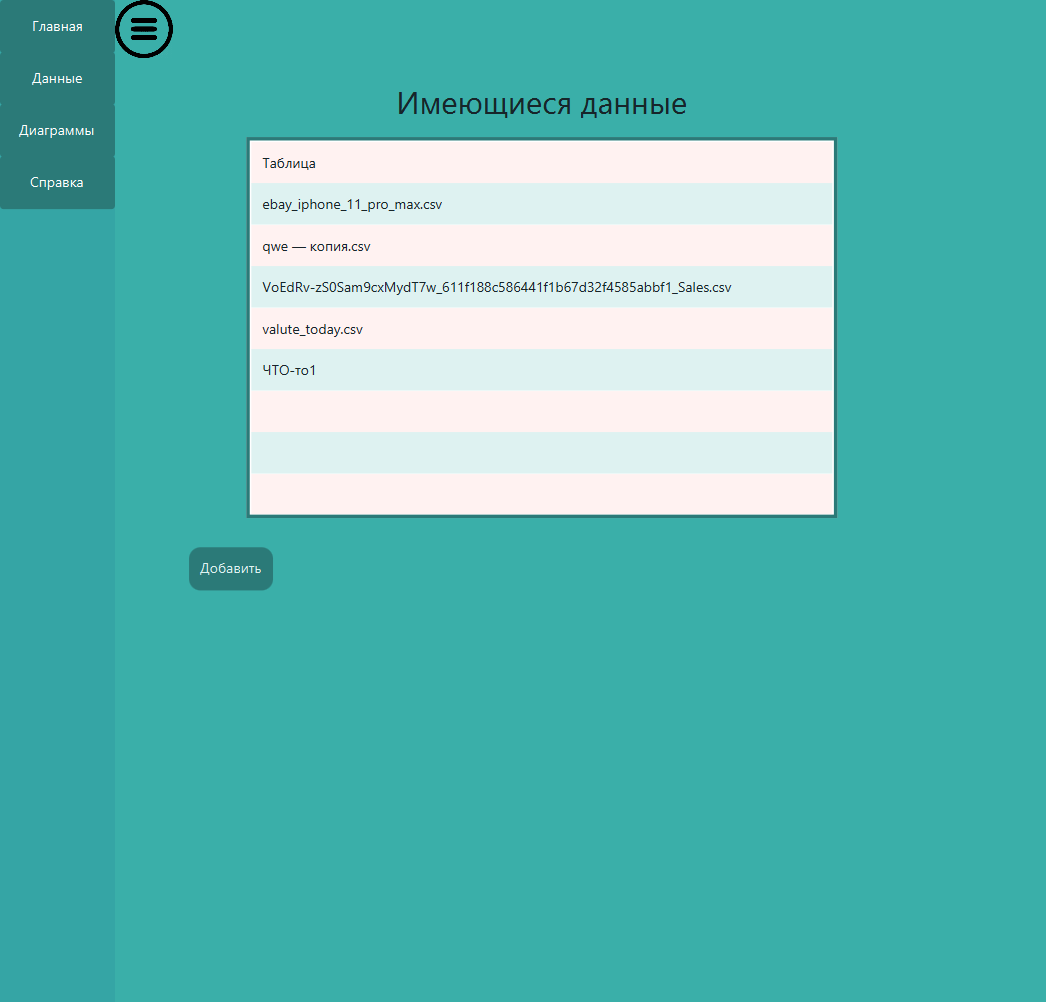
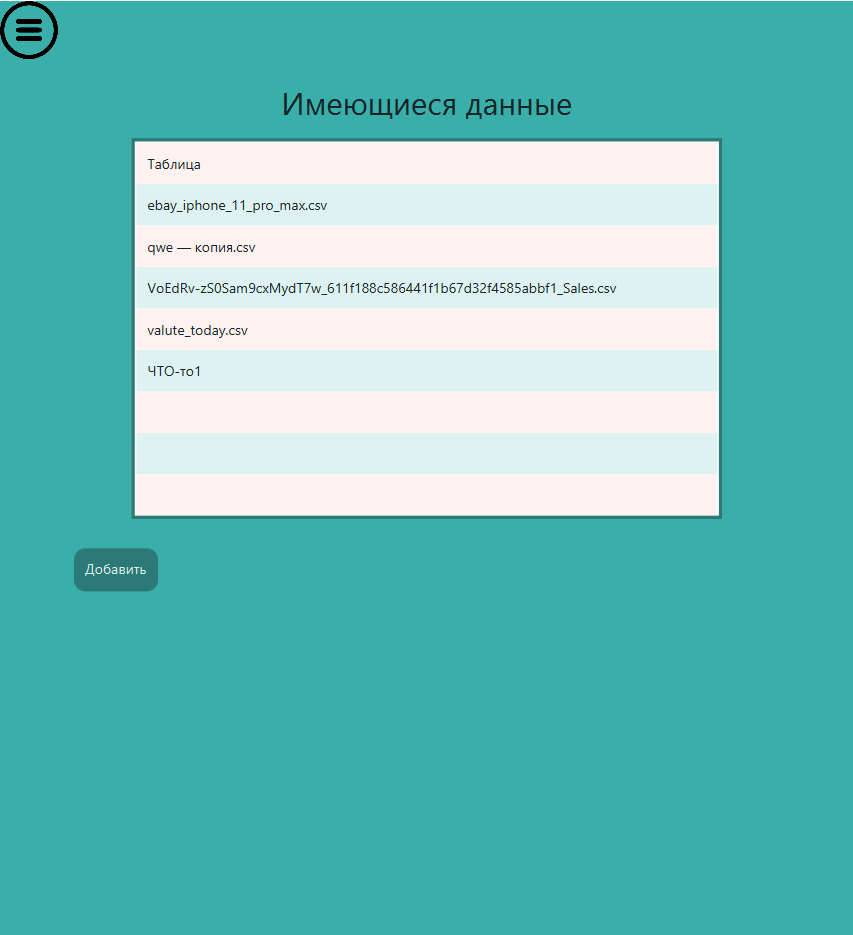
 

Рисунок 8 - Окна с видимым (слева) и невидимым (справа) боковым меню

Главную страницу программы можно посмотреть на рисунке 8. Главным элементом на странице является список имеющихся данных – данных, которые ранее уже были импортированы в программу (рисунок 9). При нажатии на элемент списка, программа перейдет на страницу «Данные» и отобразит данные из файла csv в виде таблицы.

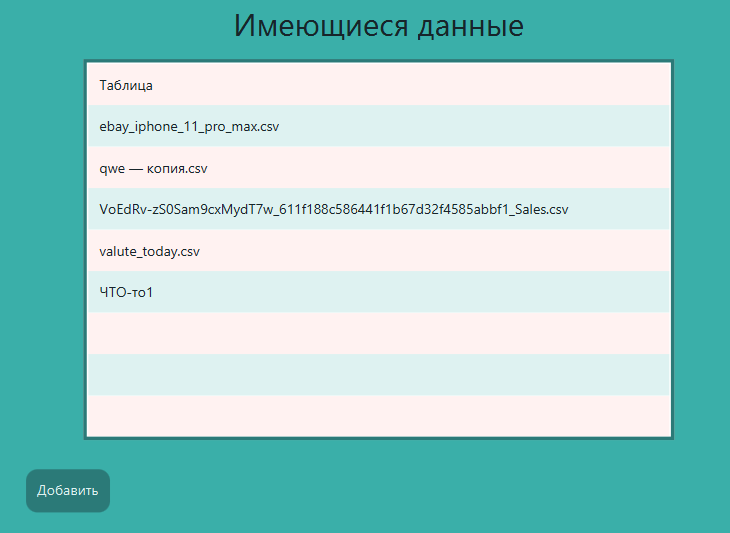


Рисунок 9 - Список имеющихся данных на главной странице

Ниже списка можно заметить кнопку «Добавить» (рисунок 9), при нажатии на которую пользователю будет предложено выбрать файл формата csv для импорта в программу. При успешном выборе файла, его название появится в списке имеющихся данных.

Все кнопки, характеризующие какое-либо действие (в том числе и вышеописанная кнопка «Добавить»), имеют стилевое оформление при обычном состоянии, наведении и нажатии (рисунок 10).

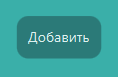
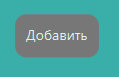
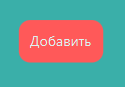
  

Рисунок 10 - Стилевое оформление кнопок. Обычное - слева.   
При наведении - в центре. При нажатии - справа.

Интерфейс страницы «Данные» представлен на рисунке 11.

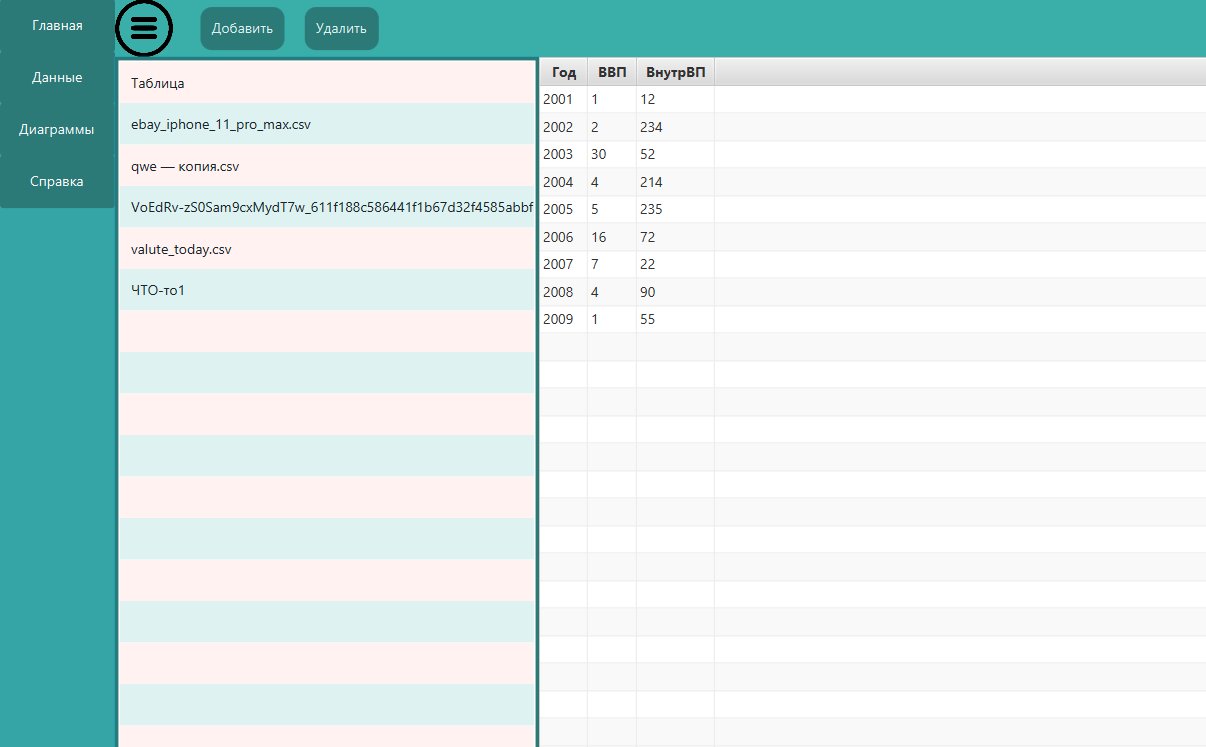


Рисунок 11 - интерфейс страницы "Данные"

Страница разделена на 3 части – список названий таблиц данных, таблица данных и панель инструментов. Список названий таблиц данных схож со списком имеющихся данных со страницы «Главная». Имеется несколько исключений – при нажатии на элемент списка нет перенаправления на другую страницу, а данные из таблицы с таким именем отображаются в таблице данных справа. И еще одно отличие – при двойном нажатии ЛКМ по элементу списка можно изменить название таблицы данных (рисунок 12).

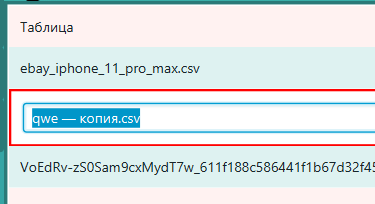


Рисунок 12 - Изменение названия таблицы данных

При нажатии на название таблицы данных из списка, данных отобразятся в таблице, расположенной справа и занимающей большую часть окна.

При нажатии на название таблицы данных, если файл связанный с этой таблицей (с DataTable) не найден, то выведется соответствующая ошибка (рисунок 13).

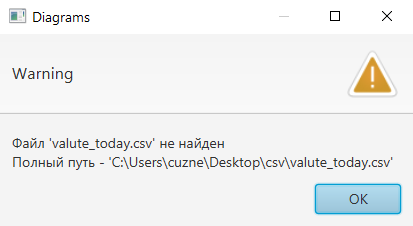


Рисунок 13 - Ошибка отсутствия файла

Сверху расположена панель инструментов (рисунок 14), состоящая из 2 кнопок «Добавить» и «Удалить». Кнопка «Добавить» идентична одноименной кнопки с главной страницы и также добавляет новую таблицу данных. С помощью кнопки «Удалить» можно удалить таблицу данных, но при этом её необходимо выбрать из списка.



Рисунок 14 - Панель инструментов страницы "Данные"

Интерфейс страницы «Диаграммы» представлен на рисунке 15. Данная страница разделена на 4 сегмента – доступные данные, диаграммы, панель настроек диаграммы и контейнер для самой диаграммы (на рисунке 15 контейнер пуст).

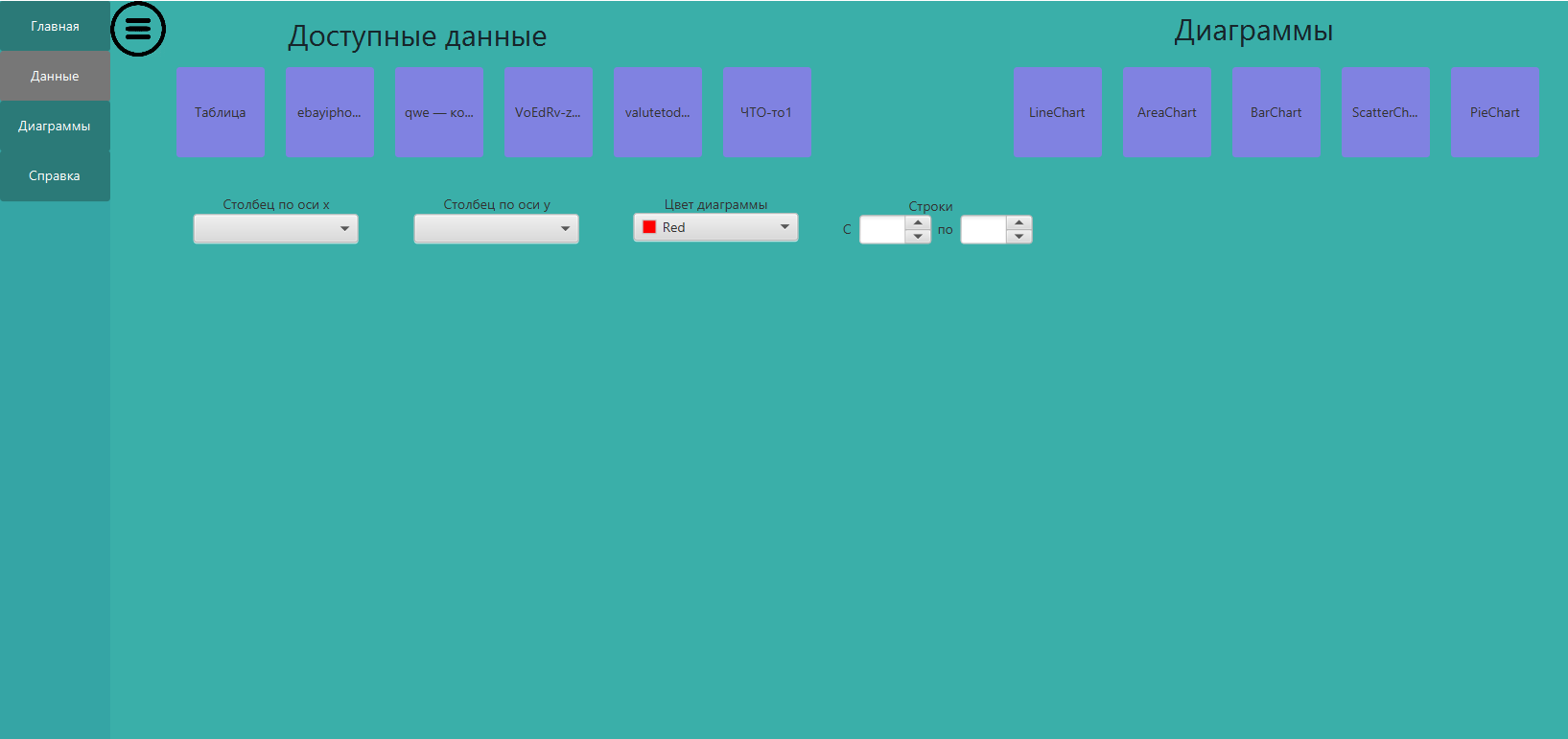


Рисунок 15 - Интерфейс страницы "Диаграммы"

«Доступные данные» - это данные которые были импортированы раннее на страницах «Главная» и «Данные». Здесь они изображены не в виде списка, а в виде множества кнопок, на которые можно нажать чтобы выбрать для основы построения диаграммы.

Выбрать только таблицу для построения диаграммы недостаточно, необходимо еще выбрать тип диаграммы (в данный момент программа может построить 5 видов диаграмм: линейная, диаграмма области, столбчатая, точечная и круговая).

Если выбрать таблицу и тип диаграммы, то построится диаграмма (рисунок 16).

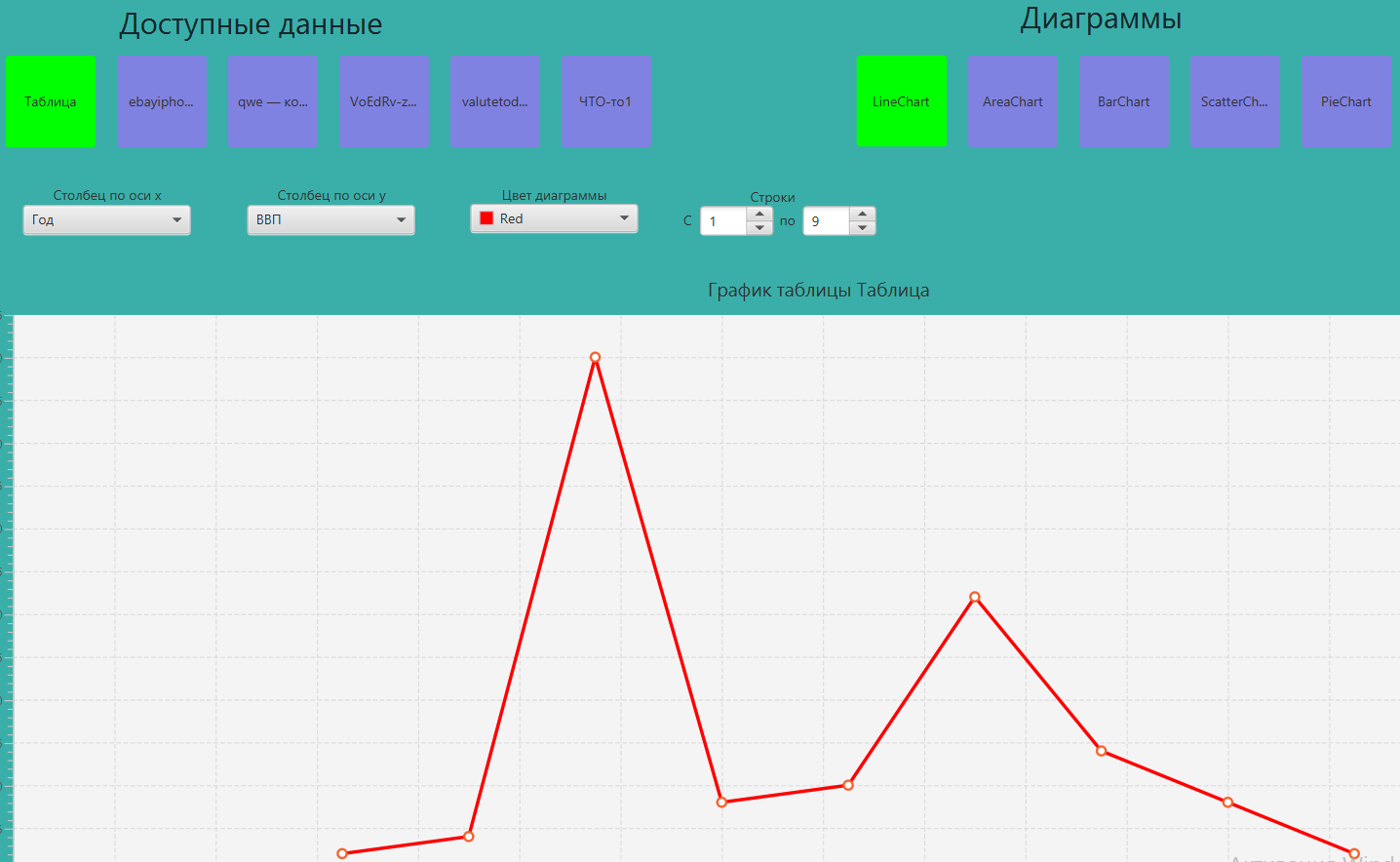


Рисунок 16 - Линейная диаграмма по данным таблицы с названием "Таблица"

Для каждого вида диаграммы имеется свой набор настроек, то есть при изменении типа диаграммы меняется набор настроек на панели настроек. Например, круговая диаграмма не имеет настройки цвета, в отличии от других диаграмм (рисунок 17).

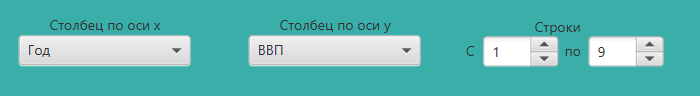


Рисунок 17 - Наборы настроек для линейной диаграммы (сверху)  
и круговой диаграммы (снизу)

Назначение каждой настройки представлено в таблице 1.

Таблица 1 - Настройки диаграммы

|  |  |
| --- | --- |
| **Название** | **Назначение** |
| Столбец по оси x | Столбец из таблицы данных, который будет использоваться для построения диаграммы для оси x |
| Столбец по оси y | Столбец из таблицы данных, который будет использоваться для построения диаграммы для оси y |
| Цвет диаграммы | Цвет, которым будет отображаться диаграмма |
| Строки | Диапазон строк, который будет использоваться для построения диаграммы |

Для кнопок-переключателей диаграмм и таблиц создано 3 вида стилей. Обычный стиль, стиль при наведении и стиль при нажатии. Эти стили изображены на рисунке 18.

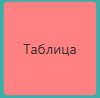
  

Рисунок 18 - Стили кнопок-переключателей диаграмм и таблиц. Обычный слева.   
При наведении в центр. При нажатии справа.

При нажатии на конкретную таблицу данных или тип даграммы, эта кнопка изменит цвет (рисунок 19) и пометится как выбранная.



Рисунок 19 - Стиль для выбранных кнопок-переключателей таблиц данных  
 и типов диаграмм

У таблиц данных есть дополнительный стиль, изображенный на рисунке 20. Он уведомляет пользователя о том, что данная таблица выбрана для построения диаграммы, но файл с данными на компьютере не найден, ввиду чего невозможно построение диаграммы.



Рисунок 20 - Стиль кнопки-переключателя таблиц данных, при отсутствии данных

Интерфейс страницы «Справка» сделан в виде пагинации страниц. На каждой странице можно найти информацию по каждой странице приложения: главная, данные, диаграммы, справка.

# Библиографический список

1. Официальный сайт JavaFX – https://openjfx.io/
2. Oracle Documentation - JavaFX - <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>
3. TutorialsPoint – JavaFX-Charts – <https://www.tutorialspoint.com/javafx/javafx_charts.htm>
4. Jenkov.com - JavaFX Tutorial - https://jenkov.com/tutorials/javafx/index.html

# Приложение (Исходный код программы)

## Класс DataTable

**public** **class** DataTable {

**private** List<Column> columns;

**private** List<Row> rows;

**private** File file;

**public** DataTable() {

columns = **new** ArrayList<>();

rows = **new** ArrayList<>();

}

**public** List<Column> getColumns() {

**return** columns;

}

**public** **void** setColumns(String... columnNames) {

**for** (String name : columnNames) {

**if** (!name.isEmpty()) {

**this**.addColumn(name);

}

}

}

**public** List<Row> getRows() {

**return** rows;

}

**public** List<Row> getRows(**int** start, **int** end) {

**return** rows.subList(start, end);

}

**public** **void** addRow(Row row) {

rows.add(row);

}

**public** **void** addRow(String... rowItems) {

rows.add(**new** Row(rowItems));

}

**public** **void** addColumn(String columnName) {

**this**.addColumn(**new** Column(columnName));

}

**public** **void** addColumn(Column column) {

columns.add(column);

}

**public** **void** addColumn(**int** index, Column column) {

columns.add(index, column);

}

**public** **void** removeColumn(**int** index) {

columns.remove(index);

}

@Override

**public** String toString() {

StringBuilder result = **new** StringBuilder();

**for** (Column col : columns) {

result.append(col.getName()).append("\t");

}

result.append("\n");

**for** (Row row : rows) {

result.append(row.toString()).append("\n");

}

result.append("\n");

**return** result.toString();

}

**public** File getFile() {

**return** file;

}

**public** **void** setFile(File file) {

**this**.file = file;

}

**public** **boolean** isLoaded() {

**return** columns.size() != 0;

}

**public** List<String> getColumnNames() {

List<String> result = **new** ArrayList<>();

**for** (Column col : **this**.getColumns()) {

result.add(col.getName());

}

**return** result;

}

}

## Класс Column

**public** **class** Column {

**private** String name;

**public** Column(String name) {

**this**.setName(name);

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

## Класс Row

**public** **class** Row {

**private** List<SimpleStringProperty> items;

**public** Row() {

items = **new** ArrayList<>();

}

**public** Row(String... items) {

**this**.items = **new** ArrayList<>();

**this**.setItems(items);

}

**public** List<SimpleStringProperty> getItems() {

**return** items;

}

**public** SimpleStringProperty getItem(**int** index) {

**return** items.get(index);

}

**public** **void** setItems(List<SimpleStringProperty> items) {

**this**.items = items;

}

**public** **void** setItems(String... items) {

**for** (String item : items) {

**this**.items.add(**new** SimpleStringProperty(item));

}

}

@Override

**public** String toString() {

StringBuilder result = **new** StringBuilder();

**for** (SimpleStringProperty item: items) {

result.append(item).append("\t");

}

result.append("\n");

**return** result.toString();

}

}

## Класс DataTableAdapter

Данный класс выполнен на основе шаблона проектирования Singleton для единообразного доступа к списку DataTable из различных контроллеров FXML-страниц.

**public** **class** DataTableAdapter {

**private** **static** DataTableAdapter *dataTableAdapter*;

**private** Map<String, DataTable> dataTables;

**private** DataTableAdapter() {

dataTables = **new** HashMap<>();

}

**public** **static** DataTableAdapter getInstance() {

**if** (*dataTableAdapter* == **null**) {

*dataTableAdapter* = **new** DataTableAdapter();

}

**return** *dataTableAdapter*;

}

**public** **void** addDataTable(String name, DataTable dataTable) {

**this**.dataTables.put(name, dataTable);

}

**public** **void** removeDataTable(String name) {

**this**.dataTables.remove(name);

}

**public** **void** changeDataTableName(String oldName, String newName) {

DataTable dataTable = **this**.getDataTable(oldName);

**this**.removeDataTable(oldName);

**this**.addDataTable(newName, dataTable);

}

**public** DataTable getDataTable(String name) {

DataTable dataTable = **this**.dataTables.get(name);

**if** (dataTable == **null**) {

**return** **null**;

}

**if** (!dataTable.isLoaded()) {

CSVImporter.*importCSV*(dataTable);

}

**return** dataTable;

}

**public** List<String> getAllDataTableNames(){

List<String> result = **new** ArrayList<>();

**for** (String key : **this**.dataTables.keySet()) {

result.add(key);

}

**return** result;

}

**public** **void** writeObject(StringBuilder outBuffer) {

**for** (String name : **this**.getAllDataTableNames()) {

outBuffer.append(name).append(";")

.append(**this**.getDataTable(name).getFile().getAbsolutePath()).append("\n");

}

}

**public** **void** readObject(List<String> lines) {

**for** (String line : lines) {

DataTable dataTable = **new** DataTable();

String[] fields = line.split(";");

File file = **new** File(fields[1]);

dataTable.setFile(file);

**this**.addDataTable(fields[0], dataTable);

}

}

}

## Класс CSVImporter

Данный класс импортирует данные из файла csv в DataTable.

**public** **class** CSVImporter {

**private** **static** DataTable *dataTable*;

**public** **static** DataTable importCSV(String path) {

*dataTable* = **new** DataTable();

*dataTable*.setFile(**new** File(path));

*importCSV*(*dataTable*);

**return** *dataTable*;

}

**public** **static** **void** importCSV(DataTable dataTable) {

**try** (BufferedReader reader = Files.*newBufferedReader*(Paths.*get*(dataTable.getFile().getAbsolutePath()))) {

String line;

**if** ((line = reader.readLine()) != **null** && !line.isEmpty()) {

line = line.strip().trim();

String[] headers = line.split(",");

dataTable.setColumns(headers);

}

**while** ((line = reader.readLine()) != **null** && !line.isEmpty()) {

line = line.strip().trim();

String[] fields = line.split(",");

dataTable.addRow(fields);

}

} **catch** (Exception e) {

System.***out***.println("File not exist - " + dataTable.getFile().getAbsolutePath());

//e.printStackTrace();

}

}

}

## Класс ScenesInitializator

Класс создан для инициализации FXML-файла в Scene единожды.

**public** **class** ScenesInitializator {

**private** ScenesInitializator() {

}

**private** **static** Scene *mainScene* = **null**;

**private** **static** Scene *dataScene* = **null**;

**private** **static** Scene *diagramScene* = **null**;

**private** **static** Scene *infoScene* = **null**;

**private** **static** Scene *predScene* = **null**;

**private** **static** MainController *mainController*;

**private** **static** DataController *dataController*;

**private** **static** DiagramController *diagramController*;

**public** **static** Scene getPredScene() {

Parent root;

**if** (*predScene* == **null**) {

**try** {

root = FXMLLoader.*load*(ScenesInitializator.**class**.getResource("PredPic.fxml"));

*predScene* = **new** Scene(root, 400, 400);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**return** *predScene*;

}

**public** **static** Scene getMainScene() {

**if** (*mainScene* == **null**) {

Parent root;

**try** {

FXMLLoader loader = **new** FXMLLoader(ScenesInitializator.**class**.getResource("MainMenu.fxml"));

root = loader.load();

*mainScene* = **new** Scene(root);

*mainScene*.getStylesheets().add(ScenesInitializator.**class**.getResource("application.css").toExternalForm());

*mainController* = loader.getController();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**return** *mainScene*;

}

**public** **static** Scene getDataScene() {

**if** (*dataScene* == **null**) {

Parent root;

**try** {

FXMLLoader loader = **new** FXMLLoader(ScenesInitializator.**class**.getResource("DataTables.fxml"));

root = loader.load();

*dataScene* = **new** Scene(root);

*dataScene*.getStylesheets().add(ScenesInitializator.**class**.getResource("application.css").toExternalForm());

*dataController* = loader.getController();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**return** *dataScene*;

}

**public** **static** Scene getDiagramScene() {

**if** (*diagramScene* == **null**) {

Parent root;

**try** {

FXMLLoader loader = **new** FXMLLoader(ScenesInitializator.**class**.getResource("Diagrams.fxml"));

root = loader.load();

*diagramScene* = **new** Scene(root);

*diagramScene*.getStylesheets().add(ScenesInitializator.**class**.getResource("application.css").toExternalForm());

*diagramController* = loader.getController();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**return** *diagramScene*;

}

**public** **static** Scene getInfoScene() {

**if** (*infoScene* == **null**) {

Parent root;

**try** {

FXMLLoader loader = **new** FXMLLoader(ScenesInitializator.**class**.getResource("HelpPage.fxml"));

root = loader.load();

*infoScene* = **new** Scene(root);

*infoScene*.getStylesheets().add(ScenesInitializator.**class**.getResource("application.css").toExternalForm());

//infoController = loader.getController();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**return** *infoScene*;

}

**public** **static** DataController getDataController() {

**if** (*dataController* == **null**) {

ScenesInitializator.*getDataScene*();

}

**return** *dataController*;

}

**public** **static** MainController getMainController() {

**if** (*mainController* == **null**) {

ScenesInitializator.*getMainScene*();

}

**return** *mainController*;

}

**public** **static** DiagramController getDiagramController() {

**if** (*diagramController* == **null**) {

ScenesInitializator.*getDiagramScene*();

}

**return** *diagramController*;

}

}

## Класс Utils

**public** **class** Utils {

**public** **static** String getHexColorString(Color color) {

**return** String.*format*("#%02X%02X%02X%02X", (**int**) (color.getRed() \* 255), (**int**) (color.getGreen() \* 255), (**int**) (color.getBlue() \* 255), (**int**) (color.getOpacity() \* 255));

}

}

## Перечисление Settings

**public** **enum** Settings {

***X\_AXIS***,

***Y\_AXIS***,

***COLOR***,

***ROWS***;

}

## Перечисление Diagrams

**public** **enum** Diagrams {

***LINE\_CHART***("LineChart") {

@Override

**public** **void** changeChartColor(Chart chart, Color color) {

String rgbColor = Utils.*getHexColorString*(color);

String style\_stroke = "-fx-stroke:" + rgbColor + ";";

**for** (XYChart.Series<?, ?> series : ((XYChart<?, ?>) chart).getData()) {

Node line = series.getNode().lookup(".chart-series-line");

**if** (line != **null**) {

line.setStyle(style\_stroke);

}

}

}

@Override

**public** Chart create(String tableName, DataTable dataTable, **int** xColumnIndex, **int** yColumnIndex, **int** rowStart,

**int** rowEnd) {

**return** DiagramCreator.*createLineDiagram*(tableName, dataTable, xColumnIndex, yColumnIndex, rowStart, rowEnd);

}

@Override

**public** Settings[] getSettings() {

Settings[] settings = {Settings.***X\_AXIS***, Settings.***Y\_AXIS***, Settings.***COLOR***, Settings.***ROWS***};

**return** settings;

}

},

***AREA\_CHART***("AreaChart") {

@Override

**public** **void** changeChartColor(Chart chart, Color color) {

String rgbStrokeColor = Utils.*getHexColorString*(color);

String rgbFillColor = Utils.*getHexColorString*(**new** Color(color.getRed(), color.getGreen(), color.getBlue(), 0.3));

String style\_stroke = "-fx-stroke:" + rgbStrokeColor + ";";

String style\_fill = "-fx-fill:" + rgbFillColor + ";";

**for** (XYChart.Series<?, ?> series : ((XYChart<?, ?>) chart).getData()) {

Node line = series.getNode().lookup(".chart-series-area-line");

Node area = series.getNode().lookup(".chart-series-area-fill");

**if** (line != **null**) {

line.setStyle(style\_stroke);

}

**if** (area != **null**) {

area.setStyle(style\_fill);

}

}

}

@Override

**public** Chart create(String tableName, DataTable dataTable, **int** xColumnIndex, **int** yColumnIndex, **int** rowStart,

**int** rowEnd) {

**return** DiagramCreator.*createAreaDiagram*(tableName, dataTable, xColumnIndex, yColumnIndex, rowStart, rowEnd);

}

@Override

**public** Settings[] getSettings() {

Settings[] settings = {Settings.***X\_AXIS***, Settings.***Y\_AXIS***, Settings.***COLOR***, Settings.***ROWS***};

**return** settings;

}

},

***BAR\_CHART***("BarChart") {

@Override

**public** **void** changeChartColor(Chart chart, Color color) {

String rgbColor = Utils.*getHexColorString*(color);

String style\_fill = "-fx-bar-fill:" + rgbColor + ";";

**for** (XYChart.Series<?, ?> series : ((XYChart<?, ?>) chart).getData()) {

**for** (Data<?, ?> data : series.getData()) {

data.getNode().setStyle(style\_fill);

}

}

}

@Override

**public** Chart create(String tableName, DataTable dataTable, **int** xColumnIndex, **int** yColumnIndex, **int** rowStart,

**int** rowEnd) {

**return** DiagramCreator.*createBarChart*(tableName, dataTable, xColumnIndex, yColumnIndex, rowStart, rowEnd);

}

@Override

**public** Settings[] getSettings() {

Settings[] settings = {Settings.***X\_AXIS***, Settings.***Y\_AXIS***, Settings.***COLOR***, Settings.***ROWS***};

**return** settings;

}

},

***SCATTER\_CHART***("ScatterChart") {

@Override

**public** **void** changeChartColor(Chart chart, Color color) {

String rgbColor = Utils.*getHexColorString*(color);

String style\_stroke = "-fx-background-color:" + rgbColor + ";";

**for** (XYChart.Series<?, ?> series : ((XYChart<?, ?>) chart).getData()) {

**for** (Data<?, ?> data : series.getData()) {

data.getNode().setStyle(style\_stroke);

}

}

}

@Override

**public** Chart create(String tableName, DataTable dataTable, **int** xColumnIndex, **int** yColumnIndex, **int** rowStart,

**int** rowEnd) {

**return** DiagramCreator.*createScatterChart*(tableName, dataTable, xColumnIndex, yColumnIndex, rowStart, rowEnd);

}

@Override

**public** Settings[] getSettings() {

Settings[] settings = {Settings.***X\_AXIS***, Settings.***Y\_AXIS***, Settings.***COLOR***, Settings.***ROWS***};

**return** settings;

}

},

***PIE\_CHART***("PieChart") {

@Override

**public** **void** changeChartColor(Chart chart, Color color) {

}

@Override

**public** Chart create(String tableName, DataTable dataTable, **int** xColumnIndex, **int** yColumnIndex, **int** rowStart,

**int** rowEnd) {

**return** DiagramCreator.*createPieChart*(tableName, dataTable, xColumnIndex, yColumnIndex, rowStart, rowEnd);

}

@Override

**public** Settings[] getSettings() {

Settings[] settings = {Settings.***X\_AXIS***, Settings.***Y\_AXIS***, Settings.***ROWS***};

**return** settings;

}

};

Diagrams(String name) {

**this**.setName(name);

}

**private** String name;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **static** Diagrams getDiagramByName(String name) {

**for** (Diagrams diagram : Diagrams.*values*()) {

**if** (diagram.getName().equals(name)) {

**return** diagram;

}

}

**return** ***LINE\_CHART***;

}

**public** **abstract** **void** changeChartColor(Chart chart, Color color);

**public** **abstract** Chart create(String tableName, DataTable dataTable, **int** xColumnIndex, **int** yColumnIndex, **int** rowStart, **int** rowEnd);

**public** **abstract** Settings[] getSettings();

}