Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение информационных систем

и технологий»

**Курсовая работа**

по дисциплине ***«*Программирование на языке Java*»***

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнитель: |  | студент группы 10702419  Кузьмицкая Мария Васильевна | |
| Преподаватель: |  | | доц. Сидорик Валерий Владимирович |

Минск 2022

Оглавление

[ВВЕДЕНИЕ 3](#_Toc103239493)

[1 Постановка и описание задачи 4](#_Toc103239494)

[2 Обзор предметной области 5](#_Toc103239495)

[2.1 Хранение данных 5](#_Toc103239496)

[2.2 Подключение к базе данных 6](#_Toc103239497)

[3 Построение математической модели 7](#_Toc103239498)

[4 Реализация 8](#_Toc103239499)

[4.1 Структура программы 8](#_Toc103239500)

[4.2 Описание элементов программы 9](#_Toc103239501)

[4.2 UML диаграммы классов 16](#_Toc103239502)

[4.4 Собрание jar-файла 17](#_Toc103239503)

[Приложение A 18](#_Toc103239504)

[Приложение B 21](#_Toc103239505)

[Список используемой литературы: 61](#_Toc103239506)

# ВВЕДЕНИЕ

Большинству учителей и преподавателей по роду деятельности приходится вести списки студентов и контролировать посещаемость лекций.

Для удобства отслеживания пропусков лекций было создано приложение «Посещаемость лекционных занятий».

В данном программном документе приведено описание программы «Посещаемость лекционных занятий», предназначенной для формирования и использования базы данных PostgreSQL. Заложенные в программу функциональные возможности позволяют решать следующие задачи:

· добавление новых записей в базу;

· редактирование занесенных данных;

· удаление записей;

Целью данного проекта является упрощение повседневной и рутинной работы преподавателей.

## 1 Постановка и описание задачи

*Цель программы*: осуществить удобное взаимодействие пользователя с созданной базой данных PostgreSQL.

*Тема курсовой работы:* создать приложение для отслеживания посещаемости лекций. Учитываются ФИО, группа, даты посещения занятий. Операции добавления даты посещения. Вывод списка студентов, посещавших занятия по заданной дате. Добавление и удаление групп. Сортировка по количеству посещений и по фамилии. Формирование отчета пропусков. Вывод на печать.

# 2 Обзор предметной области

## 2.1 Хранение данных

В данном проекте для хранения данных использовалась реляционная база данных PostgreSQL.

PostgreSQL — это свободная объектно-реляционная система управления базами данных.

PostgreSQL создана на основе некоммерческой СУБД Postgres, разработанной как open-source.

Для взаимодействия с PostgreSQL я использовала современный драйвер JDBC.

## Подключение к базе данных

Для соединения с БД необходимо использовать класс Connection пакета java.sql. После установления соединения можно выполнять различные SQL-запросы и получать результаты их обработки сервером. Приложение может открыть одно или несколько соединений с одной или несколькими различными СУБД.

Стандартный способ установления соединения — это вызов метода getConnection класса DriverManager. Методу getConnection необходимо передать строку URL (Uniform Resource Locator). Стандарт подключения к серверу базы данных позволяет использовать следующие методы getConnection с параметрами:

getConnection(url);

getConnection(url, properties);

getConnection(url, username, password);

# 3 Построение математической модели

В приложении имеются:

* Основное окно (Main Window), в котором представлен просмотр и взаимодействие с информацией, которая хранится в базе данных.
* Стартовое окно, окно с информацией о программе и окно с информацией об авторе

Проект представлен блок-схемой или, как ее по другому называют, бизнес-логикой переключения окон проекта, состоящая из 4 окон взаимосвязанных между собой: Splash Screen (стартовое окно приложения), Main Window (основное окно для работы с приложением), About program (о программе) и About Author (об авторе) (рис. 3.1.):

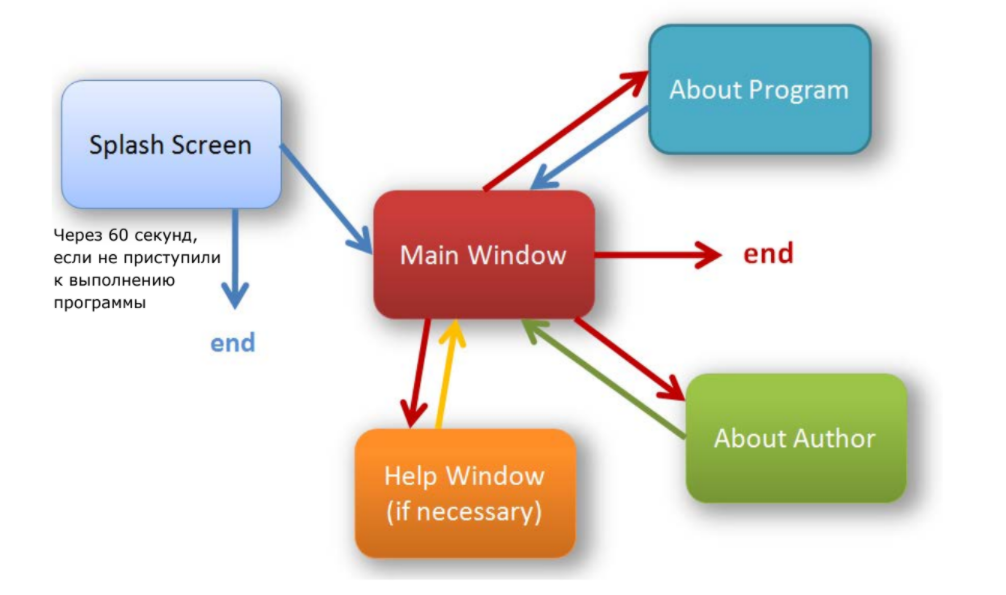


Рисунок 3.1 – Бизнес-логика переключения графических окон.

# 4 Реализация

## 4.1 Структура программы

Структура программы была создана по примеру из файла, представленного в списке литературы под номером 3 (рис. 4.1):

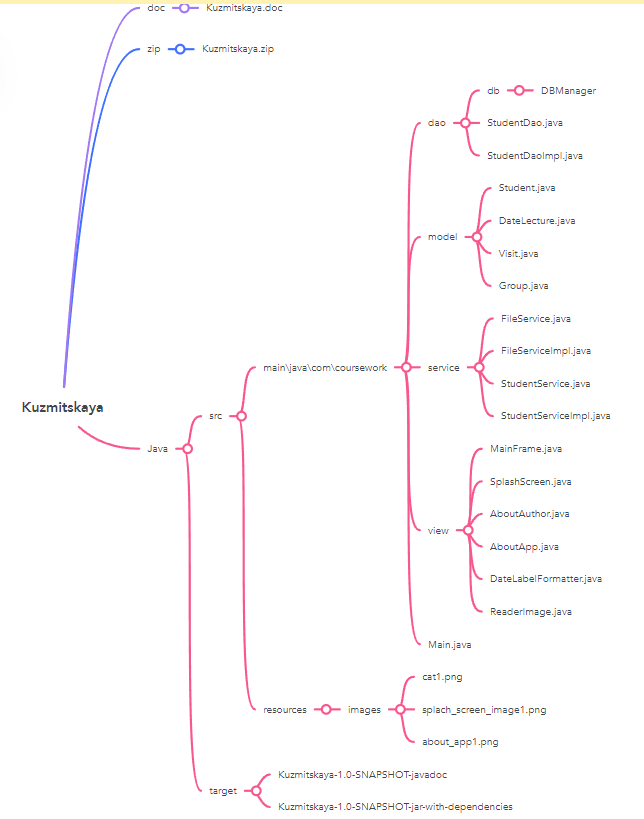


Рисунок 4.1

## Описание элементов программы

Код данной программы находится см. в Приложении B.

Программа состоит из 15 классов:

1. DBManager
2. StudentDaoImpl (имплементирован от класса StudentDao)
3. DateLecture
4. Group
5. Student
6. Visit
7. FileServiceImpl (имплементирован от класса FileService)
8. StudentServiceImpl (имплементирован от класса StudentService)
9. AboutApp
10. AboutAuthor
11. DateLabelFormatter
12. MainFrame
13. ReaderImage
14. SplashScreen
15. Main

Main

Класс Main является классом запуска описываемого приложения. В данном классе мы создаем окно приложения (рисунок 5.1, приложение A), архитектура которого прописана в классе SplashScreen и делаем это окно видимым.

SplashScreen

Класс для стартового окна (рис. 5.1).

В данном классе описываются следующие элементы:

* start – кнопка отвечающая за открытие окна приложения MainFrame (рисунок 5.2, приложение A).
* exit – кнопка отвечающая за закрытия приложения.

В случае бездействия пользователя в течении 60 секунд приложение автоматически закроется.

MainFrame

Класс MainFrame - класс контроллер окна MainFrame (рисунок 5.2, приложение A). В данном классе описываются следующие элементы:

* studentsList – панель с таблицей студентов.
* groupsButtons – панель с кнопками для выбора группы.
* author – пункт меню, после нажатия которого открывается окно AboutAuthor (рисунок 5.3, приложение A).
* app – пункт меню, после нажатия которого открывается окно AboutApp (рисунок 5.4, приложение A).
* fileButton – пункт меню, после нажатия которого открывается окно для сохранения данных таблицы в excel файл.
* dateOfVisit – панель для создания даты посещения.
* studentsByDate – панель для вывода студентов по дате посещения.
* groupsModification – панель для добавления/удаления группы.
* studentsModification – панель для добавления/удаления студентов

Методы класса:

* createPanelForAddDeleteStudent – метод для создания панели для добавления/удаления студента.
* checkFullStudentName – метод для проверки имени студента на валидность.
* createPanelForAddDeleteGroup – метод для создания панели для добавления/удаления студента.
* checkGroup – при добавлении новой группы этот метод проверят, существует ли уже такая группа.
* updateGroupsButtons – при добавлении/удалении групп этот метод обновляет отображения кнопок с группами.
* createPanelForGetStudentsByDate – метод, создающий панель для задания даты при поиске студентов по дате.
* createPanelForAddDate – метод для создания панели для добавления даты посещения.
* createDatePicker – метод для создания календаря.
* addGroupButton – метод для добавления кнопки в группу кнопок для групп студентов.
* createButtonsForGroups – создание группы кнопок для групп студентов.
* createStudentsPane – создание таблицы студентов
* createListOfDates – создание данных для таблицы
* createListOfDatesForHeader – создание данных для заголовка таблицы
* createStudentsList – создание списка студентов
* createHeaderTable – заполнение заголовка таблицы данными
* createTableData – заполнение таблицы данными
* checkDate – проверка на существование даты

ReaderImage

Класс ReaderImage– класс для получения иконки из ресурсов.

Методы класса:

* scaleImage – возвращает иконку с заданным масштабом.

DateLabelFormatter

Класс DateLabelFormatter – класс для форматирования даты, которая используется в элементе JDatePicker.

Group

Класс Group– класс сущность, описывающий основные характеристики группы. В данном классе имеются следующие поля:

* groupId – id группы в базе данных.
* groupNumber – номер группы

Имеются get и set методы для атрибутов класса.

Student

Класс Student– класс сущность, описывающий основные характеристики студента. В данном классе имеются следующие поля:

* groupID – id группы, где учится студент.
* studentId – id студента.
* name – имя студента.
* lastName – фамилия студента.
* patronymic – отчество студента.

Имеются get и set методы для атрибутов класса.

Visit

Класс Visit – класс сущность, содержащий в себе информацию о посещаемых лекциях. В данном классе имеются следующие поля:

* id – id посещения.
* studentId – id студента.
* dateId – id даты лекции.

Имеются get и set методы для атрибутов класса.

Visit

Класс Visit – класс сущность, содержащий в себе информацию о посещаемых лекциях. В данном классе имеются следующие поля:

* id – id посещения.
* studentId – id студента.
* dateId – id даты лекции.

Имеются get и set методы для атрибутов класса.

DateLecture

Класс DateLecture – класс сущность, содержащий в себе информацию о датах проводящихся лекциях. В данном классе имеются следующие поля:

* id – id даты.
* date – дата лекции.

Имеются get и set методы для атрибутов класса.

DBManager

Класс DBManager – класс для установления соединения с базой данных. В данном классе имеются следующие поля:

* URL – url для подключения к базе данных.
* USER – имя пользователя.
* PASSWORD – пароль пользователя.

Имеются get и set методы для атрибутов класса.

StudentdaoImpl

Класс StudentsDaoImpl – класс прослойка между базой данных и приложением.

Методы класса:

* addDate – добавление даты лекции.
* addStudent – добавление студента.
* getStudentsByDate – получение студентов по дате.
* addGroup – добавление группы.
* deleteGroup – удаление группы.
* getGroup – получение группы.
* addVisit – добавление посещения.
* parseDate – метод для разбора форматы даты.
* getAllGroups – получение всех групп.
* getStudentsByGroup – получение всех студентов по группе.
* getAllDates – получение всех дат лекций.
* getDatesByStudent – получение всех посещений студента.
* deleteVisit – удаление посещения.
* getGroupById – получение группы по id.
* deleteStudent – удаление студента.
* getStudentByName – получение студента по имени.

FileServiceImpl

Класс FileServiceImpl – класс предназначенный для работы с excel.

Методы класса:

* createFile – метод для создания excel таблицы.
* writeDataToExcel– метод для заполнения данными файла.
* setHeader– формирование заголовка таблицы.
* setTableData – формирование данных таблицы.
* checkDate – проверка посещаемости.

StudentServiceImpl

Класс StudentServiceImpl – класс предназначен для обеспечения логики работы с данными, отправляемыми в DAO и из клиента.

AboutAuthor

Класс AboutAuthor – класс контроллер окна AboutAuthor (рис. 5.3).

Содержит поля:

* icon – поле с иконкой.
* author – поле с информацией об авторе.
* back – кнопка для возвращения на главное окно.

AboutApp

Класс AboutApp– класс контроллер окна AboutApp (рис. 5.2).

Содержит поля:

* icon – поле с иконкой.
* label – поле с информацией о приложении.
* back – кнопка для возвращения на главное окно.

## UML диаграммы классов

Изображение выглядит как текст, карта, внутренний

Автоматически созданное описание

Рисунок 4.2Утилита Javadoc

При помощи утилиты javadoc в папке создался файл index.html (рис. 4):

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 4.3

## 4.4 Собрание jar-файла

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 4.4

# Приложение A

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5.1

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 5.2

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5.3

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5.4

# Приложение B

Main.java

**package** com.coursework;  
  
**import** com.coursework.view.SplashScreen;  
  
*/\*\*  
 \* Class for launching the application  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** Main {  
 **public static void** main(String[] args) {  
 **new** SplashScreen();  
 }  
}

SplashScreen.java

**package** com.coursework.view;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
*/\*\*  
 \* Splash screen window  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** SplashScreen **extends** JFrame{  
  
 **private final** JFrame **frame**;  
 **private** Container **container**;  
 **private** JPanel **universityPanel**;  
 **private** JPanel **facultyPanel**;  
 **private** JPanel **courseWorkPanel**;  
 **private** JPanel **courseWorkInformation**;  
 **private** JPanel **information**;  
 **private** JPanel **authorInformation**;  
 **private** JPanel **teacherInformation**;  
 **private** JPanel **iconPanel**;  
 **private** JPanel **buttons**;  
 **private** JLabel **university**;  
 **private** JLabel **faculty**;  
 **private** JLabel **department**;  
 **private** JLabel **courseWork**;  
 **private** JLabel **discipline**;  
 **private** JLabel **theme**;  
 **private** JLabel **author**;  
 **private** JLabel **authorName**;  
 **private** JLabel **teacherName**;  
 **private** JLabel **teacher**;  
 **private** JLabel **minsk2022**;  
 **private** ReaderImage **readerImage**;  
 **private** JLabel **imageLabel**;  
 **private** Icon **icon**;  
 **private** JButton **start**;  
 **private** JButton **exit**;  
 **private** Timer **timer**;  
  
 */\*\*  
 \* Constructor  
 \*/* **public** SplashScreen(){  
 **frame** = **new** JFrame(**"Lectures"**);  
 **frame**.setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);  
 **frame**.setResizable(**false**);  
 **frame**.setPreferredSize(**new** Dimension(1100, 720));  
 **frame**.setLocation(400, 100);  
  
 **container** = **new** Container();  
 **container**.setLayout(**new** BoxLayout(**container**, BoxLayout.***Y\_AXIS***));  
  
 **universityPanel** = **new** JPanel();  
 **universityPanel**.setMaximumSize(**new** Dimension(1100, 45));  
 **universityPanel**.setMinimumSize(**new** Dimension(1100, 45));  
  
 **facultyPanel** = **new** JPanel();  
 **facultyPanel**.setMaximumSize(**new** Dimension(1100, 150));  
 **facultyPanel**.setMinimumSize(**new** Dimension(1100, 150));  
  
 **courseWorkPanel** = **new** JPanel();  
 **courseWorkPanel**.setLayout(**new** BoxLayout(**courseWorkPanel**, BoxLayout.***Y\_AXIS***));  
 **courseWorkPanel**.setMaximumSize(**new** Dimension(1100, 170));  
 **courseWorkPanel**.setMinimumSize(**new** Dimension(1100, 170));  
  
 **courseWorkInformation** = **new** JPanel();  
 **courseWorkInformation**.setLayout(**new** BoxLayout(**courseWorkInformation**, BoxLayout.***X\_AXIS***));  
 **courseWorkInformation**.setMaximumSize(**new** Dimension(1100, 190));  
 **courseWorkInformation**.setMinimumSize(**new** Dimension(1100, 190));  
  
 **information** = **new** JPanel();  
 **information**.setLayout(**new** BoxLayout(**information**, BoxLayout.***Y\_AXIS***));  
 **information**.setMaximumSize(**new** Dimension(550, 190));  
 **information**.setMinimumSize(**new** Dimension(550, 190));  
  
 **authorInformation** = **new** JPanel();  
 **authorInformation**.setLayout(**new** BoxLayout(**authorInformation**, BoxLayout.***Y\_AXIS***));  
 **authorInformation**.setMaximumSize(**new** Dimension(550, 95));  
 **authorInformation**.setMinimumSize(**new** Dimension(550, 95));  
  
 **teacherInformation** = **new** JPanel();  
 **teacherInformation**.setLayout(**new** BoxLayout(**teacherInformation**, BoxLayout.***Y\_AXIS***));  
 **teacherInformation**.setMaximumSize(**new** Dimension(550, 95));  
 **teacherInformation**.setMinimumSize(**new** Dimension(550, 95));  
  
 **iconPanel** = **new** JPanel();  
 **iconPanel**.setMaximumSize(**new** Dimension(550, 190));  
 **iconPanel**.setMinimumSize(**new** Dimension(550, 190));  
  
 **buttons** = **new** JPanel();  
 **buttons**.setLayout(**new** BoxLayout(**buttons**, BoxLayout.***X\_AXIS***));  
  
 **university** = **new** JLabel();  
 **university**.setText(**"Белорусский национальный технический университет"**);  
 **university**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **university**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 24));  
  
 **faculty** = **new** JLabel();  
 **faculty**.setText(**"Факультет информационных технологий и робототехники"**);  
 **faculty**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **faculty**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 20));  
  
 **department** = **new** JLabel();  
 **department**.setText(**"Кафедра программного обеспечения информационных систем и технологий"**);  
 **department**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **department**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 20));  
  
 **courseWork** = **new** JLabel();  
 **courseWork**.setText(**"Курсовая работа"**);  
 **courseWork**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **courseWork**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 30));  
  
 **discipline** = **new** JLabel();  
 **discipline**.setText(**"по дисциплине «Программирование на языке Java»"**);  
 **discipline**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **discipline**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **theme** = **new** JLabel();  
 **theme**.setText(**"Посещаемость лекционных занятий"**);  
 **theme**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **theme**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 40));  
  
 **author** = **new** JLabel();  
 **author**.setText(**"Выполнила: студентка группы 10702419"**);  
 **author**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 20));  
  
 **authorName** = **new** JLabel();  
 **authorName**.setText(**"Кузьмицкая Мария Васильевна"**);  
 **authorName**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 20));  
  
 **teacherName** = **new** JLabel();  
 **teacherName**.setText(**"Сидорик Валерий Владимирович"**);  
 **teacherName**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 20));  
  
 **teacher** = **new** JLabel();  
 **teacher**.setText(**"Преподаватель: к.ф.-м.н.,доц."**);  
 **teacher**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 20));  
  
 **minsk2022** = **new** JLabel();  
 **minsk2022**.setText(**"Минск, 2022"**);  
 **minsk2022**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **minsk2022**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **readerImage** = **new** ReaderImage();  
 **icon** = **readerImage**.scaleImage(**"images/splash\_screen\_image1.png"**, 150, 146);  
  
 **imageLabel** = **new** JLabel();  
 **imageLabel**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **imageLabel**.setIcon(**icon**);  
  
 **start** = **new** JButton(**"Далее"**);  
 **start**.addActionListener(e -> {  
 **frame**.dispose();  
 **timer**.stop();  
 **new** MainFrame();  
 });  
 **start**.setMinimumSize(**new** Dimension(100,40));  
 **start**.setMaximumSize(**new** Dimension(100,40));  
 **exit** = **new** JButton(**"Выход"**);  
 **exit**.addActionListener(e -> System.*exit*(0));  
 **exit**.setMinimumSize(**new** Dimension(100,40));  
 **exit**.setMaximumSize(**new** Dimension(100,40));  
  
 **timer** = **new** Timer(60000, e -> System.*exit*(0));  
 **timer**.setRepeats(**false**);  
 **timer**.start();  
  
 **universityPanel**.add(**university**);  
 **facultyPanel**.add(**faculty**);  
 **facultyPanel**.add(**department**);  
 **courseWorkPanel**.add(**courseWork**);  
 **courseWorkPanel**.add(**discipline**);  
 **courseWorkPanel**.add(**theme**);  
 **authorInformation**.add(**author**);  
 **authorInformation**.add(**authorName**);  
 **teacherInformation**.add(**teacher**);  
 **teacherInformation**.add(**teacherName**);  
 **information**.add(**authorInformation**);  
 **information**.add(**teacherInformation**);  
 **iconPanel**.add(**imageLabel**);  
 **courseWorkInformation**.add(**iconPanel**);  
 **courseWorkInformation**.add(**information**);  
 **buttons**.add(**start**);  
 **buttons**.add(**exit**);  
  
 **container**.add(**universityPanel**);  
 **container**.add(**facultyPanel**);  
 **container**.add(**courseWorkPanel**);  
 **container**.add(**courseWorkInformation**);  
 **container**.add(**minsk2022**);  
 **container**.add(**buttons**);  
  
 **frame**.setContentPane(**container**);  
 **frame**.pack();  
 **frame**.setVisible(**true**);  
 }  
}

ReaderImage.java

**package** com.coursework.view;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.util.Objects;  
  
*/\*\*  
 \* Class for adding an image  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** ReaderImage {  
 */\*\*  
 \* Method for getting an image  
 \** ***@param path*** *path  
 \** ***@param sizeX*** *size x  
 \** ***@param sizeY*** *size y  
 \** ***@return*** *image  
 \*/* **public** ImageIcon scaleImage(String path, **int** sizeX, **int** sizeY) {  
 ClassLoader classLoader = Thread.*currentThread*().getContextClassLoader();  
 ImageIcon icon = **new** ImageIcon(Objects.*requireNonNull*(classLoader.getResource(**"resources/"** + path)));  
 Image newImageIcon = icon.getImage().getScaledInstance(sizeX,sizeY,Image.***SCALE\_DEFAULT***);  
 icon.setImage(newImageIcon);  
 **return** icon;  
 }  
}

MainFrame.java

**package** com.coursework.view;  
  
**import** com.coursework.model.DateLecture;  
**import** com.coursework.model.Group;  
**import** com.coursework.model.Student;  
**import** com.coursework.model.Visit;  
**import** com.coursework.service.FileService;  
**import** com.coursework.service.FileServiceImpl;  
**import** com.coursework.service.StudentService;  
**import** com.coursework.service.StudentServiceImpl;  
**import** org.jdatepicker.impl.JDatePanelImpl;  
**import** org.jdatepicker.impl.JDatePickerImpl;  
**import** org.jdatepicker.impl.UtilDateModel;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.text.ParseException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.\*;  
**import** java.util.List;  
  
*/\*\*  
 \* Main window  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** MainFrame **extends** JFrame {  
  
 **private final** FileService **fileService**;  
 **private final** StudentService **studentService**;  
 **private** Group **currentGroup**;  
 **private** Container **container**;  
 **private final** JFrame **frame**;  
 **private** JPanel **contents**;  
 **private** JPanel **studentsList**;  
 **private** JPanel **groupsButtons**;  
 **private** JPanel **modification**;  
 **private** JPanel **dateOfVisit**;  
 **private** JPanel **studentsByDate**;  
 **private** JPanel **groupsModification**;  
 **private** JPanel **studentsModification**;  
 **private** JPanel **filePanel**;  
 **private** JButton **fileButton**;  
 **private** JButton **author**;  
 **private** JButton **app**;  
  
 */\*\*  
 \* Constructor  
 \*/* **public** MainFrame() {  
 **fileService** = FileServiceImpl.*getInstance*();  
 **studentService** = StudentServiceImpl.*getInstance*();  
  
 **container** = **new** Container();  
 **container**.setLayout(**new** BoxLayout(**container**, BoxLayout.***X\_AXIS***));  
  
 **frame** = **new** JFrame(**"Lectures"**);  
 **frame**.setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);  
 **frame**.setPreferredSize(**new** Dimension(1100, 900));  
 **frame**.setResizable(**false**);  
 **frame**.setLocation(400, 100);  
  
 **contents** = **new** JPanel();  
 **contents**.setLayout(**new** BoxLayout(**contents**, BoxLayout.***Y\_AXIS***));  
  
 **studentsList** = **new** JPanel();  
 **studentsList**.setSize(**new** Dimension(700, 700));  
  
 **groupsButtons** = **new** JPanel();  
 **groupsButtons**.setMinimumSize(**new** Dimension(700, 100));  
 **groupsButtons**.setMaximumSize(**new** Dimension(700, 100));  
  
 **modification** = **new** JPanel();  
 **modification**.setLayout(**new** BoxLayout(**modification**, BoxLayout.***Y\_AXIS***));  
 **modification**.setMinimumSize(**new** Dimension(400, 700));  
 **modification**.setMaximumSize(**new** Dimension(400, 700));  
  
 **dateOfVisit** = **new** JPanel();  
 **dateOfVisit**.setLayout(**new** FlowLayout(FlowLayout.***CENTER***));  
 **dateOfVisit**.setBorder(BorderFactory.*createTitledBorder*(**"Добавить дату посещения"**));  
 **dateOfVisit**.setPreferredSize(**new** Dimension(350, 150));  
  
 **studentsByDate** = **new** JPanel();  
 **studentsByDate**.setLayout(**new** FlowLayout(FlowLayout.***CENTER***));  
 **studentsByDate**.setBorder(BorderFactory.*createTitledBorder*(**"Список студентов по дате посещения"**));  
 **studentsByDate**.setPreferredSize(**new** Dimension(350, 150));  
  
 **groupsModification** = **new** JPanel();  
 **groupsModification**.setLayout(**new** FlowLayout(FlowLayout.***CENTER***));  
 **groupsModification**.setBorder(BorderFactory.*createTitledBorder*(**"Добавление/Удаление группы"**));  
 **groupsModification**.setPreferredSize(**new** Dimension(350, 150));  
  
 **studentsModification** = **new** JPanel();  
 **studentsModification**.setLayout(**new** FlowLayout(FlowLayout.***CENTER***));  
 **studentsModification**.setBorder(BorderFactory.*createTitledBorder*(**"Добавление/Удаление студента"**));  
 **studentsModification**.setPreferredSize(**new** Dimension(350, 150));  
  
 **filePanel** = **new** JPanel();  
 **fileButton** = **new** JButton(**"Сохранить в файл"**);  
 **fileButton**.addActionListener(e -> {  
 **if** (**currentGroup** != **null**) {  
 File file = **fileService**.createFile(**studentsList**);  
 **try** {  
 **fileService**.writeDataToExcel(file, **studentService**.getStudentsByGroup(**currentGroup**), **currentGroup**.getGroupNumber());  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 }  
 } **else** {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не выбрали группу!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 }  
 });  
 **author** = **new** JButton(**"Об авторе"**);  
 **author**.addActionListener(e -> {  
 **frame**.dispose();  
 **new** AboutAuthor();  
 });  
 **app** = **new** JButton(**"О программе"**);  
 **app**.addActionListener(e -> {  
 **frame**.dispose();  
 **new** AboutApp();  
 });  
  
 **filePanel**.add(**fileButton**);  
 **filePanel**.add(**author**);  
 **filePanel**.add(**app**);  
  
 createPanelForAddDeleteStudent(**studentsModification**);  
 createPanelForAddDeleteGroup(**groupsModification**, **groupsButtons**, **studentsList**);  
 createPanelForAddDate(**dateOfVisit**);  
 createButtonsForGroups(**groupsButtons**, **studentsList**);  
 createPanelForGetStudentsByDate(**studentsByDate**, **studentsList**);  
  
 **contents**.add(**groupsButtons**);  
  
 **contents**.add(**studentsList**);  
 **contents**.add(**filePanel**);  
  
 **container**.add(**contents**);  
 **modification**.add(**dateOfVisit**);  
 **modification**.add(**studentsByDate**);  
 **modification**.add(**groupsModification**);  
 **modification**.add(**studentsModification**);  
 **container**.add(**modification**);  
  
 **frame**.setContentPane(**container**);  
 **frame**.pack();  
 **frame**.setVisible(**true**);  
 }  
  
 **private void** createPanelForAddDeleteStudent(Container studentModification) {  
 JLabel enterStudentName = **new** JLabel(**"Введите ФИО:"**);  
 JTextField studentsName = **new** JTextField(25);  
 studentModification.add(enterStudentName);  
 studentModification.add(studentsName);  
 JButton addStudentButton = **new** JButton(**"Добавить"**);  
 JButton deleteStudentButton = **new** JButton(**"Удалить"**);  
 addStudentButton.addActionListener(e -> {  
 **if** (studentsName.getText().isEmpty()) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не заполнили поле!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else if** (checkFullStudentName(studentsName.getText()) == **null**) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы ввели некорректное ФИО студента!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else if** (**currentGroup** == **null**) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не выбрали группу!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else** {  
 String studentName = studentsName.getText();  
 Student student = checkFullStudentName(studentName);  
 **studentService**.addStudent(student, **currentGroup**);  
 }  
 });  
 deleteStudentButton.addActionListener(e ->  
 {  
 **if** (studentsName.getText().isEmpty()) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не заполнили поле!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else if** (checkFullStudentName(studentsName.getText()) == **null**) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы ввели некорректное ФИО студента!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else if** (**currentGroup** == **null**) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не выбрали группу!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else** {  
 String studentName = studentsName.getText();  
 Student student = checkFullStudentName(studentName);  
 **if** (student == **null**) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы ввели некорректное имя!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else** {  
 student.setGroupId(**studentService**.getGroup(**currentGroup**.getGroupNumber()).getId());  
 }  
 **if** (**studentService**.getStudentByName(student) != **null**) {  
 Student studentDelete = **studentService**.getStudentByName(student);  
 **studentService**.deleteStudent(studentDelete);  
 } **else** {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Такого студента не существует!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 }  
  
 }  
 });  
 studentModification.add(addStudentButton);  
 studentModification.add(deleteStudentButton);  
 }  
  
 **private** Student checkFullStudentName(String studentName) {  
 **try** {  
 Student student = **new** Student();  
 student.setStudentLastname(studentName.substring(0, studentName.indexOf(**" "**)));  
 studentName = studentName.substring(studentName.indexOf(**" "**) + 1);  
 student.setStudentName(studentName.substring(0, studentName.indexOf(**" "**)));  
 studentName = studentName.substring(studentName.indexOf(**" "**) + 1);  
 student.setStudentPatronymic(studentName);  
 **return** student;  
 } **catch** (Exception exception) {  
 **return null**;  
 }  
 }  
  
 **private void** createPanelForAddDeleteGroup(Container groupModification, Container groupsButtons, Container studentsList) {  
 JLabel enterGroupNumber = **new** JLabel(**"Введите номер группы:"**);  
 JTextField groupField = **new** JTextField(25);  
 groupModification.add(enterGroupNumber);  
 groupModification.add(groupField);  
 JButton addGroupButton = **new** JButton(**"Добавить"**);  
 JButton deleteGroupButton = **new** JButton(**"Удалить"**);  
 addGroupButton.addActionListener(e -> {  
 **if** (groupField.getText().isEmpty()) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не заполнили поле!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else if** (!checkGroup(groupField.getText())) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Такая группа уже существует!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else** {  
 Group newGroup = **new** Group();  
 newGroup.setGroupNumber(groupField.getText());  
 **studentService**.addGroup(newGroup);  
 updateGroupsButtons(groupsButtons, studentsList);  
 }  
 });  
 deleteGroupButton.addActionListener(e -> {  
 **if** (groupField.getText().isEmpty()) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не заполнили поле!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else if** (**studentService**.getGroup(groupField.getText()) == **null**) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Такой группы не существует!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else** {  
 **studentService**.deleteGroup(groupField.getText());  
 updateGroupsButtons(groupsButtons, studentsList);  
 }  
 });  
 groupModification.add(addGroupButton);  
 groupModification.add(deleteGroupButton);  
 }  
  
 **private boolean** checkGroup(String newGroup) {  
 **for** (Group groups : **studentService**.getAllGroups()) {  
 **if** (newGroup.equals(groups.getGroupNumber())) {  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
  
 **private void** updateGroupsButtons(Container groupsButtons, Container studentsList) {  
 groupsButtons.removeAll();  
 List<Group> allGroups = **studentService**.getAllGroups();  
 ButtonGroup buttonGroup = **new** ButtonGroup();  
 **for** (Group group : allGroups) {  
 addGroupButton(group.getGroupNumber(), groupsButtons, studentsList, buttonGroup);  
 }  
 groupsButtons.repaint();  
 groupsButtons.revalidate();  
 }  
  
 **private void** createPanelForGetStudentsByDate(Container studentsByDate, Container studentsList) {  
 JDatePickerImpl datePicker = createDatePicker();  
 JButton getStudentsByDateButton = **new** JButton(**"Поиск"**);  
 getStudentsByDateButton.addActionListener(e -> {  
 SimpleDateFormat sdf = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
 DateLecture date = **new** DateLecture();  
 **if** (datePicker.getModel().getValue() != **null**) {  
 date.setDate(sdf.format(datePicker.getModel().getValue()));  
 List<Student> studentList = **studentService**.getStudentsByDate(date, **currentGroup**);  
 String[] studentsName = **new** String[studentList.size()];  
 **int** i = 0;  
 **for** (Student student : studentList) {  
 studentsName[i] = student.getStudentLastname() + **" "** + student.getStudentName() + **" "** + student.getStudentPatronymic();  
 i++;  
 }  
 JList<String> students = **new** JList<>(studentsName);  
 JScrollPane scrollPane = **new** JScrollPane(students);  
 scrollPane.setHorizontalScrollBarPolicy(JScrollPane.***HORIZONTAL\_SCROLLBAR\_AS\_NEEDED***);  
 scrollPane.setVerticalScrollBarPolicy(JScrollPane.***VERTICAL\_SCROLLBAR\_AS\_NEEDED***);  
 scrollPane.setPreferredSize(**new** Dimension(600, 600));  
 scrollPane.setMaximumSize(scrollPane.getSize());  
 scrollPane.setMinimumSize(scrollPane.getSize());  
 studentsList.removeAll();  
 studentsList.add(scrollPane);  
 studentsList.repaint();  
 studentsList.revalidate();  
 } **else** {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не выбрали дату!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 }  
 });  
 studentsByDate.add(datePicker);  
 studentsByDate.add(getStudentsByDateButton);  
 }  
  
 **private void** createPanelForAddDate(Container dateOfVisit) {  
 JDatePickerImpl datePicker = createDatePicker();  
 dateOfVisit.add(datePicker);  
 JButton addDateButton = **new** JButton(**"Добавить"**);  
 addDateButton.addActionListener(e -> {  
 SimpleDateFormat sdf = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
 DateLecture date = **new** DateLecture();  
 **if** (datePicker.getModel().getValue() != **null**) {  
 date.setDate(sdf.format(datePicker.getModel().getValue()));  
 **studentService**.addDate(date);  
 } **else** {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Вы не выбрали дату!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 }  
 });  
 dateOfVisit.add(addDateButton);  
 }  
  
 **private** JDatePickerImpl createDatePicker() {  
 UtilDateModel model = **new** UtilDateModel();  
 Properties p = **new** Properties();  
 p.put(**"text.today"**, **"Today"**);  
 p.put(**"text.month"**, **"Month"**);  
 p.put(**"text.year"**, **"Year"**);  
 JDatePanelImpl datePanel = **new** JDatePanelImpl(model, p);  
 **return new** JDatePickerImpl(datePanel, **new** DateLabelFormatter());  
 }  
  
 **private void** addGroupButton(String caption, Container containerButton, Container containerStudents, ButtonGroup buttonGroup) {  
 JButton button = **new** JButton(caption);  
 button.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 button.setAlignmentY(Component.***TOP\_ALIGNMENT***);  
 button.addActionListener(e -> {  
 String groupNumber = button.getText();  
 Group group = **studentService**.getGroup(groupNumber);  
 **try** {  
 createStudentsPane(containerStudents, group);  
 } **catch** (ParseException ex) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Что-то пошло не так..."**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 }  
 **currentGroup** = group;  
 });  
 buttonGroup.add(button);  
 containerButton.add(button);  
 }  
  
 **private void** createButtonsForGroups(Container containerButton, Container containerStudents) {  
 containerButton.setLayout(**new** BoxLayout(containerButton, BoxLayout.***X\_AXIS***));  
 List<Group> allGroups = **studentService**.getAllGroups();  
 ButtonGroup buttonGroup = **new** ButtonGroup();  
 **for** (Group group : allGroups) {  
 addGroupButton(group.getGroupNumber(), containerButton, containerStudents, buttonGroup);  
 }  
 }  
  
 **private void** createStudentsPane(Container container, Group group) **throws** ParseException {  
 JTable studentsTable = **new** JTable(createTableData(group), createHeaderTable()) {  
 @Override  
 **public** Class<?> getColumnClass(**int** column) {  
 **if** (column == 0 || column == 1) {  
 **return** String.**class**;  
 }  
 **return** Boolean.**class**;  
 }  
  
 @Override  
 **public void** setValueAt(Object value, **int** row, **int** col) {  
 **super**.setValueAt(value, row, col);  
 **if** (col > 1) {  
 String studentName = (String) getValueAt(row, 0);  
 String date = **null**;  
 **try** {  
 Date dateOldFormat = **new** SimpleDateFormat(**"dd/MM/yyyy"**).parse(createHeaderTable()[col]);  
 date = **new** SimpleDateFormat(**"yyyy-MM-dd"**).format(dateOldFormat);  
 } **catch** (ParseException e) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Что-то пошло не так..."**},  
 **"Warning"**,  
 JOptionPane.***ERROR\_MESSAGE***);  
 }  
 Student student = checkFullStudentName(studentName);  
 **if** ((Boolean) **this**.getValueAt(row, col)) {  
 Visit visit = **new** Visit();  
 **studentService**.addVisit(date, student, visit);  
 } **else** {  
 DateLecture deleteDate = **new** DateLecture();  
 **if** (student == **null**) {  
 JOptionPane.*showMessageDialog*(MainFrame.**this**,  
 **new** String[]{**"Такого студента не существует!"**},  
 **"Warning"**,  
 JOptionPane.***WARNING\_MESSAGE***);  
 } **else** {  
 **for** (Student allStudent : **studentService**.getStudentsByGroup(group)) {  
 **if** (allStudent.getStudentLastname().equals(student.getStudentLastname())) {  
 student.setId(allStudent.getId());  
 }  
 }  
 **for** (DateLecture allDate : **studentService**.getAllDates()) {  
 **if** (allDate.getDate().equals(date)) {  
 deleteDate.setId(allDate.getId());  
 **studentService**.deleteVisit(deleteDate.getId(), student.getId());  
 }  
 }  
 }  
 }  
 container.removeAll();  
 **try** {  
 createStudentsPane(container, group);  
 } **catch** (ParseException e) {  
 e.printStackTrace();  
 }  
 container.repaint();  
 container.revalidate();  
 }  
 }  
 };  
 studentsTable.setAutoResizeMode(JTable.***AUTO\_RESIZE\_OFF***);  
 studentsTable.getColumnModel().getColumn(0).setPreferredWidth(200);  
 studentsTable.getColumnModel().getColumn(1).setPreferredWidth(110);  
 **for** (**int** i = 2; i < studentsTable.getColumnCount(); i++) {  
 studentsTable.getColumnModel().getColumn(i).setPreferredWidth(80);  
 }  
 studentsTable.setPreferredScrollableViewportSize(**new** Dimension(600, 700));  
 studentsTable.setAutoCreateRowSorter(**true**);  
 JScrollPane scrollPane = **new** JScrollPane(studentsTable);  
 scrollPane.setHorizontalScrollBarPolicy(JScrollPane.***HORIZONTAL\_SCROLLBAR\_ALWAYS***);  
 scrollPane.setVerticalScrollBarPolicy(JScrollPane.***VERTICAL\_SCROLLBAR\_ALWAYS***);  
 scrollPane.setPreferredSize(**new** Dimension(600, 700));  
 scrollPane.setMaximumSize(scrollPane.getSize());  
 scrollPane.setMinimumSize(scrollPane.getSize());  
 container.removeAll();  
 container.add(scrollPane);  
 container.repaint();  
 container.revalidate();  
 }  
  
 **private** String[] createListOfDates() {  
 List<DateLecture> dates = **studentService**.getAllDates();  
 String[] datesOfVisit = **new** String[dates.size()];  
 **int** j = 0;  
 **for** (DateLecture date : dates) {  
 datesOfVisit[j] = date.getDate();  
 j++;  
 }  
 **return** datesOfVisit;  
 }  
  
 **private** String[] createListOfDatesForHeader() **throws** ParseException {  
 List<DateLecture> dates = **studentService**.getAllDates();  
 String[] datesOfVisit = **new** String[dates.size()];  
 **int** j = 0;  
 **for** (DateLecture date : dates) {  
 Date dateOldFormat = **new** SimpleDateFormat(**"yyyy-MM-dd"**).parse(date.getDate());  
 date.setDate(**new** SimpleDateFormat(**"dd/MM/yyyy"**).format(dateOldFormat));  
 datesOfVisit[j] = date.getDate();  
 j++;  
 }  
 **return** datesOfVisit;  
 }  
  
 **private** String[] createStudentsList(Group group) {  
 List<Student> studentList = **studentService**.getStudentsByGroup(group);  
 String[] students = **new** String[studentList.size()];  
 **int** i = 0;  
 **for** (Student student : studentList) {  
 students[i] = student.getStudentLastname() + **" "** +  
 student.getStudentName() + **" "** +  
 student.getStudentPatronymic();  
 i++;  
 }  
 **return** students;  
 }  
  
 **private** String[] createHeaderTable() **throws** ParseException {  
 String[] dates = createListOfDatesForHeader();  
 String[] header = **new** String[dates.**length** + 2];  
 header[0] = **"ФИО"**;  
 header[1] = **"Кол-во пропусков"**;  
 System.*arraycopy*(dates, 0, header, 2, dates.**length**);  
 **return** header;  
 }  
  
 **private** Object[][] createTableData(Group group) {  
 String[] students = createStudentsList(group);  
 List<Student> studentsDb = **studentService**.getStudentsByGroup(group);  
 String[] dates = createListOfDates();  
 List<DateLecture> datesDb = **studentService**.getAllDates();  
 Object[][] tableData = **new** Object[students.**length**][datesDb.size() + 2];  
 **int** i = 0;  
 **for** (Student student : studentsDb) {  
 List<DateLecture> visits = **studentService**.getDatesByStudent(student);  
 tableData[i][0] = students[i];  
 tableData[i][1] = datesDb.size() - **studentService**.getDatesByStudent(student).size();  
 **int** k = 0;  
 **for** (**int** j = 2; j < dates.**length** + 2; j++) {  
 tableData[i][j] = checkDate(visits, dates[k]);  
 k++;  
 }  
 i++;  
 }  
 **return** tableData;  
 }  
  
 **private boolean** checkDate(List<DateLecture> datesDb, String dateString) {  
 **for** (DateLecture date : datesDb) {  
 **if** (date.getDate().equals(dateString)) **return true**;  
 }  
 **return false**;  
 }  
}

DateLabelFormatter.java

**package** com.coursework.view;  
  
**import** javax.swing.\*;  
**import** java.text.ParseException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.Calendar;  
  
*/\*\*  
 \* Date formatting class  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** DateLabelFormatter **extends** JFormattedTextField.AbstractFormatter {  
  
 **private final** String **datePattern** = **"dd/MM/yyyy"**;  
 **private final** SimpleDateFormat **dateFormatter** = **new** SimpleDateFormat(**datePattern**);  
  
 */\*\*  
 \* Method for converting a string to a date  
 \** ***@param text*** *date string  
 \** ***@return*** *date  
 \** ***@throws*** *ParseException  
 \*/* @Override  
 **public** Object stringToValue(String text) **throws** ParseException {  
 **return dateFormatter**.parseObject(text);  
 }  
  
 */\*\*  
 \* Method for converting a date to a string  
 \** ***@param value*** *date  
 \** ***@return*** *date string  
 \*/* @Override  
 **public** String valueToString(Object value) {  
 **if** (value != **null**) {  
 Calendar cal = (Calendar) value;  
 **return dateFormatter**.format(cal.getTime());  
 }  
  
 **return ""**;  
 }  
  
}

AboutAuthor.java

**package** com.coursework.view;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
*/\*\*  
 \* About the Author window  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** AboutAuthor **extends** JFrame {  
  
 **private final** JFrame **frame**;  
 **private** Container **container**;  
 **private** ReaderImage **readerImage**;  
 **private** Icon **icon**;  
 **private** JPanel **image**;  
 **private** JLabel **iconLabel**;  
 **private** JLabel **author**;  
 **private** JLabel **group**;  
 **private** JLabel **info**;  
 **private** JButton **back**;  
  
 */\*\*  
 \* Constructor  
 \*/* **public** AboutAuthor() {  
 **frame** = **new** JFrame(**"Об авторе"**);  
 **frame**.setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);  
 **frame**.setPreferredSize(**new** Dimension(550, 650));  
 **frame**.setResizable(**false**);  
 **frame**.setLocation(400, 100);  
  
 **container** = **new** Container();  
 **container**.setLayout(**new** BoxLayout(**container**, BoxLayout.***Y\_AXIS***));  
  
 **readerImage** = **new** ReaderImage();  
 **icon** = **readerImage**.scaleImage(**"images/cat1.png"**, 368, 400);  
  
 **image** = **new** JPanel();  
 **image**.setMaximumSize(**new** Dimension(550, 450));  
 **image**.setMinimumSize(**new** Dimension(550, 450));  
  
 **iconLabel** = **new** JLabel();  
 **iconLabel**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **iconLabel**.setIcon(**icon**);  
  
 **author** = **new** JLabel();  
 **author**.setText(**"Кузьмицкая Мария Васильевна"**);  
 **author**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **author**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **group** = **new** JLabel();  
 **group**.setText(**"студентка группы 10702419"**);  
 **group**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **group**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **info** = **new** JLabel();  
 **info**.setText(**"grisdeborah44@gmail.com"**);  
 **info**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **info**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **back** = **new** JButton(**"Назад"**);  
 **back**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
 **back**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **back**.setMinimumSize(**new** Dimension(100,40));  
 **back**.setMaximumSize(**new** Dimension(100,40));  
 **back**.addActionListener(e -> {  
 **frame**.dispose();  
 **new** MainFrame();  
 });  
  
 **image**.add(**iconLabel**);  
  
 **container**.add(**image**);  
 **container**.add(**author**);  
 **container**.add(**group**);  
 **container**.add(**info**);  
 **container**.add(**back**);  
  
 **frame**.setContentPane(**container**);  
 **frame**.pack();  
 **frame**.setVisible(**true**);  
 }  
}

AboutApp.java

**package** com.coursework.view;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
*/\*\*  
 \* About the Application window  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** AboutApp **extends** JFrame {  
  
 **private** JFrame **frame**;  
 **private** Container **container**;  
 **private** JPanel **theme**;  
 **private** JPanel **info**;  
 **private** JPanel **image**;  
 **private** JPanel **infoText**;  
 **private** ReaderImage **readerImage**;  
 **private** Icon **icon**;  
 **private** JLabel **iconLabel**;  
 **private** JLabel **courseTheme**;  
 **private** JLabel **label**;  
 **private** JLabel **functionNumber1**;  
 **private** JLabel **functionNumber2**;  
 **private** JLabel **functionNumber3**;  
 **private** JLabel **functionNumber4**;  
 **private** JLabel **functionNumber5**;  
 **private** JLabel **functionNumber6**;  
 **private** JButton **back**;  
  
 */\*\*  
 \* Constructor  
 \*/* **public** AboutApp() {  
 **frame** = **new** JFrame(**"О программе"**);  
 **frame**.setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);  
 **frame**.setPreferredSize(**new** Dimension(900, 550));  
 **frame**.setResizable(**false**);  
 **frame**.setLocation(400, 100);  
  
 **container** = **new** Container();  
 **container**.setLayout(**new** BoxLayout(**container**, BoxLayout.***Y\_AXIS***));  
  
 **theme** = **new** JPanel();  
 **theme**.setMaximumSize(**new** Dimension(900, 85));  
 **theme**.setMinimumSize(**new** Dimension(900, 85));  
  
 **info** = **new** JPanel();  
 **info**.setLayout(**new** BoxLayout(**info**, BoxLayout.***X\_AXIS***));  
 **info**.setMaximumSize(**new** Dimension(900, 280));  
 **info**.setMinimumSize(**new** Dimension(900, 280));  
  
 **image** = **new** JPanel();  
 **image**.setMaximumSize(**new** Dimension(350, 450));  
 **image**.setMinimumSize(**new** Dimension(350, 450));  
  
 **infoText** = **new** JPanel();  
 **infoText**.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));  
 **infoText**.setMaximumSize(**new** Dimension(400, 450));  
 **infoText**.setMinimumSize(**new** Dimension(400, 450));  
  
 **readerImage** = **new** ReaderImage();  
 **icon** = **readerImage**.scaleImage(**"images/about\_app1.png"**, 250, 242);  
  
 **iconLabel** = **new** JLabel();  
 **iconLabel**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **iconLabel**.setIcon(**icon**);  
  
 **courseTheme** = **new** JLabel();  
 **courseTheme**.setText(**"Посещение лекционных занятий"**);  
 **courseTheme**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **courseTheme**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 30));  
  
 **label** = **new** JLabel();  
 **label**.setText(**"Приложение позволяет"**);  
 **label**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **label**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **functionNumber1** = **new** JLabel();  
 **functionNumber1**.setText(**"1. Контролировать посещаемость студентов."**);  
 **functionNumber1**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **functionNumber1**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **functionNumber2** = **new** JLabel();  
 **functionNumber2**.setText(**"2. Добавлять/Удалять группы."**);  
 **functionNumber2**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **functionNumber2**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **functionNumber3** = **new** JLabel();  
 **functionNumber3**.setText(**"3. Добавлять/Удалять студентов."**);  
 **functionNumber3**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **functionNumber3**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **functionNumber4** = **new** JLabel();  
 **functionNumber4**.setText(**"4. Добавлять даты посещений."**);  
 **functionNumber4**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **functionNumber4**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **functionNumber5** = **new** JLabel();  
 **functionNumber5**.setText(**"5. Сохранять результат в excel таблицы."**);  
 **functionNumber5**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **functionNumber5**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **functionNumber6** = **new** JLabel();  
 **functionNumber6**.setText(**"6. Сортировать список студентов."**);  
 **functionNumber6**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **functionNumber6**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
  
 **back** = **new** JButton(**"Назад"**);  
 **back**.setFont(**new** Font(**"Times New Roman"**, Font.***PLAIN***, 25));  
 **back**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **back**.setMinimumSize(**new** Dimension(170,70));  
 **back**.setMaximumSize(**new** Dimension(170,70));  
 **back**.addActionListener(e -> {  
 **frame**.dispose();  
 **new** MainFrame();  
 });  
  
 **infoText**.add(**label**);  
 **infoText**.add(**functionNumber1**);  
 **infoText**.add(**functionNumber2**);  
 **infoText**.add(**functionNumber3**);  
 **infoText**.add(**functionNumber4**);  
 **infoText**.add(**functionNumber5**);  
 **infoText**.add(**functionNumber6**);  
  
 **theme**.add(**courseTheme**);  
 **info**.add(**iconLabel**);  
 **info**.add(**infoText**);  
  
 **container**.add(**theme**);  
 **container**.add(**info**);  
 **container**.add(**back**);  
  
 **frame**.setContentPane(**container**);  
 **frame**.pack();  
 **frame**.setVisible(**true**);  
 }  
}

StudentServiceImpl.java

**package** com.coursework.service;  
  
**import** com.coursework.dao.StudentDao;  
**import** com.coursework.dao.StudentDaoImpl;  
**import** com.coursework.model.DateLecture;  
**import** com.coursework.model.Group;  
**import** com.coursework.model.Student;  
**import** com.coursework.model.Visit;  
  
**import** java.util.List;  
  
*/\*\*  
 \* Students service  
 \*/***public class** StudentServiceImpl **implements** StudentService{  
 **private final** StudentDao **studentDao** = StudentDaoImpl.*getInstance*();  
  
 **private static volatile** StudentServiceImpl *studentService*;  
  
 **private** StudentServiceImpl(){}  
  
 */\*\*  
 \** ***@return*** *instance of the class  
 \*/* **public static** StudentServiceImpl getInstance() {  
 **if** (*studentService* == **null**) {  
 **synchronized** (StudentServiceImpl.**class**) {  
 **if** (*studentService* == **null**) {  
 *studentService* = **new** StudentServiceImpl();  
 }  
 }  
  
 }  
 **return** *studentService*;  
 }  
  
 */\*\*  
 \* Method for searching group by number  
 \** ***@param group*** *group  
 \** ***@return*** *group  
 \*/* @Override  
 **public** Group getGroup(String group) {  
 **return studentDao**.getGroup(group);  
 }  
  
 */\*\*  
 \* Adding a date  
 \** ***@param date*** *date  
 \*/* @Override  
 **public void** addDate(DateLecture date) {  
 **studentDao**.addDate(date);  
 }  
  
 */\*\*  
 \* Adding a student  
 \** ***@param student*** *student  
 \** ***@param group*** *group  
 \*/* @Override  
 **public void** addStudent(Student student, Group group) {  
 **studentDao**.addStudent(student, group);  
 }  
  
 */\*\*  
 \* Method for searching students by date  
 \** ***@param date*** *date  
 \** ***@param group*** *group  
 \** ***@return*** *list of students  
 \*/* @Override  
 **public** List<Student> getStudentsByDate(DateLecture date, Group group) {  
 **return studentDao**.getStudentsByDate(date, group);  
 }  
  
 */\*\*  
 \* Adding a group  
 \** ***@param group*** *group  
 \*/* @Override  
 **public void** addGroup(Group group) {  
 **studentDao**.addGroup(group);  
 }  
  
 */\*\*  
 \* Delete a group  
 \** ***@param group*** *group  
 \*/* @Override  
 **public void** deleteGroup(String group) {  
 **studentDao**.deleteGroup(group);  
 }  
  
 */\*\*  
 \* Adding a visit  
 \** ***@param date*** *date  
 \** ***@param student*** *student  
 \** ***@param visit*** *visit  
 \*/* @Override  
 **public void** addVisit(String date, Student student, Visit visit) {  
 **studentDao**.addVisit(date, student, visit);  
 }  
  
 */\*\*  
 \* Method for searching all groups  
 \** ***@return*** *list of groups  
 \*/* @Override  
 **public** List<Group> getAllGroups() {  
 **return studentDao**.getAllGroups();  
 }  
  
 */\*\*  
 \* Method for searching students by group  
 \** ***@param group*** *group  
 \** ***@return*** *list of students  
 \*/* @Override  
 **public** List<Student> getStudentsByGroup(Group group) {  
 **return studentDao**.getStudentsByGroup(group);  
 }  
  
 */\*\*  
 \* Method for searching all dates  
 \** ***@return*** *list of dates  
 \*/* @Override  
 **public** List<DateLecture> getAllDates() {  
 **return studentDao**.getAllDates();  
 }  
  
 */\*\*  
 \* A method for finding the dates of student visits  
 \** ***@param student*** *student  
 \** ***@return*** *list of date  
 \*/* @Override  
 **public** List<DateLecture> getDatesByStudent(Student student) {  
 **return studentDao**.getDatesByStudent(student);  
 }  
  
 */\*\*  
 \* Method for deleting a visit  
 \** ***@param dateId*** *date id  
 \** ***@param studentId*** *student id  
 \*/* @Override  
 **public void** deleteVisit(**int** dateId, **int** studentId) {  
 **studentDao**.deleteVisit(dateId, studentId);  
 }  
  
 */\*\*  
 \* Method for searching group by id  
 \** ***@param groupId*** *group id  
 \** ***@return*** *group  
 \*/* @Override  
 **public** Group getGroupById(**int** groupId) {  
 **return studentDao**.getGroupById(groupId);  
 }  
  
 */\*\*  
 \* Method for deleting a student  
 \** ***@param student*** *student  
 \*/* @Override  
 **public void** deleteStudent(Student student) {  
 **studentDao**.deleteStudent(student);  
 }  
  
 */\*\*  
 \* Method for searching for a student by name  
 \** ***@param student*** *student  
 \** ***@return*** *student  
 \*/* @Override  
 **public** Student getStudentByName(Student student) {  
 **return studentDao**.getStudentByName(student);  
 }  
}

FileServiceImpl.java

**package** com.coursework.service;  
  
**import** com.coursework.model.DateLecture;  
**import** com.coursework.model.Student;  
  
**import** org.apache.poi.ss.usermodel.Cell;  
**import** org.apache.poi.xssf.usermodel.XSSFRow;  
**import** org.apache.poi.xssf.usermodel.XSSFSheet;  
**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.io.File;  
**import** java.io.FileOutputStream;  
**import** java.io.IOException;  
**import** java.util.\*;  
**import** java.util.List;  
**import** java.util.stream.Collectors;  
  
*/\*\*  
 \* File Service  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** FileServiceImpl **implements** FileService {  
  
 **private final** StudentService **studentService**;  
 **private static volatile** FileServiceImpl *fileService*;  
  
 **private** FileServiceImpl() {  
 **studentService** = StudentServiceImpl.*getInstance*();  
 }  
  
 */\*\*  
 \** ***@return*** *instance of the class  
 \*/* **public static** FileServiceImpl getInstance() {  
 **if** (*fileService* == **null**) {  
 **synchronized** (FileServiceImpl.**class**) {  
 **if** (*fileService* == **null**) {  
 *fileService* = **new** FileServiceImpl();  
 }  
 }  
  
 }  
 **return** *fileService*;  
 }  
  
 */\*\*  
 \* Method for creating a file  
 \** ***@param container*** *container  
 \** ***@return*** *file  
 \*/* @Override  
 **public** File createFile(Container container) {  
 JFileChooser jFileChooser = **new** JFileChooser();  
 jFileChooser.showSaveDialog(container);  
  
 **return new** File(jFileChooser.getSelectedFile().getPath() + **".xlsx"**);  
 }  
  
 */\*\*  
 \* Method for filling in the file  
 \** ***@param file*** *file  
 \** ***@param students*** *list of students  
 \** ***@param group*** *group  
 \** ***@throws*** *IOException  
 \*/* @Override  
 **public void** writeDataToExcel(File file, List<Student> students, String group) **throws** IOException {  
 XSSFWorkbook workbook = **new** XSSFWorkbook();  
 XSSFSheet spreadsheet = workbook.createSheet(**"Student Data"**);  
 XSSFRow row;  
  
 Map<Integer, Object[]> studentData = **new** TreeMap<>();  
 String[] groupRow = **new** String[2];  
 groupRow[1] = **"Группа №"** + group;  
 studentData.put(  
 1,  
 groupRow  
 );  
 studentData.put(  
 2,  
 setHeader(**studentService**.getAllDates()));  
  
 setTableData(studentData, students, **studentService**.getAllDates());  
 Set<Integer> keyId = studentData.keySet();  
  
 **int** rowId = 0;  
 **for** (Integer key : keyId) {  
 row = spreadsheet.createRow(rowId++);  
 Object[] objectArr = studentData.get(key);  
 **int** cellId = 0;  
 **for** (Object obj : objectArr) {  
 Cell cell = row.createCell(cellId++);  
 cell.setCellValue((String) obj);  
 }  
 }  
  
 spreadsheet.setColumnWidth(0, 1250);  
 spreadsheet.setColumnWidth(1, 8000);  
 spreadsheet.setColumnWidth(2, 4100);  
 **for** (**int** i = 3; i < **studentService**.getAllDates().size() + 3; i++) {  
 spreadsheet.setColumnWidth(i, 2700);  
 }  
  
 FileOutputStream out = **new** FileOutputStream(file.getPath());  
 workbook.write(out);  
 out.close();  
 }  
  
 **private** String[] setHeader(List<DateLecture> dates) {  
 String[] headerStrings = **new** String[dates.size() + 3];  
 headerStrings[0] = **"№"**;  
 headerStrings[1] = **"ФИО"**;  
 headerStrings[2] = **"Кол-во пропусков"**;  
 **for** (**int** i = 3; i < headerStrings.**length**; i++) {  
 headerStrings[i] = dates.get(i - 3).getDate();  
 }  
 **return** headerStrings;  
 }  
  
 **private void** setTableData(Map<Integer, Object[]> studentData, List<Student> students1, List<DateLecture> dates) {  
 String[] rowData = **new** String[dates.size() + 3];  
 String studentsFullName;  
 **int** numberOfPasses;  
 List<Student> students = students1.stream()  
 .sorted(Comparator.*comparing*(Student::getFullName))  
 .collect(Collectors.*toList*());  
  
 **for** (**int** i = 0; i < students.size(); i++) {  
 List<DateLecture> visits = **studentService**.getDatesByStudent(students.get(i));  
 studentsFullName = students.get(i).getFullName();  
 numberOfPasses = dates.size() - **studentService**.getDatesByStudent(students.get(i)).size();  
 rowData[0] = String.*valueOf*(i + 1);  
 rowData[1] = studentsFullName;  
 rowData[2] = String.*valueOf*(numberOfPasses);  
 **for** (**int** j = 0; j < dates.size(); j++) {  
 **if** (*checkDate*(visits, dates.get(j).getDate())) {  
 rowData[j + 3] = **"+"**;  
 } **else** {  
 rowData[j + 3] = **"-"**;  
 }  
 }  
 studentData.put(i + 3, rowData);  
 rowData = **new** String[dates.size() + 3];  
 }  
 }  
  
 **private static boolean** checkDate(List<DateLecture> datesDb, String dateString) {  
 **for** (DateLecture date : datesDb) {  
 **if** (date.getDate().equals(dateString)) **return true**;  
 }  
 **return false**;  
 }  
}

Visit.java

**package** com.coursework.model;  
  
*/\*\*  
 \* Class-entity for student visit  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** Visit {  
 **private int id**;  
 **private int dateId**;  
 **private int studentId**;  
  
 */\*\*  
 \* Getter for id  
 \** ***@return*** *id  
 \*/* **public int** getId() {  
 **return id**;  
 }  
  
 */\*\*  
 \* Setter for id  
 \** ***@param id*** *\*/* **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 */\*\*  
 \* Getter for date id  
 \** ***@return*** *date id  
 \*/* **public int** getDateId() {  
 **return dateId**;  
 }  
  
 */\*\*  
 \* Setter for date id  
 \** ***@param dateId*** *\*/* **public void** setDateId(**int** dateId) {  
 **this**.**dateId** = dateId;  
 }  
  
 */\*\*  
 \* Getter for student id  
 \** ***@return*** *student id  
 \*/* **public int** getStudentId() {  
 **return studentId**;  
 }  
  
 */\*\*  
 \* Setter for student id  
 \** ***@param studentId*** *\*/* **public void** setStudentId(**int** studentId) {  
 **this**.**studentId** = studentId;  
 }  
}

Student.java

**package** com.coursework.model;  
  
*/\*\*  
 \* Class-entity for student  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** Student {  
 **private int id**;  
 **private int groupId**;  
 **private** String **studentName**;  
 **private** String **studentLastname**;  
 **private** String **studentPatronymic**;  
  
 */\*\*  
 \* Getter for id  
 \** ***@return*** *id  
 \*/* **public int** getId() {  
 **return id**;  
 }  
  
 */\*\*  
 \* Setter for id  
 \** ***@param id*** *\*/* **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 */\*\*  
 \* Getter for group id  
 \** ***@return*** *group id  
 \*/* **public int** getGroupId() {  
 **return groupId**;  
 }  
  
 */\*\*  
 \* Setter for group id  
 \** ***@param group\_id*** *\*/* **public void** setGroupId(**int** group\_id) {  
 **this**.**groupId** = group\_id;  
 }  
  
 */\*\*  
 \* Getter for student name  
 \** ***@return*** *student name  
 \*/* **public** String getStudentName() {  
 **return studentName**;  
 }  
  
 */\*\*  
 \* Setter for student name  
 \** ***@param student\_name*** *\*/* **public void** setStudentName(String student\_name) {  
 **this**.**studentName** = student\_name;  
 }  
  
 */\*\*  
 \* Getter for student lastname  
 \** ***@return*** *lastname  
 \*/* **public** String getStudentLastname() {  
 **return studentLastname**;  
 }  
  
 */\*\*  
 \* Setter for student lastname  
 \** ***@param student\_lastname*** *\*/* **public void** setStudentLastname(String student\_lastname) {  
 **this**.**studentLastname** = student\_lastname;  
 }  
  
 */\*\*  
 \* Getter for student patronymic  
 \** ***@return*** *patronymic  
 \*/* **public** String getStudentPatronymic() {  
 **return studentPatronymic**;  
 }  
  
 */\*\*  
 \* Setter for student patronymic  
 \** ***@param student\_patronymic*** *\*/* **public void** setStudentPatronymic(String student\_patronymic) {  
 **this**.**studentPatronymic** = student\_patronymic;  
 }  
  
 */\*\*  
 \* Getter for student full name  
 \** ***@return*** *full name  
 \*/* **public** String getFullName() {  
 **return this**.**studentLastname** + **" "** + **this**.**studentName** + **" "** + **this**.**studentPatronymic**;  
 }  
}

Group.java

**package** com.coursework.model;  
  
*/\*\*  
 \* Class-entity for group of students  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** Group {  
 **private int id**;  
 **private** String **groupNumber**;  
  
 */\*\*  
 \* Getter for id  
 \** ***@return*** *id  
 \*/* **public int** getId() {  
 **return id**;  
 }  
  
 */\*\*  
 \* Setter for id  
 \** ***@param id*** *\*/* **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 */\*\*  
 \* Getter for group number  
 \** ***@return*** *group number  
 \*/* **public** String getGroupNumber() {  
 **return groupNumber**;  
 }  
  
 */\*\*  
 \* Setter for group number  
 \** ***@param groupNumber*** *\*/* **public void** setGroupNumber(String groupNumber) {  
 **this**.**groupNumber** = groupNumber;  
 }  
}

DateLecture.java

**package** com.coursework.model;  
  
*/\*\*  
 \* Class-entity for lecture dates  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** DateLecture {  
 **private int id**;  
 **private** String **date**;  
  
 */\*\*  
 \* Getter for date  
 \** ***@return*** *id of date  
 \*/* **public int** getId() {  
 **return id**;  
 }  
  
 */\*\*  
 \* Setter for date  
 \** ***@param id*** *\*/* **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 */\*\*  
 \* Getter for date  
 \** ***@return*** *date  
 \*/* **public** String getDate() {  
 **return date**;  
 }  
  
 */\*\*  
 \* Setter for date  
 \** ***@param date*** *\*/* **public void** setDate(String date) {  
 **this**.**date** = date;  
 }  
  
}

StudentsDaoImpl.java

**package** com.coursework.dao;  
  
**import** com.coursework.dao.db.DBManager;  
**import** com.coursework.model.DateLecture;  
**import** com.coursework.model.Group;  
**import** com.coursework.model.Student;  
**import** com.coursework.model.Visit;  
  
**import** java.sql.\*;  
**import** java.text.SimpleDateFormat;  
**import** java.util.ArrayList;  
**import** java.util.Date;  
**import** java.util.List;  
  
*/\*\*  
 \* The layer between the database and the application  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** StudentDaoImpl **implements** StudentDao {  
  
 **private static volatile** StudentDaoImpl *instance*;  
 **private static final** String ***INSERT\_DATE*** = **"insert into dates (date\_of\_visit) values (?)"**;  
 **private static final** String ***INSERT\_STUDENT*** = **"insert into students (group\_id, student\_name, student\_lastname,student\_patronymic) values ((select id from groups\_students where group\_number = ?),?,?,?)"**;  
 **private static final** String ***SELECT\_STUDENTS\_BY\_DATE*** = **"select \* from students inner join student\_date on students.id = student\_date.student\_id inner join dates on student\_date.date\_id = dates.id where (select id from dates where date\_of\_visit = ?) = student\_date.date\_id and group\_id = (select id from groups\_students where group\_number = ?)"**;  
 **private static final** String ***INSERT\_GROUP*** = **"insert into groups\_students (group\_number) values (?)"**;  
 **private static final** String ***DELETE\_GROUP1*** = **"delete from students where group\_id = (select id from groups\_students where group\_number = ?)"**;  
 **private static final** String ***DELETE\_GROUP2*** = **"delete from groups\_students where group\_number = ?"**;  
 **private static final** String ***DELETE\_GROUP3*** = **"delete from student\_date where student\_id = (select id from students where group\_id = (select id from groups\_students where group\_number = ?))"**;  
 **private static final** String ***INSERT\_VISIT*** = **"insert into student\_date(date\_id, student\_id) values ((select id from dates where date\_of\_visit = ?),(select id from students where student\_name = ? and student\_lastname = ? and student\_patronymic = ?))"**;  
 **private static final** String ***SELECT\_GROUP\_BY\_NUMBER*** = **"select \* from groups\_students where group\_number = ?"**;  
 **private static final** String ***SELECT\_GROUP\_BY\_ID*** = **"select \* from groups\_students where id = ?"**;  
 **private static final** String ***SELECT\_ALL\_GROUPS*** = **"select \* from groups\_students"**;  
 **private static final** String ***SELECT\_STUDENTS\_BY\_GROUP*** = **"select \* from students where group\_id = ?"**;  
 **private static final** String ***SELECT\_ALL\_DATES*** = **"select \* from dates"**;  
 **private static final** String ***SELECT\_DATES\_BY\_STUDENT*** = **"select dates.id, dates.date\_of\_visit from dates inner join student\_date on dates.id = student\_date.date\_id where (select id from students where student\_name = ? and student\_lastname = ? and student\_patronymic = ?) = student\_date.student\_id"**;  
 **private static final** String ***DELETE\_VISIT*** = **"delete from student\_date where date\_id = ? and student\_id = ?"**;  
 **private static final** String ***SELECT\_STUDENT\_BY\_NAME*** = **"select \* from students where student\_name = ? and student\_lastname = ? and student\_patronymic = ? and group\_id = ?"**;  
 **private static final** String ***DELETE\_STUDENT2*** = **"delete from students where group\_id = ? and student\_name = ? and student\_lastname = ? and student\_patronymic = ?"**;  
 **private static final** String ***DELETE\_STUDENT1*** = **"delete from student\_date where student\_id = ?"**;  
  
 **private** StudentDaoImpl() {  
 }  
  
 */\*\*  
 \** ***@return*** *instance of the class  
 \*/* **public static** StudentDaoImpl getInstance() {  
 **if** (*instance* == **null**) {  
 **synchronized** (StudentDaoImpl.**class**) {  
 **if** (*instance* == **null**) {  
 *instance* = **new** StudentDaoImpl();  
 }  
 }  
 }  
 **return** *instance*;  
  
 }  
  
 */\*\*  
 \* Method for initializing group using a database  
 \** ***@param resultSet*** *resultSet  
 \** ***@return*** *group  
 \** ***@throws*** *SQLException  
 \*/* **public** Group initGroup(ResultSet resultSet) **throws** SQLException {  
 Group group = **new** Group();  
 group.setId(resultSet.getInt(**"id"**));  
 group.setGroupNumber(resultSet.getString(**"group\_number"**));  
 **return** group;  
 }  
  
 */\*\*  
 \* Method for initializing student using a database  
 \** ***@param resultSet*** *resultSet  
 \** ***@return*** *student  
 \** ***@throws*** *SQLException  
 \*/* **public** Student initStudent(ResultSet resultSet) **throws** SQLException {  
 Student student = **new** Student();  
 student.setId(resultSet.getInt(**"id"**));  
 student.setGroupId(resultSet.getInt(**"group\_id"**));  
 student.setStudentName(resultSet.getString(**"student\_name"**));  
 student.setStudentLastname(resultSet.getString(**"student\_lastname"**));  
 student.setStudentPatronymic(resultSet.getString(**"student\_patronymic"**));  
 **return** student;  
 }  
  
 */\*\*  
 \* Method for initializing date using a database  
 \** ***@param resultSet*** *resultSet  
 \** ***@return*** *date of lecture  
 \** ***@throws*** *SQLException  
 \*/* **public** DateLecture initDates(ResultSet resultSet) **throws** SQLException {  
 DateLecture dates = **new** DateLecture();  
 dates.setId(resultSet.getInt(**"id"**));  
 dates.setDate(resultSet.getString(**"date\_of\_visit"**));  
 **return** dates;  
 }  
  
 */\*\*  
 \* Method for initializing visit using a database  
 \** ***@param resultSet*** *resultSet  
 \** ***@return*** *visit  
 \** ***@throws*** *SQLException  
 \*/* **public** Visit initVisit(ResultSet resultSet) **throws** SQLException {  
 Visit visit = **new** Visit();  
 visit.setId(resultSet.getInt(**"id"**));  
 visit.setDateId(resultSet.getInt(**"date\_of\_visit"**));  
 visit.setStudentId(resultSet.getInt(**"date\_of\_visit"**));  
 **return** visit;  
 }  
  
 */\*\*  
 \* Adding a date to the database  
 \** ***@param date*** *date  
 \*/* @Override  
 **public void** addDate(DateLecture date) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 Date utilDate = parseDate(date.getDate());  
 java.sql.Date sqlDate = **new** java.sql.Date(utilDate.getTime());  
  
 PreparedStatement preparedStatement = connection.prepareStatement(***INSERT\_DATE***);  
 preparedStatement.setDate(1, sqlDate);  
 preparedStatement.executeUpdate();  
 **try** (ResultSet resultSet = preparedStatement.getGeneratedKeys()) {  
 **if** (resultSet.next()) {  
 date.setId(resultSet.getInt(**"id"**));  
 }  
 }  
  
 } **catch** (SQLException e) {  
 System.***out***.println(**"Incorrectly entered data"**);  
 ;  
 }  
 }  
  
 */\*\*  
 \* Adding a student to the database  
 \** ***@param student*** *student  
 \** ***@param group*** *group  
 \*/* @Override  
 **public void** addStudent(Student student, Group group) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***INSERT\_STUDENT***);  
 preparedStatement.setString(1, group.getGroupNumber());  
 preparedStatement.setString(2, student.getStudentName());  
 preparedStatement.setString(3, student.getStudentLastname());  
 preparedStatement.setString(4, student.getStudentPatronymic());  
 preparedStatement.executeUpdate();  
 **try** (ResultSet resultSet = preparedStatement.getGeneratedKeys()) {  
 **if** (resultSet.next()) {  
 student.setId(resultSet.getInt(**"id"**));  
 }  
 }  
  
 } **catch** (SQLException e) {  
 System.***out***.println(**"Incorrectly entered data"**);  
 ;  
 }  
 }  
  
 */\*\*  
 \* Method for searching students by date  
 \** ***@param date*** *date  
 \** ***@param group*** *group  
 \** ***@return*** *list of students  
 \*/* @Override  
 **public** List<Student> getStudentsByDate(DateLecture date, Group group) {  
 List<Student> students = **new** ArrayList<>();  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 Date utilDate = parseDate(date.getDate());  
 java.sql.Date sqlDate = **new** java.sql.Date(utilDate.getTime());  
  
 PreparedStatement preparedStatement = connection.prepareStatement(***SELECT\_STUDENTS\_BY\_DATE***);  
 preparedStatement.setDate(1, sqlDate);  
 preparedStatement.setString(2, group.getGroupNumber());  
 ResultSet resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 students.add(initStudent(resultSet));  
 }  
 } **catch** (SQLException e) {  
 System.***out***.println(**"error"**);  
 }  
 **return** students;  
 }  
  
 */\*\*  
 \* Adding a group to the database  
 \** ***@param group*** *group  
 \*/* @Override  
 **public void** addGroup(Group group) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***INSERT\_GROUP***);  
 preparedStatement.setString(1, group.getGroupNumber());  
 preparedStatement.executeUpdate();  
 **try** (ResultSet resultSet = preparedStatement.getGeneratedKeys()) {  
 **if** (resultSet.next()) {  
 group.setId(resultSet.getInt(**"id"**));  
 }  
 }  
  
 } **catch** (SQLException e) {  
 System.***out***.println(**"Incorrectly entered data"**);  
 ;  
 }  
 }  
  
 */\*\*  
 \* Delete a group to the database  
 \** ***@param group*** *group  
 \*/* @Override  
 **public void** deleteGroup(String group) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement1 = connection.prepareStatement(***DELETE\_GROUP1***);  
 preparedStatement1.setString(1, group);  
 preparedStatement1.executeUpdate();  
 PreparedStatement preparedStatement2 = connection.prepareStatement(***DELETE\_GROUP2***);  
 preparedStatement2.setString(1, group);  
 preparedStatement2.executeUpdate();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 System.***out***.println(**"Incorrectly entered data"**);  
 }  
 }  
  
 */\*\*  
 \* Method for searching group by number  
 \** ***@param group*** *group  
 \** ***@return*** *group  
 \*/* @Override  
 **public** Group getGroup(String group) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***SELECT\_GROUP\_BY\_NUMBER***);  
 preparedStatement.setString(1, group);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 **if** (resultSet.next()) {  
 **return** initGroup(resultSet);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
  
 */\*\*  
 \* Method for searching group by id  
 \** ***@param groupId*** *group id  
 \** ***@return*** *group  
 \*/* @Override  
 **public** Group getGroupById(**int** groupId) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***SELECT\_GROUP\_BY\_ID***);  
 preparedStatement.setInt(1, groupId);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 **if** (resultSet.next()) {  
 **return** initGroup(resultSet);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
  
 */\*\*  
 \* Adding a visit to the database  
 \** ***@param date*** *date  
 \** ***@param student*** *student  
 \** ***@param visit*** *visit  
 \*/* @Override  
 **public void** addVisit(String date, Student student, Visit visit) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 Date utilDate = parseDate(date);  
 java.sql.Date sqlDate = **new** java.sql.Date(utilDate.getTime());  
  
 PreparedStatement preparedStatement = connection.prepareStatement(***INSERT\_VISIT***);  
 preparedStatement.setDate(1, sqlDate);  
 preparedStatement.setString(2, student.getStudentName());  
 preparedStatement.setString(3, student.getStudentLastname());  
 preparedStatement.setString(4, student.getStudentPatronymic());  
 preparedStatement.executeUpdate();  
 **try** (ResultSet resultSet = preparedStatement.getGeneratedKeys()) {  
 **if** (resultSet.next()) {  
 visit.setId(resultSet.getInt(**"id"**));  
 }  
 }  
  
 } **catch** (SQLException e) {  
 System.***out***.println(**"Incorrectly entered data"**);  
 }  
 }  
  
 */\*\*  
 \* Method for parsing the date  
 \** ***@param object*** *date  
 \** ***@return*** *date  
 \*/* @Override  
 **public** Date parseDate(String object) {  
 **try** {  
 Date date;  
 SimpleDateFormat format = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
 format.setLenient(**false**);  
 date = format.parse(object);  
 **return** date;  
 } **catch** (Exception exception) {  
 System.***out***.println(**"DateLecture entered incorrectly"**);  
 **return null**;  
 }  
 }  
  
 */\*\*  
 \* Method for searching all groups  
 \** ***@return*** *list of groups  
 \*/* @Override  
 **public** List<Group> getAllGroups() {  
 List<Group> groups = **new** ArrayList<>();  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery(***SELECT\_ALL\_GROUPS***);  
 **while** (resultSet.next()) {  
 groups.add(initGroup(resultSet));  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return** groups;  
 }  
  
 */\*\*  
 \* Method for searching students by group  
 \** ***@param group*** *group  
 \** ***@return*** *list of students  
 \*/* @Override  
 **public** List<Student> getStudentsByGroup(Group group) {  
 List<Student> students = **new** ArrayList<>();  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***SELECT\_STUDENTS\_BY\_GROUP***);  
 preparedStatement.setInt(1, group.getId());  
 ResultSet resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 students.add(initStudent(resultSet));  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return** students;  
 }  
  
 */\*\*  
 \* Method for searching all dates  
 \** ***@return*** *list of dates  
 \*/* @Override  
 **public** List<DateLecture> getAllDates() {  
 List<DateLecture> dates = **new** ArrayList<>();  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery(***SELECT\_ALL\_DATES***);  
 **while** (resultSet.next()) {  
 dates.add(initDates(resultSet));  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return** dates;  
 }  
  
 */\*\*  
 \* A method for finding the dates of student visits  
 \** ***@param student*** *student  
 \** ***@return*** *list of dates  
 \*/* @Override  
 **public** List<DateLecture> getDatesByStudent(Student student) {  
 List<DateLecture> dates = **new** ArrayList<>();  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***SELECT\_DATES\_BY\_STUDENT***);  
 preparedStatement.setString(1, student.getStudentName());  
 preparedStatement.setString(2, student.getStudentLastname());  
 preparedStatement.setString(3, student.getStudentPatronymic());  
 ResultSet resultSet = preparedStatement.executeQuery();  
 **while** (resultSet.next()) {  
 dates.add(initDates(resultSet));  
 }  
 } **catch** (SQLException e) {  
 System.***out***.println(**"error"**);  
 }  
 **return** dates;  
 }  
  
 */\*\*  
 \* Method for deleting a visit  
 \** ***@param dateId*** *date id  
 \** ***@param studentId*** *student id  
 \*/* @Override  
 **public void** deleteVisit(**int** dateId, **int** studentId) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***DELETE\_VISIT***);  
 preparedStatement.setInt(1, dateId);  
 preparedStatement.setInt(2, studentId);  
 preparedStatement.executeUpdate();  
 } **catch** (SQLException e) {  
 System.***out***.println(**"Incorrectly entered data"**);  
 }  
 }  
  
 */\*\*  
 \* Method for deleting a student  
 \** ***@param student*** *student  
 \*/* @Override  
 **public void** deleteStudent(Student student) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***DELETE\_STUDENT1***);  
 preparedStatement.setInt(1, student.getId());  
 preparedStatement.executeUpdate();  
 PreparedStatement preparedStatement1 = connection.prepareStatement(***DELETE\_STUDENT2***);  
 preparedStatement1.setInt(1, student.getGroupId());  
 preparedStatement1.setString(2, student.getStudentName());  
 preparedStatement1.setString(3, student.getStudentLastname());  
 preparedStatement1.setString(4, student.getStudentPatronymic());  
 preparedStatement1.executeUpdate();  
 } **catch** (SQLException e) {  
 System.***out***.println(**"Incorrectly entered data"**);  
 }  
 }  
  
 */\*\*  
 \* Method for searching for a student by name  
 \** ***@param student*** *student  
 \** ***@return*** *student  
 \*/* @Override  
 **public** Student getStudentByName(Student student) {  
 **try** (Connection connection = DBManager.*getConnection*()) {  
 PreparedStatement preparedStatement = connection.prepareStatement(***SELECT\_STUDENT\_BY\_NAME***);  
 preparedStatement.setString(1, student.getStudentName());  
 preparedStatement.setString(2, student.getStudentLastname());  
 preparedStatement.setString(3, student.getStudentPatronymic());  
 preparedStatement.setInt(4, student.getGroupId());  
 ResultSet resultSet = preparedStatement.executeQuery();  
 **if** (resultSet.next()) {  
 **return** initStudent(resultSet);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
}

DBManager.java

**package** com.coursework.dao.db;  
  
**import** java.sql.Connection;  
**import** java.sql.DriverManager;  
**import** java.sql.SQLException;  
  
*/\*\*  
 \* Class for connecting to the database  
 \** ***@author*** *Kuzmitskaya Maryia  
 \** ***@version*** *1.0  
 \*/***public class** DBManager {  
 */\*\*  
 \* URL of our database  
 \*/* **public static** String *URL* = **"jdbc:postgresql://localhost:5432/lectures"**;  
 */\*\*  
 \* Login of our database  
 \*/* **public static** String *USER* = **"postgres"**;  
 */\*\*  
 \* Password of our database  
 \*/* **public static** String *PASSWORD* = **"pass"**;  
  
 */\*\*  
 \* Method for getting a connection  
 \** ***@return*** *connection  
 \** ***@throws*** *SQLException  
 \*/* **public static** Connection getConnection () **throws** SQLException {  
 **return** DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);  
 }  
}

# Список используемой литературы:

1. Кей С.Хорстман – Java. Библиотека профессионала.
2. Г. Шилдт – Swing руководство для начинающих.
3. Экслер, А.Б., "Самоучитель работы в Интернете" - Москва.: NT Press, 2010г.
4. Кузьмин А.В. Золотарева Н.Н. Поиск в Интернете - Санкт - Петербург.: Издательство НиТ, 2011г.
5. Гусев, В.С., "Яндекс. Эффективный поиск" - Москва, Санкт - Петербург, Киев.: Диалектика, 2010г.
6. Егоров, А.Б., " Поиск в Интернете" - Санкт - Петербург.: НиТ, 2010г.
7. Гусев, В.С., "Поиск, Internet" - Москва, Санкт - Петербург, Киев.: Диалектика, 2010г.
8. Гусев, В.С., "Google. Эффективный поиск" - Москва, Санкт - Петербург, Киев.: Диалектика, 2010г.
9. www.citforum.ru - CIT forum. Поисковые системы в сети Интернет
10. Андрей Аликберов. "Несколько слов о том, как работают роботы поисковых машин". http://www.citforum.ru/internet/search/art\_1. shtml