

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине «Проектирование программ в интеллектуальных системах»

на тему:

**“ПОСТРОЕНИЕ ГРАФИЧЕСКОГО
ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА НА ОСНОВЕ
ДИАЛОГОВ”**

Выполнил:

Студент группы
821702
Макаревич Д.А.

Проверил:

Садовский М.Е.

Минск, 2020

Задание:

Разработать оконное приложение с одним главным окном и несколькими дочерними диалогами. Вызов диалогов осуществляется через соответствующие пункты меню. Команды меню должны дублироваться на панели инструментов.

Вариант 14

Приложение должно быть построено при помощи шаблона проектирования Model-View-Controller.

Приложение должно уметь:

- формировать массив записей путем ввода записей через окна диалога (для полей типа дата использовать соответствующий тип data и компонент для ввода дат);
- осуществлять поиск записей в массиве соответствие с условиями, указанными в отдельном диалоговом окне (не в диалоге удаления);
- в варианте задания условия поиска должны вводиться в специальном диалоговом окне, результат поиска выводиться в нём же с помощью стандартных элементов управления, выбранной библиотеки пользовательского интерфейса.
- удалять запись из массива по условиям, указанным в варианте. Условия удаления должны вводиться в отдельном диалоговом окне (не в диалоге поиска);
- пользователю должно сообщаться о том были ли удалены записи, и сколько было удалено, согласно введенным условиям или таких записей не было найдено;
- отображать весь текущий массив записей в главном окне приложения с помощью стандартных элементов управления выбранной библиотеки.
- Сохранять и загружать массив записей из указываемого пользователем файла, посредством стандартного диалога сохранения/загрузки. Формат хранения данных в файле XML. Для записи использовать DOM парсер, для чтения SAX парсер.

Вариант 14

ФИО студента	Курс	группа	Общее число работ	Количество выполненных работ	Язык программирования

Условия поиска и удаления:

- по ФИО студента или группе;
- по курсу или языку программирования;
- по количеству выполненных работ или общему числу работ;
- По количеству не выполненных работ.

Замечание: Список языков программирования, количество выполненных работ или общее число работ в диалоге поиска, собирается системой и выводится в выпадающий список;

Главное окно приложения:



Диалог поиска:



Диалог удаления:

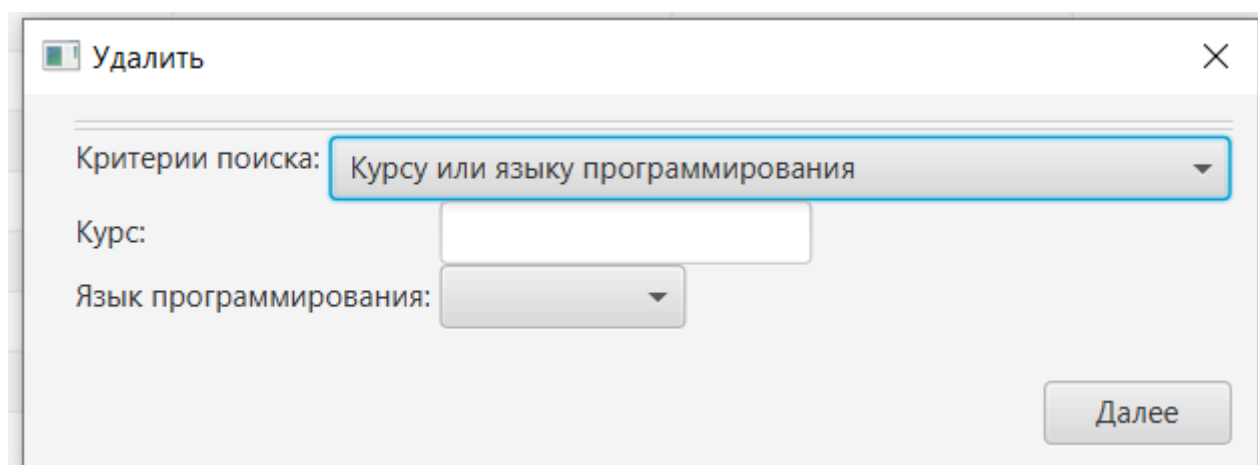


Рисунок 3 Удаление

Сохранение документ в XML формате:

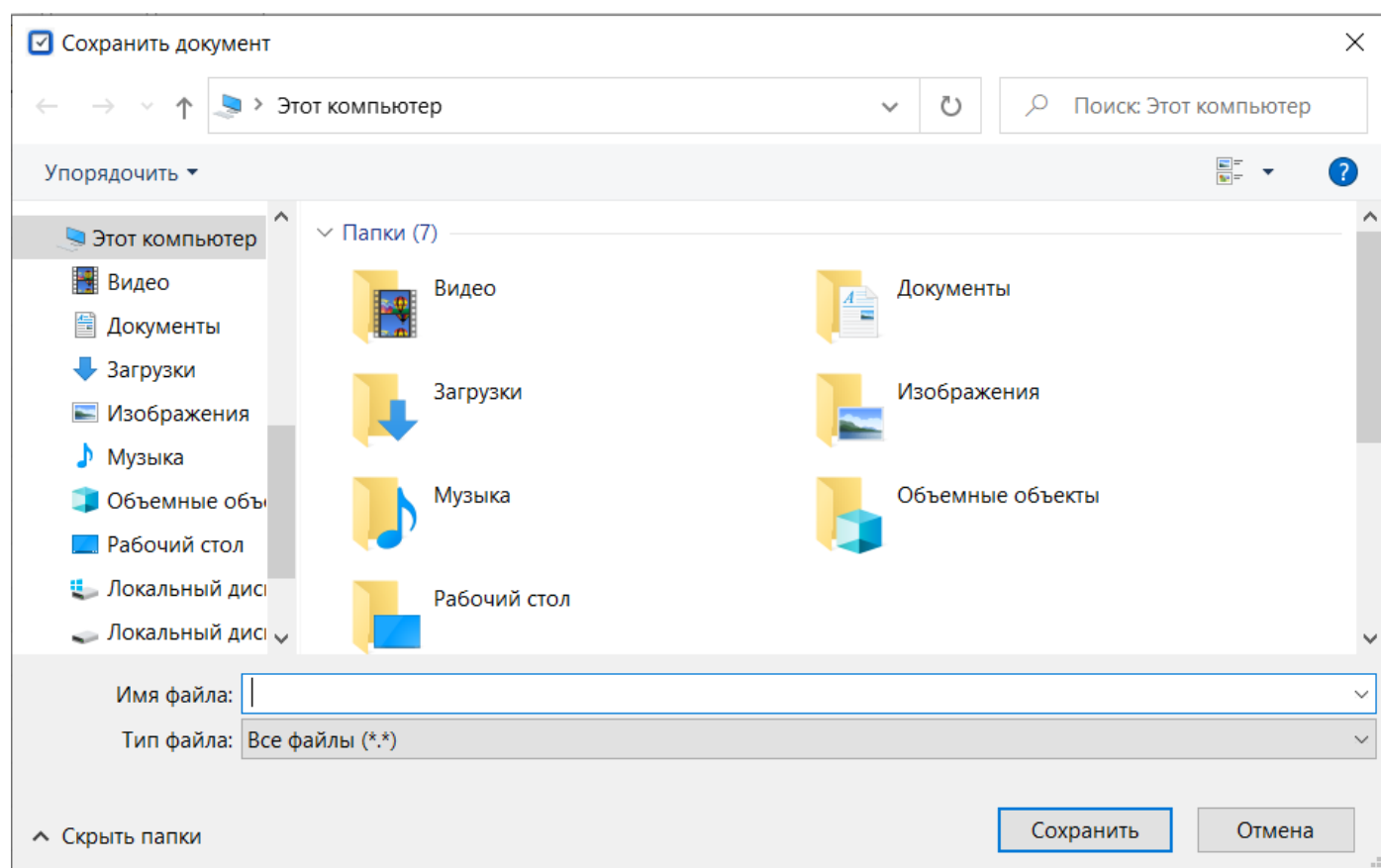
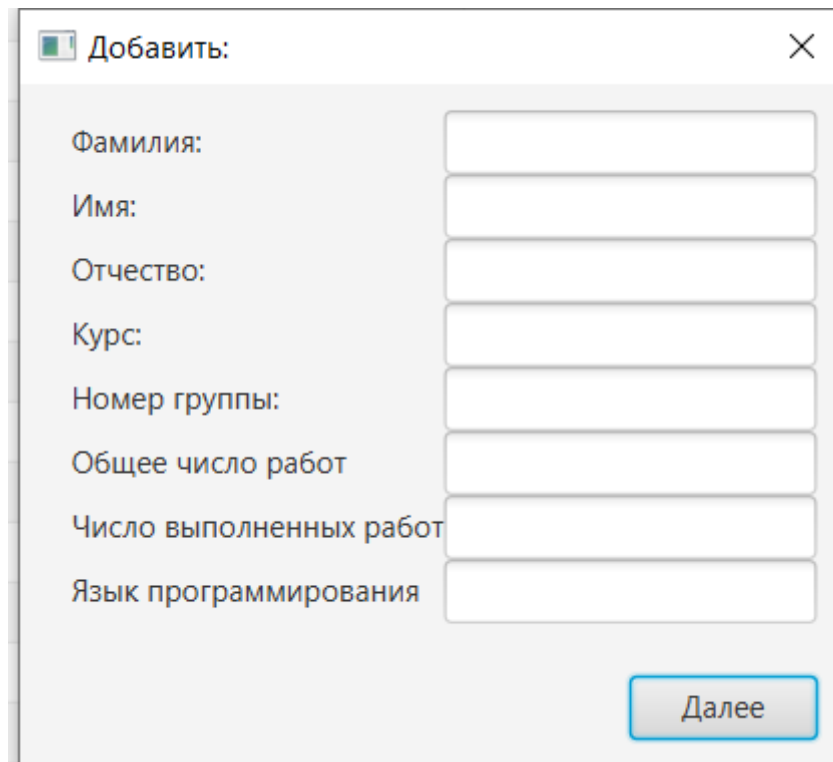


Рисунок 4 Сохранение файла

Диалог добавления:



Добавить: X

Фамилия:

Имя:

Отчество:

Курс:

Номер группы:

Общее число работ

Число выполненных работ

Язык программирования

Далее

Рисунок 5 Добавление

Описание классов

Данный класс содержит в себе список всех студентов над которыми совершаются дальнейшие действия.

Класс Model:

```
public class Model {

    private List<Student> studentList;

    public Model(int studentsNumber) {
        if (studentsNumber > 0) {
            studentList = generateStudents(studentsNumber);
        } else {
            studentList = new ArrayList<>();
        }
    }

    private List<Student> generateStudents(int studentsNumber) {
        List<Student> students = new ArrayList<>();

        for (int i = 0; i < studentsNumber; i++) {
            students.add(new Student(new SNP(RandomizationData.reqSurname(), RandomizationData.reqName(), RandomizationData.reqPatronymic()),
                RandomizationData.reqGroupNumber(),
                RandomizationData.reqCourseNumber(),
                RandomizationData.reqTotalWorksNumber(),
                RandomizationData.reqFinishedWorksCounter(),
                RandomizationData.reqComputerLanguage()
            ));
        }
        return students;
    }

    public List<Student> getStudentList() {
        return studentList;
    }

    public void setStudentList(List<Student> studentList) {
        this.studentList = studentList;
    }

    public void addStudent(Student student) {
        studentList.add(student);
    }
}
```

Рисунок 4 Класс модели

Класс Студент хранит всю информацию об объекте.

Класс Student:

```
public class Student {

    private SNP snp;
    private String groupNumber;
    private int courseNumber;
    private String computerLanguage;
    private int finishedWorksCounter;
    private int totalWorksNumber;

    public Student(SNP snp, String groupNumber, int courseNumber, int totalWorksNumber, int finishedWorksCounter, String computerLanguage) {
        this.snp = snp;
        this.groupNumber = groupNumber;
        this.courseNumber = courseNumber;
        this.totalWorksNumber = totalWorksNumber;
        this.finishedWorksCounter = finishedWorksCounter;
        this.computerLanguage = computerLanguage;
    }

    public int getCourseNumber() { return courseNumber; }

    public String getAlignSnp() { return snp.getSurname() + " " + snp.getName() + " " + snp.getPatronymic(); }

    public void setCourseNumber(int courseNumber) { this.courseNumber = courseNumber; }

    public SNP getSnp() { return snp; }

    public void setSnp(SNP snp) { this.snp = snp; }

    public String getComputerLanguage() { return computerLanguage; }

    public void setComputerLanguage(String computerLanguage) { this.computerLanguage = computerLanguage; }

    public String getGroupNumber() { return groupNumber; }

    public void setGroupNumber(String groupNumber) { this.groupNumber = groupNumber; }

    public int getFinishedWorksCounter() { return finishedWorksCounter; }

    public void setFinishedWorksCounter(int finishedWorksCounter) { this.finishedWorksCounter = finishedWorksCounter; }

    public int getTotalWorksNumber() { return totalWorksNumber; }

    public void setTotalWorksNumber(int totalWorksNumber) { this.totalWorksNumber = totalWorksNumber; }

}
```

Рисунок 5 Класс Student

Класс контроллера:

```
public class Controller {
    private Model model;

    public Controller(Model model) { this.model = model; }

    public List<Student> getStudentList() { return model.getStudentList(); }

    public void newDoc(int entitiesNumber) { this.model = new Model(entitiesNumber); }

    public void addStudent(String surname, String name, String patronymic, String groupNumber, int courseNumber, int totalWorksNumber, int finishedWorksCounter, String computerLanguage) {
        model.addStudent(new Student(new SNP(surname, name, patronymic), groupNumber, courseNumber, totalWorksNumber, finishedWorksCounter, computerLanguage));
    }

    public void openDoc(File file) {
        try {
            model.setStudentList(DocOpener.openDoc(file));
        } catch (Exception exception) {
            exception.printStackTrace();
            return;
        }
    }

    public void saveDoc(File file) {
        List<Student> studentList = model.getStudentList();
        Element students = student; for group course totalWorks finishedWorks computerLanguage
```

Рисунок 6 Класс контроллера

Класс отвечающий за представление данных:

```
public class View {
    private final String REGEX_DIGITS_ONLY = "\\d+$";
    private Scene scene;
    private TableElement tableElement;
    private Controller controller;
    private Stage stage;
    private VBox root;

    private enum WindowType {
        DELETE, SEARCH
    }

    public View(Controller controller) {
        final int STAGE_WIDTH = 1460, STAGE_HEIGHT = 781;
        final String STAGE_TITLE_TEXT = "LABA_2";

        this.controller = controller;
        initWindow();
        stage = new Stage();
        stage.setWidth(STAGE_WIDTH);
        stage.setHeight(STAGE_HEIGHT);
        stage.setTitle(STAGE_TITLE_TEXT);
        stage.getIcons().add(new Image(url: "graphicalInterface/icon.jpg"));
        stage.setScene(scene);
    }
}
```

Рисунок 7 Класс View

Класс представляющий данные в виде таблицы:

```
public class TableElement {
    private int rowsOnPage, currentPage = 1, numberOfPages;
    private Label paginationLabel, itemCountLabel;
    private Button resetSearchButton;
    private TextField rowsOnPageField;
    private TableView<Student> table;
    private ToolBar navigator, pagination;
    private Pane tableElement;
    private List<Student> defaultStudentList;
    private ObservableList<Student> studentObsList, curStudentObsList;

    public TableElement(List<Student> studentList) {
        final int TABLE_HEIGHT = 500,
                TABLE_WIDTH = 1460,
                DEFAULT_ROWS_ON_PAGE_NUMBER = 15;

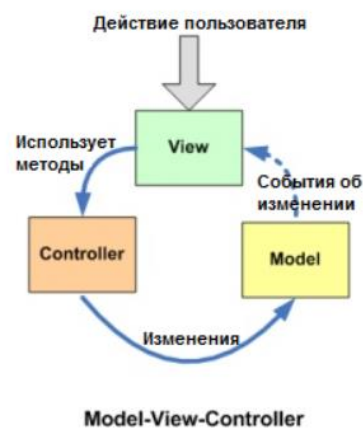
        final String SNP_COLUMN_LABEL_TEXT = "ФИО студента",
                GROUP_COLUMN_LABEL_TEXT = "Группа",
                COURSE_COLUMN_LABEL_TEXT = "Курс",
```

Рисунок 8 Класс таблицы

Ход работы

Приложение работает по принципу MVC представленному графически ниже. Пользователь выполняет какое-либо действие над интерфейсом (напр. нажимает кнопку) View обрабатывает это событие в зависимости конкретного действия пользователя используя методы Controller. Controller в свою очередь изменяет данные Model и после этого измененные данные снова отображаются View.

Model-View-Controller



Вывод

В данной лабораторной работе было реализовано оконное приложение с помощью библиотеки JavaFx без использования редактора форм. Приложение имеет главное окно и несколько дочерних диалогов: диалога добавления, удаления, поиска, сохранения, загрузки и создания нового документа. Приложение построено при помощи модели проектирования Model-View-Controller и полностью соответствует всем заявленным требованиям.