**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра вычислительной техники**

# ОТЧЕТ

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: Обработка событий**

Студент гр. 9308 _____ Яловега Н.В.

Преподаватель _____ Гречухин М.Н.

Санкт-Петербург

2021

**Цель работы**

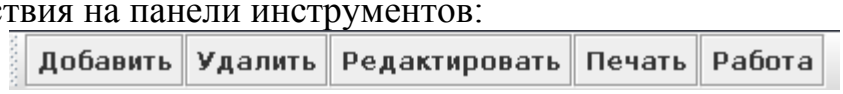
Знакомство с правилами обработки действий пользователя(событий), реализация логики приложения с использованием библиотек java.awt и java.swing.

**Описание возможных действий**

Для реализации обработки событий интерфейс программы из лабораторной работы №3 был расширен: добавлены диалоговые окна Добавления и Редактирования элементов, подтверждающие окна, окна сообщений.

Экранная форма предлагает действия такие как:

- Действия на панели инструментов:




*Добавить* – вызывает вспомогательное окно-диалог, помогающее задать параметры нового элемента
*Удалить* – удаляет элемент, соответствующий выбранной в таблице строке
*Изменить* – вызывает аналогичное окно-диалог, для редактирования элемента требуется выбрать соответствующую строку таблицы и после нажать на кнопку
*Печать* – экспорт активной таблицы в форматы PDF, XML
*Работа* – получение таблицы работ

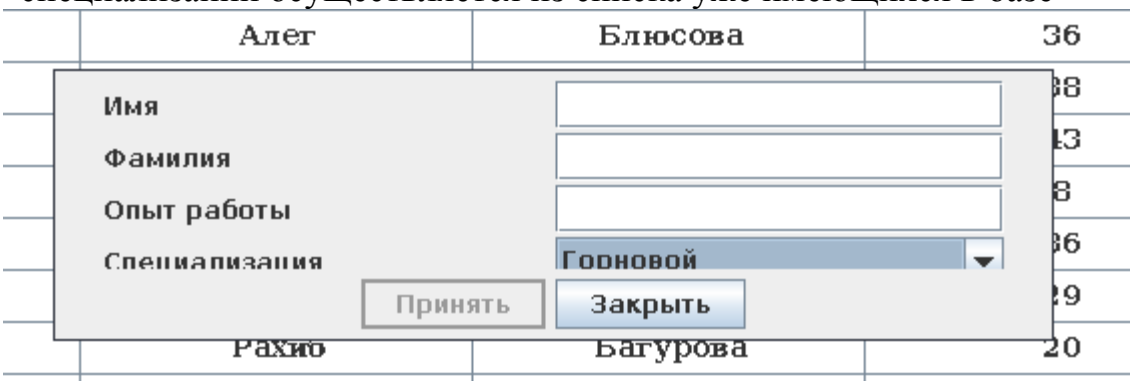
- Действия на панели поиска


*Строка ввода текста* – текст, который будет введен с строку будет найден в таблице
*Поиск* – осуществляет поиск по таблице

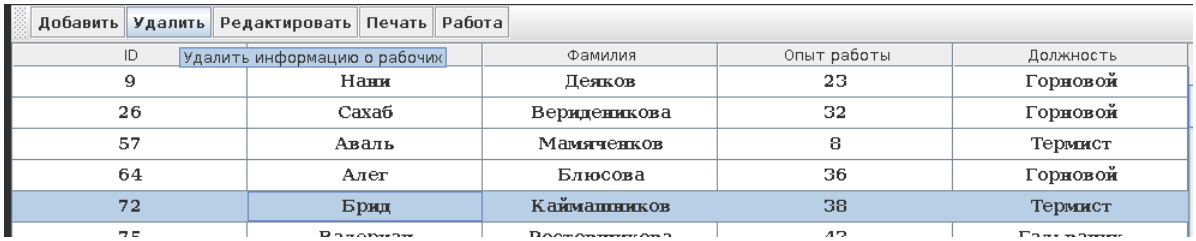
## Реализация

Для демонстрации будут представлены только три различных действия, полная реализации будет в прилагаемом файле с программой.

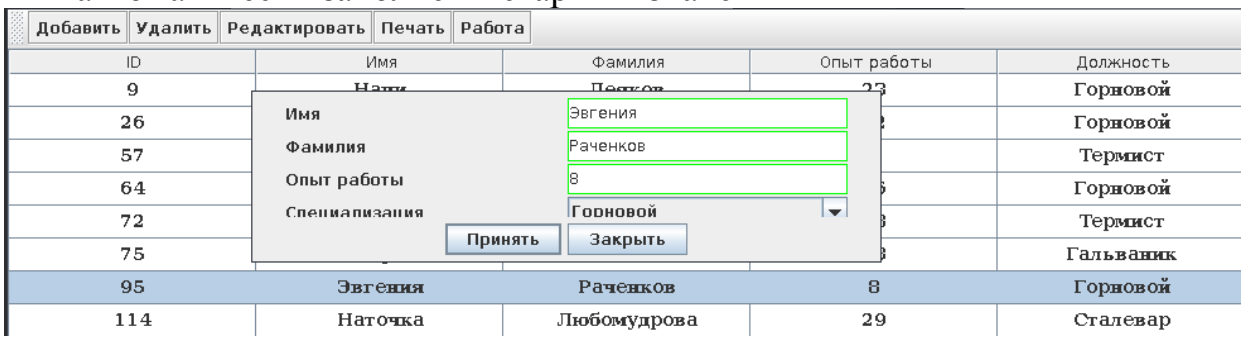
1) **Добавление элемента** вызывает модальное окно с заданными масками(шаблонами) ввода для избегания некорректного ввода. Выбор специализайии осуществляется из списка уже имеющихся в базе




2) Удаление элемента осуществляется путём выбора строки в таблице с удаляемым объектом:




3) **Изменение элемента** вызывает модальное окно с заданными масками(шаблонами) ввода для избегания некорректного ввода. Выбор специализайии осуществляется из списка уже имеющихся в базе. Поля автоматически заполнены старыми значениями



**Исходный код программы со слушателями**

```
package Factory.gui;

import Factory.exceptions.EmptyFileException;
import Factory.model.*;
import Factory.service.*;

import Factory.util.ReportUtil;
import org.w3c.dom.*;

import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.io.*;
import java.util.List;
import java.util.logging.*;

/** Класс приложения, визуализирующий экранную форму с рабочими */
public class WorkerWindow
{
    /** Стандартный конструктор */
    WorkerWindow()
    {
        show();
    }

    /** Окно приложения */
    private JFrame window;

    /** Модель таблицы */
    private DefaultTableModel model;

    /** Добавить */
    private JButton add;

    /** Удалить */
    private JButton delete;
```

```
    /** Изменить */
    private JButton edit;

    /** Печать */
    private JButton print;

    /** Работа */
    private JButton work;

    /** Панель инструментов */
    private JToolBar toolBar;

    /** Таблица */
    protected JTable dataWorkers;

    /** Поле поискового запроса */
    private JTextField textSearch;

    /** Поиск */
    private JButton search;

    /** Скролл */
    private JScrollPane scroll;

    /** Сервис Рабочего */
    private EmployeeService employeeService = new
EmployeeService();

    /** Сервис Профессий */
    private SpecialisationService specialisationService = new
SpecialisationService();

    /** Диалог добавления данных */
    private AddDialogWorker addDialogWorker;

    /** Диалог измения данных */
    private EditDialogWorker editDialogWorker;

    /** Метод отображения окна */
    public void show(){
        log.info("Открытие окна WorkerWindow");
        window = new JFrame("factory: Список рабочих завода");
        window.setSize(1000,500);
        window.setLocation(310,130);
        window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        // Создание кнопок и прикрепление иконок
        log.info("Добавление кнопок к окну WorkerWindow");
        add = new JButton("Добавить");
        delete = new JButton("Удалить");
        edit = new JButton("Редактировать");
```

```
        print = new JButton("Печать");
        work = new JButton("Работа");

        // Настройка подсказок
        add.setToolTipText("Добавить информацию о рабочих");
        delete.setToolTipText("Удалить информацию о рабочих");
        edit.setToolTipText("Изменить информацию о рабочих");
        print.setToolTipText("Распечатать информацию о рабочих");
        work.setToolTipText("Показать договоры, которые выполняют
рабочие");
        // Добавление кнопок на панель инструментов
        toolBar = new JToolBar("Панель инструментов");
        toolBar.add(add);
        toolBar.add(delete);
        toolBar.add(edit);
        toolBar.add(print);
        toolBar.add(work);
        // Размещение панели инструментов
        window.setLayout(new BorderLayout());
        window.add(toolBar,BorderLayout.NORTH);
        // Создание таблицы с данными
        log.info("Добавление таблицы с данными к окну
WorkerWindow");
        String[] columns = {"ID", "Имя", "Фамилия", "Опыт работы",
"Должность"};

        List<Employee> workersList = employeeService.findAll();
        String [][] data = new String[workersList.size()][5];
        for (int i = 0; i < workersList.size(); i++)
        {
            data[i] = workersList.get(i).toTableFormat();
        }

        // Настройка таблицы
        model = new DefaultTableModel(data, columns)
        {
            public boolean isCellEditable(int rowIndex, int
columnIndex)
            {
                return false;
            }
        };
        this.dataWorkers = new JTable(model);
        RowSorter<TableModel> sort = new
TableRowSorter<TableModel>(model);
        dataWorkers.setRowSorter(sort);
        dataWorkers.setFont(new Font(Font.SERIF,Font.BOLD,14));
        dataWorkers.setIntercellSpacing(new Dimension(0,1));
        dataWorkers.setRowHeight(dataWorkers.getRowHeight()+10);

dataWorkers.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
```

```
dataWorkers.setDefaultRenderer(dataWorkers.getColumnClass(1), new
DefaultTableCellRenderer(){
            public Component getTableCellRendererComponent(JTable
table, Object value, boolean isSelected, boolean hasFocus, int
row, int column) {

super.setHorizontalAlignment(SwingConstants.CENTER);
                super.getTableCellRendererComponent(table, value,
isSelected, hasFocus, row, column);
                return this;
            }

        });

        scroll = new JScrollPane(this.dataWorkers);

        // Размещение таблицы с данными
        window.add(scroll,BorderLayout.CENTER);

        // Подготовка компонентов поиска
        textSearch = new JTextField();
        textSearch.setColumns(20);
        search = new JButton("Поиск");
        window.getRootPane().setDefaultButton(search);
        // remove the binding for pressed

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
        // retarget the binding for released

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
        // Добавление компонентов на панель
        JPanel searchPanel = new JPanel();
        searchPanel.add(textSearch);
        searchPanel.add(search);

        // Размещение панели поиска внизу окна
        window.add(searchPanel,BorderLayout.SOUTH);

        // Если не выделена строка, то прячем кнопки

dataWorkers.getSelectionModel().addListSelectionListener((e) -> {
            boolean check = !
dataWorkers.getSelectionModel().isSelectionEmpty();
            edit.setVisible(check);
```

```
            delete.setVisible(check);
            work.setVisible(check);
        });

        add.addActionListener((e) ->
        {
            log.info("Старт Add listener");
            addDialogWorker = new AddDialogWorker(window,
WorkerWindow.this, "Добавление записи");
            addDialogWorker.setVisible(true);
        });

        add.setMnemonic(KeyEvent.VK_A);
        delete.addActionListener((e) -> {
            log.info("Старт Delete listener");
            if (dataWorkers.getRowCount() > 0) {
                if (dataWorkers.getSelectedRow() != -1) {
                    try {

employeeService.delete(Integer.parseInt(dataWorkers.getValueAt(dat
aWorkers.getSelectedRow(), 0).toString()));

model.removeRow(dataWorkers.convertRowIndexToModel(dataWorkers.get
SelectedRow()));
                        JOptionPane.showMessageDialog(window, "Вы
удалили строку");
                        log.info("Была удалена строка данных");
                    } catch (Exception ex) {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку для удаления");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего удалять");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });

        delete.setMnemonic(KeyEvent.VK_D);

        edit.addActionListener((e)-> {
            log.info("Старт Edit listener");
            if (model.getRowCount() != 0) {
```

```
                if (dataWorkers.getSelectedRow() != -1) {
                    t1 = new Thread(() -> {
                        JOptionPane.showMessageDialog(null,"1
поток запущен");
                        editDialogWorker = new
EditDialogWorker(window, WorkerWindow.this, "Редактирование");
                        editDialogWorker.setVisible(true);
                    });
                    t1.start();
                } else {
                    JOptionPane.showMessageDialog(null, "Не
выбрана строка. Нечего редактировать");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего редактировать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });
        edit.setMnemonic(KeyEvent.VK_E);

        print.addActionListener((e)->{
            log.info("Старт Print listener");
            t2 = new Thread(() -> {
                try {
                    JFileChooser fileChooser = new JFileChooser();
                    fileChooser.setDialogTitle("Select where you
want to save the report");
                    FileFilter pdf = new
FileNameExtensionFilter("PDF file(.pdf)", "pdf");
                    fileChooser.addChoosableFileFilter(pdf);
                    fileChooser.setCurrentDirectory(new
File("."));
                    String resultpath = "reportWorkers.pdf";
                    int returnVal =
fileChooser.showSaveDialog(null);
                    if(returnVal == JFileChooser.APPROVE_OPTION)
                    {
                        File file = fileChooser.getSelectedFile();
                        resultpath = file.getAbsolutePath();
                    }
                    checkList();
                    makeXml();
                    ReportUtil.print("dataWorkers.xml",
"window/dataWorkers", "workers.jrxml", resultpath);
                    JOptionPane.showMessageDialog(null, "2 поток
закончил работу. Отчет создан");
                } catch (Exception ex) {
```

```
                    JOptionPane.showMessageDialog(null, "Ошибка: "
+ ex.toString());
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            });
            t2.start();
        });

        work.addActionListener((e) -> {
            log.info("Старт work listener");
            if (dataWorkers.getRowCount() > 0) {
                if (dataWorkers.getSelectedRow() != -1) {
                    try
                    {
                        new
WorkerContractWindow(Integer.parseInt(dataWorkers.getValueAt(dataW
orkers.getSelectedRow(), 0).toString()));
                    }
                    catch (Exception ex)
                    {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка");
                }
            }
            else
            {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
        });


        search.addActionListener((e) -> {
            if (model.getRowCount() != 0) {
                if (!textSearch.getText().isEmpty())
                    log.info("Запуск нового поиска по ключевому
слову: " + textSearch.getText());
                else
                    log.info("Сброс ключевого слова поиска");
                TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(((DefaultTableModel) model));
```

```
                sorter.setStringConverter(new
TableStringConverter() {
                    @Override
                    public String toString(TableModel model, int
row, int column) {
                        return model.getValueAt(row,
column).toString().toLowerCase();
                    }
                });
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" +
textSearch.getText().toLowerCase()));
                dataWorkers.setRowSorter(sorter);
            }
        });

        window.setVisible(true);
    }


    /** Метод загрузки данных в XML файл */
    public void makeXml()
    {

    }

    /**
     * Вспомогательный метод добавления данных в таблицу
     * @param arr - данные, полученные от пользователя
     */
    public void addR(String[] arr)
    {
        Employee newW = new Employee(arr[0], arr[1],
Integer.parseInt(arr[2]),
specialisationService.findByName(arr[3]));
        employeeService.persist(newW);
        model.addRow(newW.toTableFormat());
    }

    /**
     * Вспомогательный метод изменения данных в таблице
     * @param arr - данные, полученные от пользователя
     */
    public void editR(String[] arr)
    {
        Employee W =
employeeService.findById(Integer.parseInt(arr[0]));
        W.setName(arr[1]);
        W.setSurname(arr[2]);
        W.setExp(Integer.parseInt(arr[3]));

W.setSpecialisation(specialisationService.findByName(arr[4]));
```

```
            employeeService.update(W);
        }
    }
```

## Выводы

При выполнении лабораторной работы была реализована обработка действий пользователя при помощи библиотек java.awt и java.swing. Также был расширен интерфейс программы, реализованной в третьей лабораторной работе.