

**Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
«Севастопольский государственный университет»**

**ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ РАЗРАБОТКИ
ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА В JAVA
ПРИЛОЖЕНИЯХ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ
SWING**

Методические указания для студентов, обучающихся по
направлениям
09.03.02 – «Информационные системы и технологии»,
09.03.03 – «Прикладная информатика»
по учебным планам подготовки бакалавров
дневной и заочной форм обучения

**Севастополь
2015**

УДК 004.42 (075.8)

Исследование возможностей разработки пользовательского интерфейса в Java приложениях с использованием библиотеки Swing: методические указания к лабораторной работе №4 по дисциплине “Платформа Java” для студентов / Сост. **С.А. Кузнецов, А.Л. Овчинников** — Севастополь: Изд-во СевГУ, 2015. — 25 с.

Цель указаний: оказание помощи студентам при выполнении лабораторной работы №4 по дисциплине “Платформа Java”.

Методические указания составлены в соответствии с требованиями программы дисциплины «Платформа Java» для студентов дневной и заочной формы обучения и утверждены на заседании кафедры «Информационные системы» протоколом № 1 от 31 августа 2015 года.

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: Кожаяев Е.А., канд. техн. наук, доцент кафедры кибернетики и вычислительной техники.

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ	4
2. ПОСТАНОВКА ЗАДАЧИ.....	4
3. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.....	5
3.1 Общие сведения о библиотеке Swing.....	5
3.2. Основные компоненты библиотеки Swing	6
3.3. Менеджеры компоновки в Swing	8
3.4. События и обработка событий компонентов Swing	11
3.5. Модели компонентов Swing	12
3.6. Использование диалога выбора файла	14
3.7. Использование WindowBuilder для создания интерфейса в Eclipse	14
4. ВАРИАНТЫ ЗАДАНИЙ	17
5. СОДЕРЖАНИЕ ОТЧЕТА.....	18
6. КОНТРОЛЬНЫЕ ВОПРОСЫ	18
БИБЛИОГРАФИЧЕСКИЙ СПИСОК	18
ПРИЛОЖЕНИЕ А	19
А.1 Разработка графического интерфейса приложения.....	19
А.2 Текст программы.....	20

1. ЦЕЛЬ РАБОТЫ

В ходе выполнения данной лабораторной работы необходимо ознакомиться с особенностями инструментария библиотеки SWING для создания графического интерфейса приложений на языке Java и приобрести практические навыки создания Java-программ с графическим интерфейсом, позволяющим пользователю осуществлять взаимодействие с программой: задавать исходные данные, просматривать результаты работы программы в удобном виде.

2. ПОСТАНОВКА ЗАДАЧИ

Необходимо создать Java приложение с графическим интерфейсом пользователя, реализующее добавление, редактирование, сортировку и удаление данных заданного по варианту типа информации Т(см. табл. 4.1). Данные отображать в виде таблицы. Реализовать поля ввода для добавления и редактирования новых записей. Предусмотреть возможность загрузки информации из текстового файла и сохранения в текстовый файл.

При написании программы следует учесть следующие требования и рекомендации:

1. Создать публичный класс, представляющий заданный по варианту задания (см. табл. 4.1) тип информации (т.е. строку таблицы).
2. Создать модель данных таблицы. Для этого создать класс, расширяющий абстрактный класс `AbstractTableModel`. Создать в нем объект коллекции типа Т, соответствующий варианту задания.

Переопределить методы:

- *`public Class<?> getColumnClass(int columnIndex)`*
- *`public int getColumnCount()`*
- *`public String getColumnName(int columnIndex)`*
- *`public Object getValueAt(int rowIndex, int columnIndex)`*
- *`public boolean isCellEditable(int rowIndex, int columnIndex)`*
- *`public void setValueAt(Object value, int rowIndex, int columnIndex)`*

Определить методы:

- *`public void addRow(<объект>) – добавления элемента (строки)`*
 - *`public void deleteRow(String Поле_1) – удаления элемента по значению поля 1`*
 - *`public void updateRow(int row, <объект>) – изменения элемента заданной строки)`*
3. Для реализации окна приложения реализовать дочерний класс `JFrame`.
 4. Для представления таблицы с данными использовать компонент класса `JTable`, разместив его в контейнере `JScrollPane` (для возможности добавления полос прокрутки).
 5. Поля ввода для добавления и редактирования данных реализовать текстовыми компонентами `JTextField`. Каждое поле снабдить подписью при помощи компонентов `JLabel`.

6. Для выполнения действий открытия файла, добавления, изменения, удаления записи, сортировки и сохранения файла реализовать соответствующие кнопки, с использованием компонентов `JButton` и добавлением `ActionListener`. Реализовать загрузку записи в поля для редактирования при щелчке по строке таблицы.
7. Для выбора файла при открытии и сохранении использовать компонент `JFileChooser`.
8. Удаление и сортировка элементов должно проходить по ключевому полю `P` (см. табл. 4.1). Направление сортировки `U` (см. табл. 4.1).

3. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

3.1 Общие сведения о библиотеке `Swing`

Swing — это библиотека для создания графического интерфейса пользователя -представляет собой набор классов, применяемых для создания графических пользовательских интерфейсов (Graphical User Interface — GUI) современных приложений, в том числе Web-приложений.

В сравнении с использовавшейся в ранних версиях Java библиотекой `AWT`, библиотека **Swing** имеет ряд преимуществ:

- богатый набор интерфейсных примитивов;
- настраиваемый внешний вид на различных платформах (look and feel);
- раздельная архитектура модель-вид (model-view);
- встроенная поддержка HTML.

`Swing` предоставляет богатый набор визуальных компонентов, например, кнопок, полей редактирования, полос прокрутки, переключателей, таблиц и т.п. Посредством **Swing** можно разработать интерфейс приложения, удовлетворяющий потребностям пользователей, на профессиональном уровне.

Архитектура **MVC** (Model-View-Controller — *Модель-Представление-Контроллер*):

- *Модель* задает состояние компонента.
- *Представление* определяет, как компонент будет отображаться на экране, в том числе как будет представлено текущее состояние модели.
- *Контроллер* обеспечивает реакцию компонента на действия пользователя.

В **Swing** используется модифицированный вариант **MVC**, в котором представление и контроллер объединены в единый элемент, называемый *представителем пользовательского интерфейса*. Подход, используемый в **Swing**, известен как архитектура "*Модель-Представитель*" или архитектура с выделенной моделью.

В состав графического пользовательского интерфейса, созданного средствами `Swing`, входят элементы двух типов: *компоненты* и *контейнеры*. *Компоненты* — это независимые элементы (например, кнопки или линейные регуляторы). *Контейнер*

может содержать в себе несколько компонентов и представляет собой специальный тип компонента. Чтобы компонент отобразился на экране, его надо поместить в контейнер.

Таким образом, в составе графического пользовательского интерфейса должен присутствовать хотя бы один контейнер. Поскольку контейнеры являются компонентами, один контейнер может находиться в составе другого. Это позволяет формировать так называемую *иерархию контейнеров*, на вершине которой должен находиться *контейнер верхнего уровня*. Контейнер верхнего уровня не может включаться в состав других контейнеров. Более того, любая иерархия должна начинаться именно с контейнера верхнего уровня. В приложениях для этой цели чаще всего используется объект ***JFrame***, в апплетах — ***JApplet*** (Java-апплет — прикладная программа Java, встраиваемая в Web-страницы и выполняемая в веб-обозревателе с использованием виртуальной машины Java).

3.2. Основные компоненты библиотеки Swing

Классы, представляющие все компоненты и контейнеры **Swing**, содержатся в пакете *javax.swing*.

Базовым строительным блоком всей библиотеки визуальных компонентов Swing является ***JComponent***. Это суперкласс каждого компонента. Он является абстрактным классом и содержит множество функций, которые реализуют компоненты **Swing**.

JFrame — контейнер, предназначенный для размещения на нем других компонентов. ***JFrame*** регистрируется в ОС как окно и получает многие из знакомых свойств окна операционной системы: минимизация/максимизация, изменение размеров и перемещение.

Основные методы класса ***JFrame***:

- **get/setTitle()** - Получить/установить заголовок фрейма.
- **get/setState()** - Получить/установить состояние фрейма (минимизировать, максимизировать и т.д.).
- **is/setVisible()** - Получить/установить видимость фрейма, другими словами, отображение на экране.
- **get/setLocation()** - Получить/установить месторасположение в окне, где фрейм должен появиться.
- **get/setSize()** - Получить/установить размер фрейма.
- **add()** - Добавить компоненты к фрейму.

JLabel — компонент, позволяющий отобразить текст с иконкой.

Основные методы класса ***JLabel***:

- **get/setText()** - Получить/установить текст в метке.
- **get/setIcon()** - Получить/установить изображение в метке.
- **get/setHorizontalAlignment()** - Получить/установить горизонтальную позицию текста.
- **get/setVerticalAlignment()** - Получить/установить вертикальную позицию текста.

- **get/setDisplayedMnemonic()** - Получить/установить мнемонику (подчеркнутый символ) для метки.
- **get/setLabelFor()** - Получить/установить компонент, к которому присоединена данная метка; когда пользователь нажимает комбинацию клавиш Alt+мнемоника, фокус перемещается на указанный компонент.

JButton — компонент Swing, реализующий кнопку.

Основные методы класса **JButton**:

- **get/setText()** - Получить/ установить текст в кнопке.
- **get/setIcon()** - Получить/ установить изображение в кнопке.
- **get/setHorizontalAlignment()** - Получить/установить горизонтальную позицию текста.
- **get/setVerticalAlignment()** - Получить/установить вертикальную позицию текста.
- **get/setDisplayedMnemonic()** - Получить/установить мнемонику (подчеркнутый символ), которая в комбинации с кнопкой Alt вызывает нажатие кнопки.

JTextField — компонент Swing, реализующий строку ввода текста.

Основные методы класса **JTextField**:

get/setText() - Этот метод получает/устанавливает текст внутри **JTextField**.

JComboBox — комбинированный список — выпадающий список элементов, в котором пользователь может выбрать ноль или один (и только один) элемент.

Основные методы класса **JComboBox**:

- **addItem()** - Добавить элемент к **JComboBox**.
- **get/setSelectedIndex()** - Получить/установить индекс выбранного элемента в **JComboBox**.
- **get/setSelectedItem()** - Получить/установить выбранный объект.
- **removeAllItems()** - Удалить все объекты из **JComboBox**.
- **removeItem()** - Удалить конкретный объект из **JComboBox**.

Компоненты **JCheckBox** и **JRadioButton** предоставляют пользователю переключатели для выбора. **JRadioButton** обычно группируются вместе для предоставления пользователю возможности выбора только одного варианта из нескольких представленных. Как только вы выбрали **JRadioButton**, вы не можете снять его отметку до тех пор, пока не выберете другой вариант из группы. **JCheckBox** работает иначе — он позволяет отмечать/снимать отметку с варианта независимо от состояния других переключателей **JCheckBox**.

Классом, который позволяет группировать вместе компоненты **JCheckBox** или **JRadioButton**, является класс **ButtonGroup**. Он позволяет группировать варианты **JRadioButton** таким образом, что при выборе одного, с другого отметка автоматически снимается.

Основные методы класса **ButtonGroup**:

- **add()** - Добавить *JCheckBox* или *JRadioButton* к *ButtonGroup*.
- **getElements()** - Получить все компоненты в *ButtonGroup*, для того, чтобы можно было выполнить итерацию по ним для поиска выбранного.

Компонент *JTextArea* расширяет возможности компонента *TextField*. В то время как *TextField* ограничен лишь одной строкой текста, *JTextArea* позволяет пользователю вводить несколько строк.

Дополнительные методы класса *JTextArea* по сравнению с *TextField*:

- **is/setLineWrap()** - Устанавливает, должна ли переноситься строка, если она становится слишком длинной.
- **is/setWrapStyleWord()** - Устанавливает, должно ли переноситься слово на следующую строку, если оно слишком длинное.

JScrollPane — предоставляет Swing-компонент для обработки всех действий по прокрутке, если необходимая информация не может полностью отображаться в каком-либо объекте (например, текст в *JTextArea*).

JList является полезным компонентом для предоставления пользователю вариантов для множественного выбора. Необходимо использовать *JList* совместно с *JScrollPane*, поскольку он может предоставлять больше вариантов, чем помещается в видимой области.

Основные методы класса *JList*:

- **get/setSelectedIndex()** - Получить/установить выбранную строку списка; в случае со списками с множественным выбором возвращается `int[]`.
- **get/setSelectionMode()** - Получить/установить режим выбора в одиночный выбор, одиночный интервал или множественный интервал.
- **setListData()** - Установить данные для использования в *JList*.
- **get/setSelectedValue()** - Получить выбранный объект.

JOptionPane — компонент, позволяющий организовать диалоговое окно, предлагающее пользователю стандартные варианты ответа. Компонент позволяет изменять: заголовок фрейма, само сообщение, отображаемую пиктограмму, варианты кнопок и необходимость текстового ответа. Этот компонент также имеет множество различных вариантов применения, которые предлагается изучить самостоятельно используя среду разработки и документацию Java.

Пример вывода диалогового окна с вариантами выбора Да и Нет:

```
int reply = JOptionPane.showConfirmDialog(null, "Вы согласны?", "Заголовок
окна", JOptionPane.YES_NO_OPTION);
if (reply == JOptionPane.YES_OPTION){//если нажата кнопка Да
                                //выполнить действие
}
```

3.3. Менеджеры компоновки в Swing

Менеджеры компоновки используются для автоматического позиционирования и задания размеров дочерних элементов в контейнере.

Компоновщиком является любой объект реализующий интерфейс *LayoutManager* или *LayoutManager2* (поддерживает выравнивание и ограничения).

Стандартные компоновщики **Swing**:

BorderLayout - размещает элементы в один из пяти регионов, как было указано при добавлении элемента в контейнер: вверх, вниз, влево, вправо, в центр. По умолчанию элемент добавляется в центр. Если в указанном регионе уже есть элемент, то он замещается новым. Поэтому, когда надо разместить несколько элементов в одном регионе, то их объединяют в один контейнер (обычно *JPanel*).

По умолчанию в **Swing** используется менеджер **BorderLayout**. В нем определены следующие константы для установки компонентов.

- BorderLayout.NORTH (верх)
- BorderLayout.SOUTH (низ)
- BorderLayout.EAST (справа)
- BorderLayout.WEST (слева)
- BorderLayout.CENTER (заполнить середину до других компонент или до краев)

По умолчанию принимается константа Center.

Пример:

// создаем фрейм и устанавливаем его размер.

JFrame jf = new JFrame();

jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

jf.setSize(400, 300);

jf.setVisible(true);

// создаем панель.

JPanel p = new JPanel();

jf.add(p);

// к панели добавляем менеджер BorderLayout.

p.setLayout(new BorderLayout());

// к панели добавляем кнопку и устанавливаем для нее верхнее расположение.

p.add(new JButton("OK"), BorderLayout.NORTH);

FlowLayout - размещает элементы по порядку в том же направлении, что и ориентация контейнера (слева на право по умолчанию), применяя один из пяти видов выравнивания, указанного при создании менеджера.

Пример:

// создаем окно и устанавливаем его размер.

JFrame jf = new JFrame();

jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

jf.setSize(400, 300);

jf.setVisible(true);

```

        // создаем панель.
        JPanel p = new JPanel();
        jf.add(p);

        // к панели добавляем менеджер FlowLayout.
        p.setLayout(new FlowLayout());

        // к панели добавляем кнопки.
        p.add(new JButton("Кнопка 1"));
        p.add(new JButton("Кнопка 2"));
        p.add(new JButton("Кнопка 3"));
        p.add(new JButton("Кнопка 4"));

```

GridLayout - размещает элементы в виде таблицы. Количество столбцов и строк указывается при создании менеджера. По умолчанию одна строка, а число столбцов равно числу элементов. Вся область контейнера разбивается на ячейки и размер каждого элемента устанавливается в размер ячейки. Целесообразно использование для выравнивания панелей и других контейнеров, а не элементов управления (иначе можем получить огромные кнопки).

Пример:

```

        // создаем окно и устанавливаем его размер.
        JFrame jf = new JFrame();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setSize(400, 300);
        jf.setVisible(true);

        // создаем панель.
        JPanel p = new JPanel();
        jf.add(p);

        // к панели добавляем менеджер GridLayout
        // и устанавливаем размеры таблицы 3*3.
        p.setLayout(new GridLayout(3,3));

        // к панели добавляем кнопки.
        p.add(new JButton("Кнопка 1"));
        p.add(new JButton("Кнопка 2"));
        p.add(new JButton("Кнопка 3"));
        p.add(new JButton("Кнопка 4"));

```

BoxLayout — размещает элементы по вертикали или по горизонтали. Обычно он используется не напрямую, а через контейнерный класс **Box**.

GridBagLayout, как и **GridLayout** менеджер, устанавливает компоненты в таблицу, но он более гибок, так как предоставляет возможность определять для компонентов разную ширину и высоту колонок и строк таблицы. По существу,

GridBagLayout помещает компоненты в ячейки, и затем использует привилегированные размеры компонентов, чтобы определить, насколько большой ячейка должна быть.

Существуют и более сложные менеджеры компоновки такие как **CardLayout**, **GroupLayout**, **SpringLayout**.

Если в контейнере отсутствует компоновщик (был вызван метод контейнера `setLayout(null)`), то позицию и размеры элементов необходимо задать явно методами элемента

- `setLocation(Point p)` - переместить компонент в указанную точку;
- `setLocation(int x, int y)` - переместить компонент в указанную точку;
- `setSize(Dimension d)` - установить размеры компонента;
- `setSize(int width, int height)` - установить размеры компонента;
- `setBounds(Rectangle r)` - переместить и установить размеры компонента (вписать в четырехугольник);
- `setBounds(int x, int y, int width, int height)` - переместить и установить размеры компонента.

3.4. События и обработка событий компонентов Swing

В Swing используется механизм обработки событий, который называется *модель делегирования событий*. *Источник* генерирует событие, которое передается одному или нескольким *обработчикам*. В рамках данной схемы обработчики лишь ожидают возникновения события. При возникновении события они обрабатывают его и возвращают управление. Преимущество такого подхода в том, что логика обработки событий отделена от логики пользовательского интерфейса, генерирующего эти события. Элемент интерфейса "делегирует" обработку события отдельному фрагменту кода. В модели делегирования событий обработчик, чтобы получать оповещения о событиях, должен быть зарегистрирован в источнике.

Согласно модели делегирования, событие является объектом, описывающим изменения состояния источника. Событие может быть следствием действий пользователя с элементом графического интерфейса или сгенерировано программными средствами. Суперклассом всех событий является `java.util.EventObject`. Многие события объявлены в пакете `java.awt.event`, но некоторые содержатся в `javax.swing.event`.

Источник события — это объект, сгенерировавший его. Как отмечалось выше, сгенерировав событие, источник должен передать его всем зарегистрированным обработчикам. Следовательно, чтобы обработчик получил событие, он должен быть зарегистрирован в источнике. Регистрация осуществляется путем вызова метода **addTunListener()**, принадлежащего источнику (например, `addActionListener`, `addKeyListener`, `addMouseListener` и тп). Для каждого типа события определен собственный метод регистрации. Заголовок метода имеет вид, подобный представленному ниже:

```
public void addTunListener (TunListener el)
```

Источник должен также предоставлять метод, позволяющий отменить регистрацию обработчика событий определенного типа. Этот метод имеет заголовок, представленный в следующей форме:

```
public void removeTunListener(TunListener el)
```

Обработчик — это объект, оповещаемый о возникновении события. К нему предъявляются два основных требования. Во-первых, чтобы получать оповещение о конкретном типе событий, он должен быть зарегистрирован в одном или нескольких источниках. Во-вторых, он должен реализовывать метод, предназначенный для обработки события.

Методы, позволяющие получать и обрабатывать события, определены в интерфейсах, содержащихся в пакетах *java.awt.event*, *javax.swing.event* и *java.beans*. Например, в интерфейсе **ActionListener** объявлен метод, который вызывается тогда, когда пользователь щелкает на кнопке или выполняет другое действие, затрагивающее компонент. Это событие может быть обработано любым объектом, при условии, что он реализует интерфейс **ActionListener**.

Общие правила, которые следует учитывать при обработке событий, таковы: обработчик должен выполнять свою задачу быстро и возвращать управление. По возможности он не должен осуществлять сложные операции, поскольку это может замедлить работу всего приложения.

Пример класса-обработчика нажатий кнопки:

```
class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JButton source = (JButton) e.getSource(); // получаем источник сообщения
                                                    //(объект - кнопка)
        String buttonText = source.getText();      // получаем из кнопки текст
        // и показываем диалоговое окно с текстом нажатой кнопки
        JOptionPane.showMessageDialog(null, "Нажата кнопка " + buttonText);
    }
}
```

Пример назначения обработчика кнопке:

```
JFrame frame = new JFrame("Обработчик сообщений");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JButton button = new JButton("Кнопка Один");
button.addActionListener(new MyActionListener());
```

3.5. Модели компонентов Swing

Модель — это некий интерфейс с методами, который нужно реализовать, позволяющий визуальному компоненту правильно отображать реальные массивы данных.

Для сложных компонентов (списки, таблицы, деревья) и даже для некоторых более простых, например, **JComboBox**, модели - это самый эффективный способ работы с данными. Они убирают большую часть работы по обработке данных из

самого компонента и предоставляют оболочку для общих объектных классов данных (например, *Vector* и *ArrayList*).

Название интерфейсов, описывающих модели для различных типов компонентов имеют названия *TunModel*, где *Tun* – название класса данного компонента. Создание новой модели для данного компонента выполняется реализацией соответствующего модельного интерфейса. Кроме этого, в библиотеке Swing уже созданы классы, описывающие модели по умолчанию для различных компонентов, например *DefaultComboBoxModel*, *DefaultListModel*, *DefaultTableModel* и др.

Пример использования модели по умолчанию компонента *JList*:

```
DefaultListModel listModel = new DefaultListModel();
JList list = new JList(listModel);
```

```
// добавление элементов в модель
for (int i = 0; i < 25; i++) {
    listModel.addElement("Элемент списка " + i);
}
```

```
//удаление 5-го элемента из модели
int removeIndex = 5;
listModel.remove(removeIndex);
```

```
//создание объекта класса JList с указанием модели
JList list = new JList(listModel);
list.setSelectedIndex(0);
```

Модель *TableModel* позволяет полностью описать каждую ячейку таблицы *JTable*. Определенные в ней методы дают возможность таблице получить значение произвольной ячейки и изменить его (модель данных таблицы считается изменяемой, для этого не нужно проводить дополнительных действий или реализовывать новых интерфейсов), узнать о том, можно ли пользователю редактировать ячейки, получить исчерпывающую информацию о столбцах (их количестве, названиях и типе) и строках. Нужно определить эти методы и с их помощью передать таблице все подготовленные данные. Однако для базовых табличных операций бывает достаточно использовать модель по умолчанию *DefaultTableModel*.

Пример использования модели по умолчанию компонента *JTable*:

```
//создаем стандартную модель
private DefaultTableModel dtm;
dtm = new DefaultTableModel();
```

```
//задаем названия столбцов
dtm.setColumnIdentifiers(new String[] {"Номер", "Товар", "Цена"});
```

```
//наполняем модель данными
dtm.addRow(new String[]{"№1", "Блокнот", "5.5"});
dtm.addRow(new String[] {"№2", "Телефон", "175"});
dtm.addRow(new String[] {"№3", "Карандаш", "1.2"});

// передаем модель в таблицу
JTable table = new JTable(dtm);

// удаляем последнюю строку (отсчет с нуля)
dtm.removeRow(dtm.getRowCount()-1);
```

3.6. Использование диалога выбора файла

При работе с файлами из приложения возникает необходимость использовать диалог для выбора файлов. Компонент `JFileChooser` из библиотеки `Swing` как раз реализует такой диалог.

Для создания окна диалога выбора файла необходимо вызвать метод `showDialog` экземпляра `JFileChooser`. В случае если пользователь успешно выберет файл, метод `showDialog` вернет значение равное константе `JFileChooser.APPROVE_OPTION`, а выбранный файл возможно будет получить с использованием метода `getSelectedFile()`. В примере ниже показано использование `JFileChooser` для выбора файла при нажатии пользователем кнопки "Открыть файл":

```
JButton btnFileOpen = new JButton("Открыть файл");
btnFileOpen.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JFileChooser fileopen = new JFileChooser();
        int ret = fileopen.showDialog(jfMyFrame.this, "Открыть
файл");

        if (ret == JFileChooser.APPROVE_OPTION) {
            File file = fileopen.getSelectedFile();
            // выполнение действия над файлом
            ...
        }
    }
});
```

3.7. Использование WindowBuilder для создания интерфейса в Eclipse

`WindowBuilder` является мощным и простым в использовании Java GUI дизайнером. Он состоит из конструкторов `SWT`, **Swing** и `GWT` и дает возможность очень легко создать Java-приложений с графическим интерфейсом, не тратя много времени на написание кода для отображения простых форм. В *WindowBuilder Pro* можно создавать сложные окна в считанные минуты. Можете легко добавлять

элементы управления с перетаскиванием курсором, добавлять обработчики событий к элементам управления, делать интернационализацию вашего приложения и многое другое.

3.7.1 Установка WindowBuilder

WindowBuilder Pro распространяется как плагин для *Eclipse* и включен в дистрибутив *Eclipse IDE for Java Developers*. Если же используется другой дистрибутив, можно установить *WindowBuilder Pro* непосредственно из среды *Eclipse*. Необходимо в меню *Help* выбрать пункт *Install New Software*. Далее в появившемся диалоговом окне (см. рис. 3.1) в графе *Work with* указать ссылку соответствующую вашей версии *Eclipse*. Ссылки для установки *WindowBuilder* можно найти по адресу <http://www.eclipse.org/windowbuilder/download.php>. Далее нужно выбрать все предлагаемые компоненты (см. рис. 3.1) и жать кнопку *Next* до появления кнопки *Finish*, нажав на которую, мы запустим установку *WindowBuilder*. При этом необходимо, чтобы компьютер имел доступ в сеть Интернет.

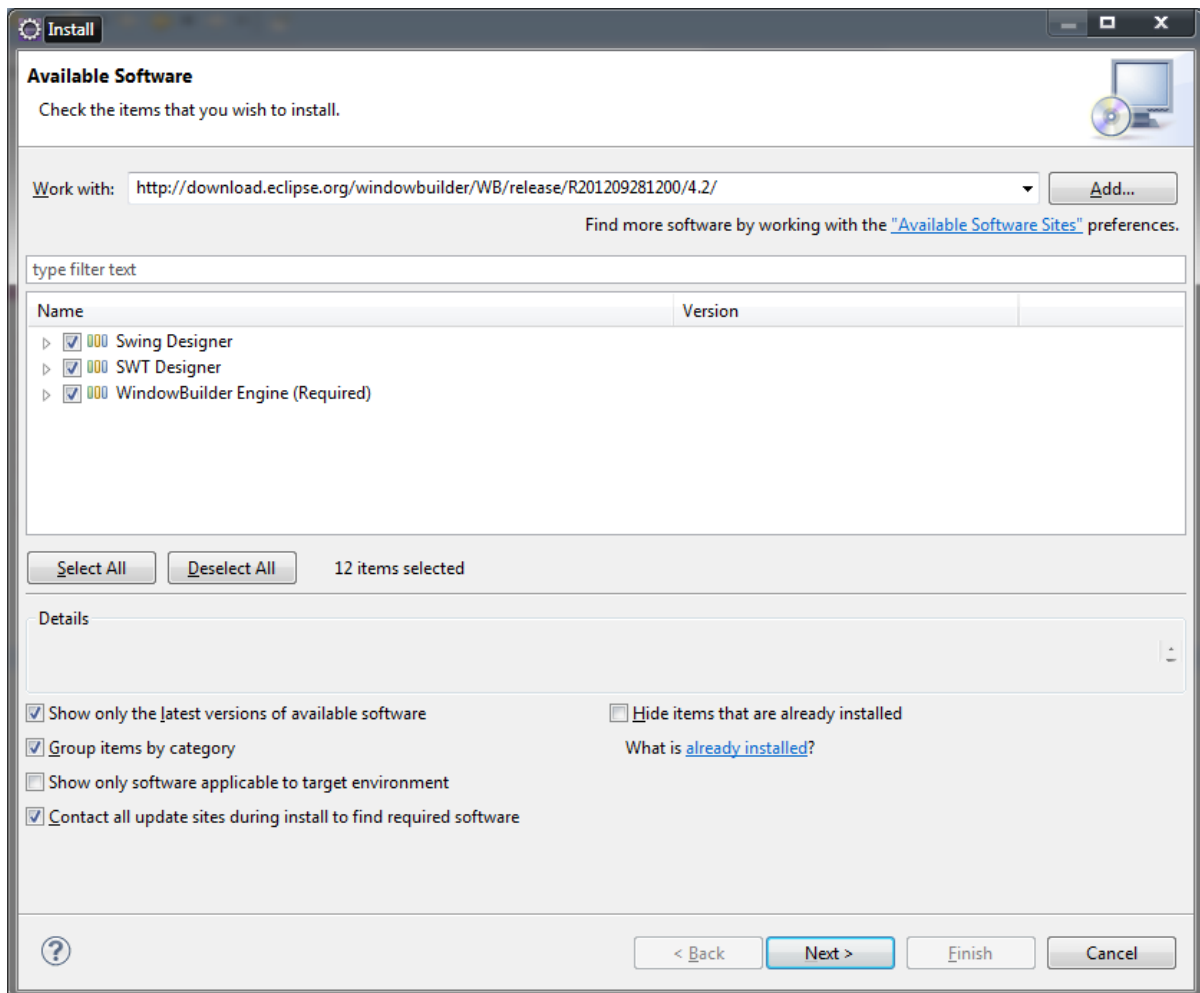


Рисунок 3.1 — Установка WindowBuilder

3.7.2 Создание Swing проекта в Eclipse

После установки *WindowBuilder* в меню инструментов рядом с кнопкой *New* появится кнопка с выпадающим меню *Create new visual classes*. Необходимо в выпадающем меню выбрать раздел *Swing* далее выбрать необходимый класс, например *JFrame*. Далее в появившемся окне нужно задать имя класса и закончить создание класса нажатием кнопки *Finish*. После этого на вкладке *Source* откроется окно исходного кода класса, созданного автоматически. Для добавления компонентов, управления их размещением, редактирования их свойств и т.д. необходимо выбрать вкладку *Design*.

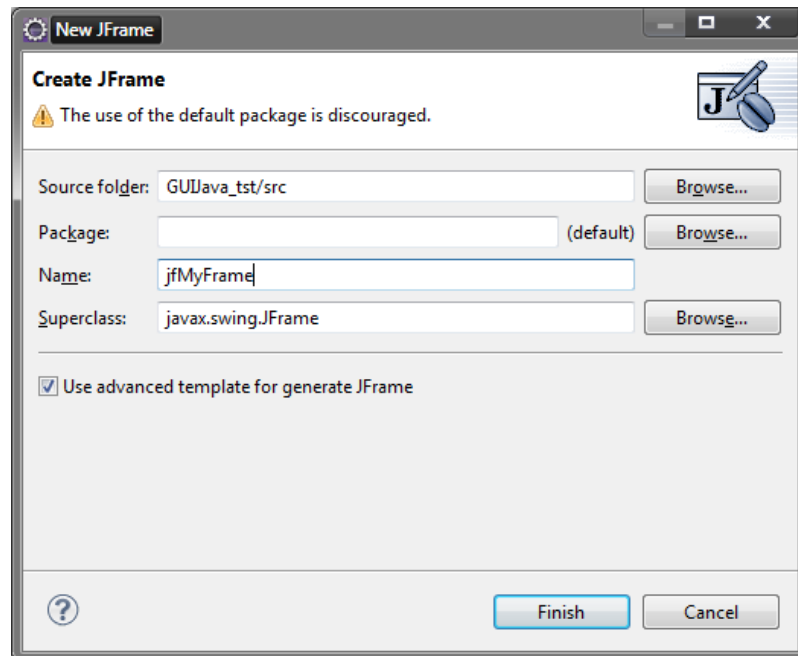


Рисунок 3.2 — Создание класса *jfMyFrame* на базе класса *JFrame*

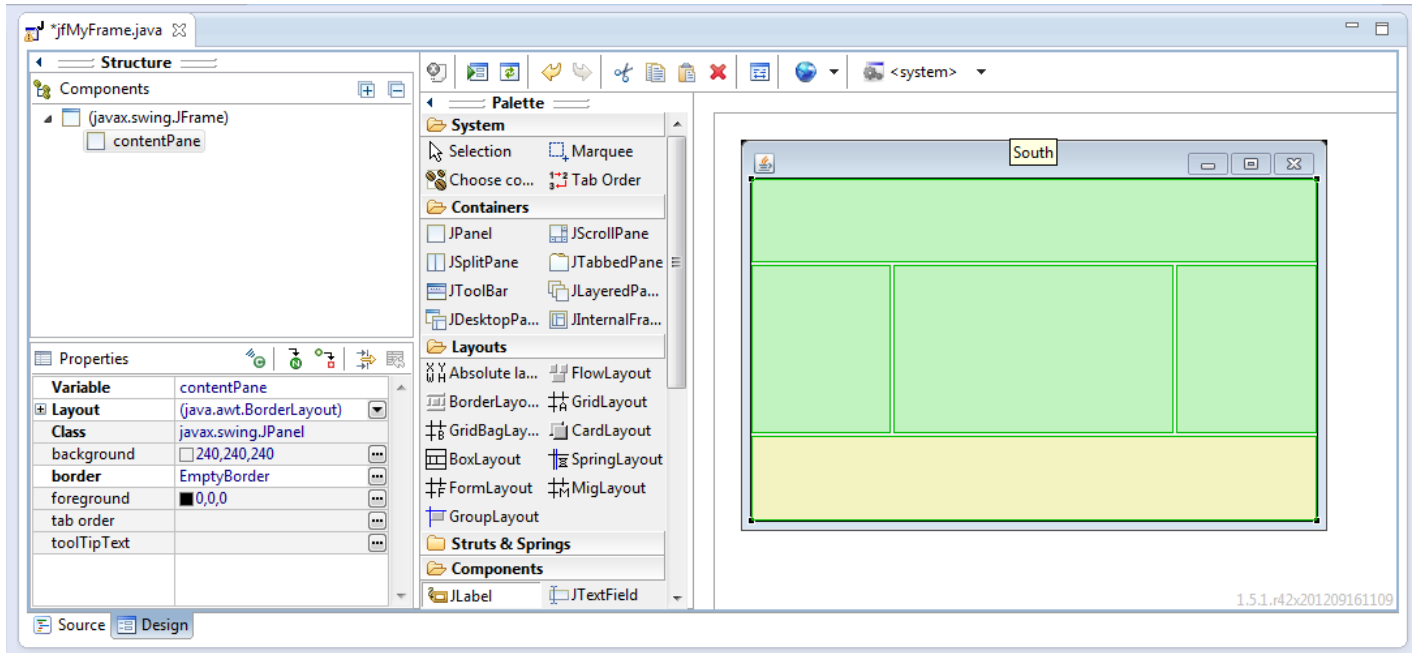


Рисунок 3.3 — Визуальное оформление компонентов класса jfMyFrame

4. ВАРИАНТЫ ЗАДАНИЙ

Таблица 4.1 Варианты заданий

№	Тип информации (см. ниже)	Поле для сортировки (P)	Направление (U)	Тип коллекции (T)
1	A	1	Убывание	HashSet
2	B	2	Возрастание	HashMap
3	C	3	Убывание	ArrayList
4	D	4	Возрастание	LinkedList
5	E	1	Убывание	HashSet
6	A	2	Возрастание	HashMap
7	B	3	Убывание	ArrayList
8	C	4	Возрастание	LinkedList
9	D	1	Убывание	HashSet
10	E	2	Возрастание	HashMap
11	A	3	Убывание	ArrayList
12	B	4	Возрастание	LinkedList
13	C	1	Убывание	HashSet
14	D	2	Возрастание	HashMap
15	E	3	Убывание	ArrayList

Тип информации:

А: Компакт диск(Название альбома, Исполнитель, Количество треков, Длительность звучания);

В: Ноутбук(Идентификатор модели, Производитель процессора, Тактовая частота процессора, Объем ОЗУ);

С: Автомобиль(Марка, Год выпуска, Объем двигателя, Максимальная скорость);

Д: Смартфон(Модель, Размер экрана, Тип экрана, Объем встроенной флэш-памяти).

Е: Книга(Автор, Год издания, Количество страниц, Издательство);

5. СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

Титульный лист, цель работы, постановку задачи, вариант задания, текст программы с комментариями, скриншоты выполнения и описание тестовых примеров, выводы по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Расскажите о назначении библиотеки Swing?
2. Что такое модель MVC?
3. Какой механизм для обработки событий реализован в библиотеке Swing?
4. Что такое менеджеры компоновки?
5. Что такое модели в Swing и как можно их применять?
6. Поясните разницу между контейнерами и компонентами?
7. Для чего применяется WindowBuilder?
8. Назовите основные компоненты Swing.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ноутон, П. Java™ 2 [Текст] : пер. с англ. / П. Ноутон, Г. Шилдт. - СПб. : БХВ – Петербург, 2007. - 1050 с.
2. Шилдт, Г. Искусство программирования на Java [Текст] : пер. с англ. / Г. Шилдт, Д. Холмс. - М. ; СПб. ; К. : Вильямс, 2005. - 334 с.
3. Хабибуллин, И. Ш. Java 2 [Текст] : самоучитель / И. Ш. Хабибуллин. - СПб. : БХВ - Петербург, 2005. - 720 с.
4. Портянкин, И. А. Swing: эффективные пользовательские интерфейсы [Текст] / И. А. Портянкин. - М. и др. : Питер, 2005. - 528 с.
5. Шилдт, Г. Swing: Руководство для начинающих [Текст] : пер. с англ. / Г. Шилдт - М. ; СПб. ; К. : Вильямс, 2007. – 967 с.

ПРИЛОЖЕНИЕ А

Пример графического Java приложения для работы с таблицами с использованием библиотеки Swing.

А.1 Разработка графического интерфейса приложения.

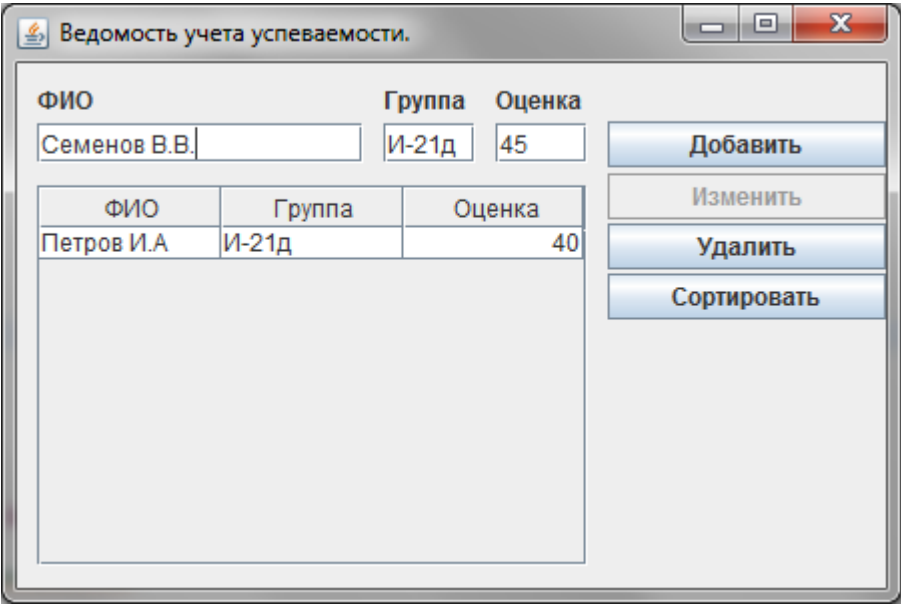


Рисунок А.1 — Графический интерфейс Java приложения.

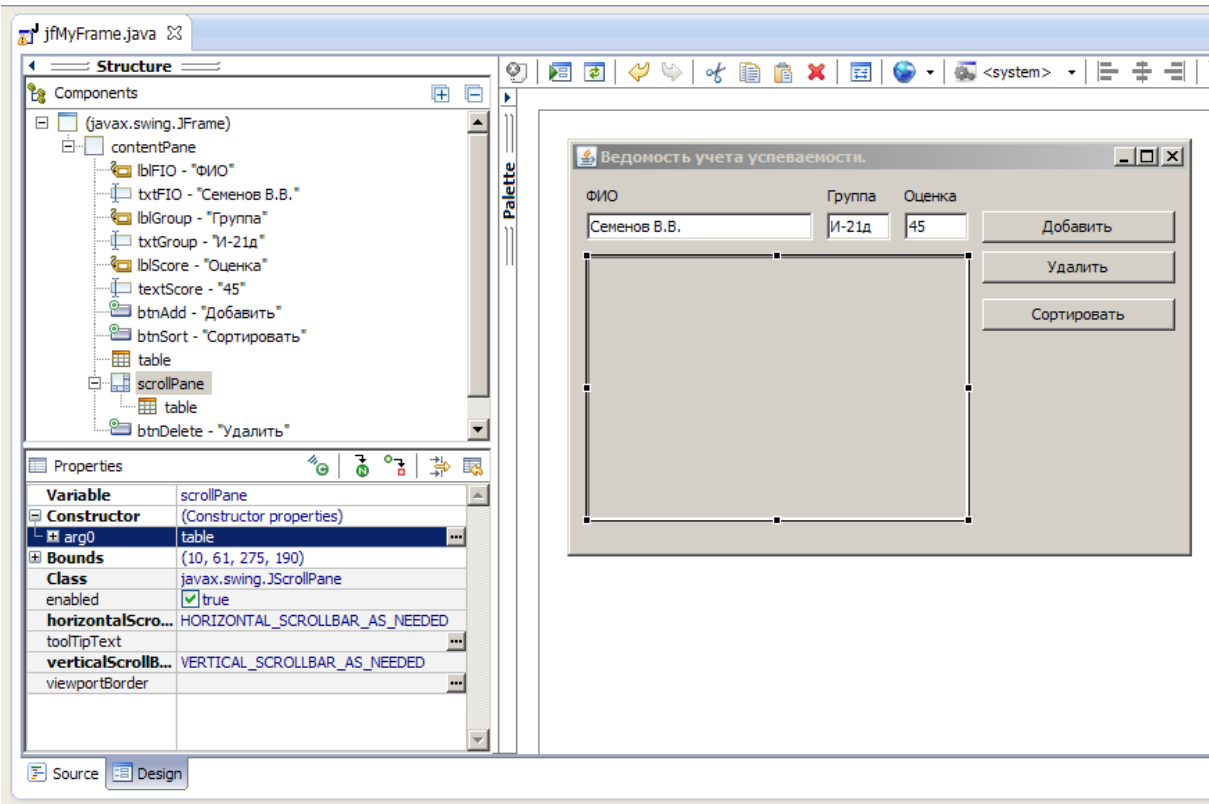


Рисунок А.2 — Разработка графического интерфейс Java приложения в WindowsBuilder Pro среды Eclipse.

A.2 Текст программы.

```

/*
 * Приложение использует компоненты Swing
 * для создания графического интерфейса,
 * демонстрирует возможности реализации
 * моделей табличных компонентов для добавления
 * удаления, редактирования и сортировки элементов.
 * Для хранения элементов применяется коллекция класса ArrayList.
 * Записи таблицы - объекты Студент с полями:
 * - ФИО
 * - Группа
 * - Оценка
 */
//Подключение всех необходимых пакетов
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JOptionPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import javax.swing.table.*;
import javax.swing.JScrollPane;

public class jfMyFrame extends JFrame implements KeyListener{

    // Объявление всех используемых компонентов.
    private JPanel contentPane;
    private JTextField txtFIO;
    private JTextField txtGroup;
    private JTextField textScore;
    private JTable table;
    private JButton btnSort;
    private JButton btnUpdate;
    private JScrollPane scrollPane;
    private JButton btnDelete;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    jfMyFrame frame = new jfMyFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}

```

```

    });
}

@Override
public void keyTyped(KeyEvent e) {
    // обработка события нажатия клавиш на полях ввода
    //если изменяют поля ввода и выделена строка таблицы - разешаем кнопку Update
    btnUpdate.setEnabled( ((e.getSource()==txtFIO || e.getSource()==txtGroup ||
e.getSource()==textScore)&&(table.getSelectedRow())>=0)) );
}

@Override
public void keyPressed(KeyEvent e) {
    // TODO Auto-generated method stub
}

@Override
public void keyReleased(KeyEvent e) {
    // TODO Auto-generated method stub
}

/**
 * Create the frame.
 */
public jfMyFrame() {
    //В конструкторе фрейма описываем параметры расположения всех объектов на
    форме,
    //их размеры, инициализационные значения и прочее
    super("Ведомость учета успеваемости.");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblFIO = new JLabel("ФИО");
    lblFIO.setBounds(10, 11, 29, 14);
    contentPane.add(lblFIO);

    txtFIO = new JTextField();
    txtFIO.setText("Семенов В.В.");
    txtFIO.setBounds(10, 30, 163, 20);
    contentPane.add(txtFIO);
    txtFIO.setColumns(10);
    txtFIO.addKeyListener(this);

    JLabel lblGroup = new JLabel("Группа");
    lblGroup.setBounds(183, 11, 46, 14);
    contentPane.add(lblGroup);

    txtGroup = new JTextField();
    txtGroup.setText("И-21д");
    txtGroup.setBounds(183, 30, 46, 20);
    contentPane.add(txtGroup);
    txtGroup.setColumns(10);
    txtGroup.addKeyListener(this);

    JLabel lblScore = new JLabel("Оценка");
    lblScore.setBounds(239, 11, 72, 14);

```

```

        contentPane.add(lblScore);

        textScore = new JTextField();
        textScore.setText("45");
        textScore.setBounds(239, 30, 46, 20);
        contentPane.add(textScore);
        textScore.setColumns(10);
        textScore.addKeyListener(this);

        переменной коллекции
        final ArrayList <CStudent> tbl=new ArrayList <CStudent> (); //создание
        tbl.add(new CStudent("Петров И.А", "И-21д", 40)); //добавление элементов в
        коллекцию

        переменной экземпляра класса модели таблицы
        final StudentsTableModel stm=new StudentsTableModel(tbl); // объявление

        JButton btnAdd = new JButton("Добавить");
        btnAdd.setBounds(295, 29, 139, 23);
        contentPane.add(btnAdd);
        btnAdd.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                stm.addRow(new
CStudent(txtFIO.getText(), txtGroup.getText(), Integer.parseInt(textScore.getText())));
            }
        });

        btnUpdate = new JButton("Изменить");
        btnUpdate.setBounds(295, 55, 139, 23);
        btnUpdate.setEnabled(false);

        contentPane.add(btnUpdate);
        btnUpdate.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (table.getSelectedRow() >= 0)
stm.updateRow(table.getSelectedRow(), new
CStudent(txtFIO.getText(), txtGroup.getText(), Integer.parseInt(textScore.getText())));
                else JOptionPane.showMessageDialog(jfMyFrame.this, "Не выбрана ни
одна строка таблицы");
            }
        });

        btnDelete = new JButton("Удалить");
        btnDelete.setBounds(295, 80, 139, 23);
        contentPane.add(btnDelete);
        btnDelete.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                stm.deleteRow(txtFIO.getText());
            }
        });

        btnSort = new JButton("Сортировать");
        btnSort.setBounds(295, 105, 139, 23);
        contentPane.add(btnSort);
        btnSort.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                stm.Sort();
            }
        });

        table = new JTable(stm);

```

```

table.setBounds(10, 61, 301, 63);
table.addMouseListener(new MouseListener(){
    //загрузка записи в поля редактирования при щелчке на строке таблицы
    public void mouseClicked(MouseEvent e){
        int row=table.getSelectedRow();
        txtFIO.setText(table.getValueAt(row, 0).toString());
        txtGroup.setText(table.getValueAt(row, 1).toString());
        textScore.setText(table.getValueAt(row, 2).toString());
        //сделать кнопку обновления недоступной(пока нет изменений)
        btnUpdate.setEnabled(false);
    }

    @Override
    public void mousePressed(MouseEvent e) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseEntered(MouseEvent e) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseExited(MouseEvent e) {
        // TODO Auto-generated method stub
    }
});
contentPane.add(table);

scrollPane = new JScrollPane(table);
scrollPane.setBounds(10, 61, 275, 190);
contentPane.add(scrollPane);

}
/*Класс описывающий один объект таблицы*/
public class CStudent implements Comparable <CStudent> {

    private String FIO;
    private String Group;
    private int Score;

    public CStudent(String fio, String group, int score) {
        this.setFIO(fio);
        this.setGroup(group);
        this.setScore(score);
    }

    public void setFIO(String fio) {
        this.FIO = fio;
    }

    public String getFIO() {
        return FIO;
    }

    public void setGroup(String group) {

```

24

```

        this.Group = group;
    }

    public String getGroup() {
        return Group;
    }
    public void setScore(int score) {
        this.Score = score;
    }

    public int getScore() {
        return Score;
    }

    @Override
    public int compareTo(CStudent cs) { //переопределенный метод интерфейса
        // необходим для выполнения сортировки
        return this.FIO.compareTo(cs.FIO);
    }
}

/*Модель описывающая все манипуляции с данными */
public class StudentsTableModel extends AbstractTableModel {
    //идентификатор для сериализации
    private static final long serialVersionUID = 7927259757559420606L;
    private ArrayList<CStudent> is_students; //хранилище данных

    public StudentsTableModel(ArrayList<CStudent> is_students) { //конструктор

        this.is_students = is_students;
    }
    //обязательный метод - возвращает тип значения столбца
    public Class<?> getColumnClass(int columnIndex) {
        return getValueAt(0, columnIndex).getClass();
    }

    public int getColumnCount() {
        return 3;
    }
    //обязательный метод - возвращает имя столбца
    public String getColumnName(int columnIndex) {
        switch (columnIndex) {
            case 0:
                return "ФИО";
            case 1:
                return "Группа";
            case 2:
                return "Оценка";
        }
        return "";
    }

    public int getRowCount() { //обязательный метод - возвращает количество строк
        return is_students.size();
    }
    //обязательный метод - возвращает значение поля таблицы
    public Object getValueAt(int rowIndex, int columnIndex) {
        CStudent student = is_students.get(rowIndex);
        switch (columnIndex) {

```

Comparable

модели


```

        case 0:
            return student.getFIO();
        case 1:
            return student.getGroup();
        case 2:
            return student.getScore();
    }
    return "";
}

// определяет возможно ли редактирование таблицы
public boolean isCellEditable(int rowIndex, int columnIndex) {
    return false;
}

//выполняет изменение данных в ячейке
public void setValueAt(Object value, int rowIndex, int columnIndex) {

}

public void addRow(CStudent nr) { //добавление строки в таблицу
    is_students.add(nr);
    fireTableDataChanged();//вызываем для обновления таблицы
    //выделяем добавленную строку - иначе после обновления таблицы теряет
    выделение

    int index = is_students.indexOf(nr);
    table.setRowSelectionInterval(index, index);
}

public void deleteRow(String fio) { //удаление строки из таблицы
    Iterator <CStudent> istud=is_students.iterator();
    boolean flag=false;
    while(istud.hasNext()) {
        if(istud.next().getFIO().equals(fio)) {
            istud.remove();
            flag=true;
        }
    }

    if(flag){
        fireTableDataChanged();//вызываем для обновления таблицы
    }else
        JOptionPane.showMessageDialog(jfMyFrame.this,"Запись '"+fio+"' не
        найдена!");
}

public void updateRow(int index,CStudent nr) { //обновление строки в таблице

    is_students.set(index, nr);
    fireTableDataChanged();//вызываем для обновления таблицы
    //выделяем измененную строку - иначе после обновления таблицы теряет
    выделение

    table.setRowSelectionInterval(index, index);
}

public void Sort(){ //сортировка
    Collections.sort(is_students);
    fireTableDataChanged(); //вызываем для обновления таблицы
}

}
}

```


Заказ № _____ от « _____ » _____ 2015г. Тираж _____ экз.
Изд-во СевГУ