Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический университет

“ЛЭТИ” им.В.И.Ульянова (Ленина)»

Кафедра МОЭВМ

**ОТЧЕТ**

**по лабораторно-практической работе № 5**

**«Сохранение и загрузка данных из файла»**

**по дисциплине: «Объектно - ориентированное программирование на языке Java»**

Выполнил: Локтионов Т. И.

Факультет КТИ

Группа № 3311

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2024 г

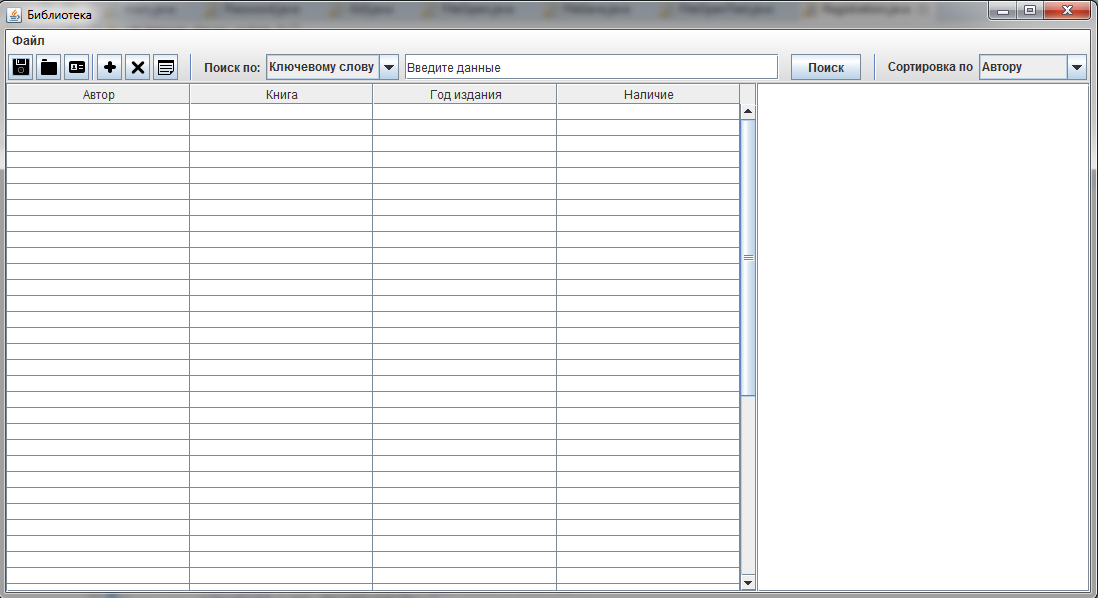
**Цель работы:**знакомство с организацией обмена данными между объектами экранной формы и файлом.

**Описание задания:**1. Создать обработчики (Listeners) для сохранения и выгрузки данных в/из файла.   
2. Определить методы для различных объектов экранной формы.

**Описание экранной формы:**

Экранная форма предназначена для отображения списка больных и врачей для администратора регистратуры поликлиники, она может менять свой размер на экране (начальный размер 800х600). Форма должна реализовывать следующие функции: загрузку списка пациентов, болезней, врачей, дат приема и состояния приема из файла и выгрузку этой информации в файл; редактирование списка, включая: добавление, удаление, корректировку информации; удобный поиск, по ключевым словам, и/или другими методами (имя пациента, дата и т.д.)

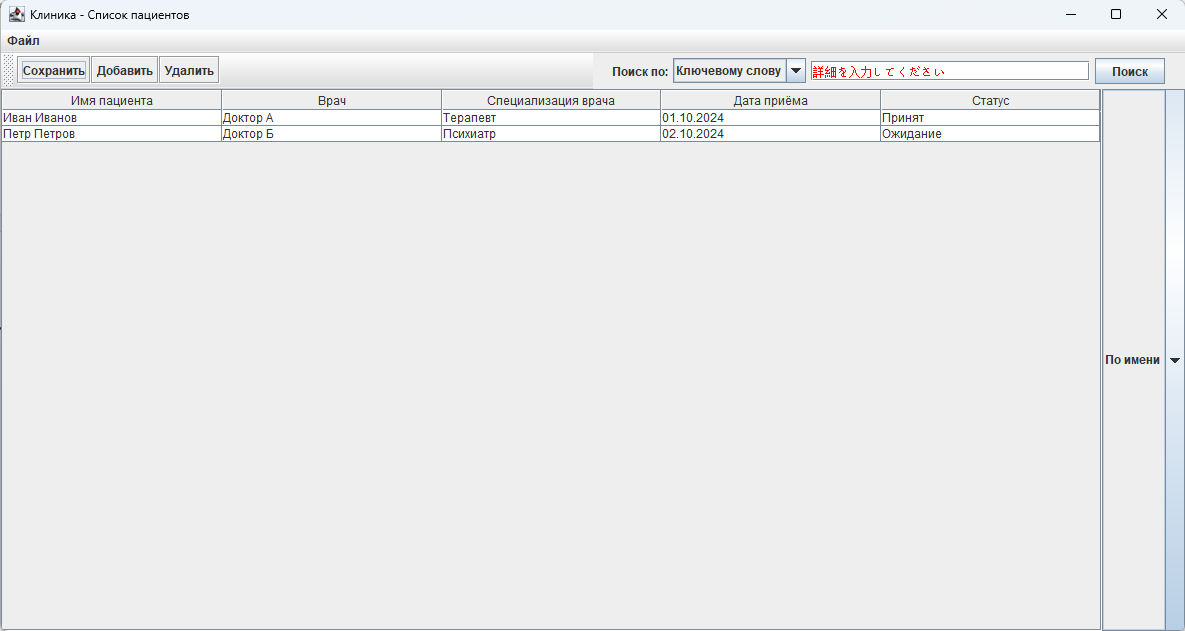
**Макет** (взят из приложенных к методичке файлов):



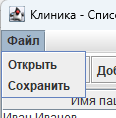
**Перечень возможных исключений:**

* Пустые поля ввода, которые недопустимы при добавлении пациента в таблицу.
* Некорректные данные, которые не соответствуют установленным требованиям (например, неверный формат).
* Дублирование записей, когда пользователь пытается добавить существующего пациента.
* Неожиданные ошибки, возникающие в ходе выполнения программы, которые могут быть перехвачены и обработаны.

**Работоспособность приложения:**

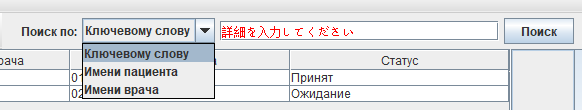
JFrame: ****

JMenuBar:

****

JToolBar:



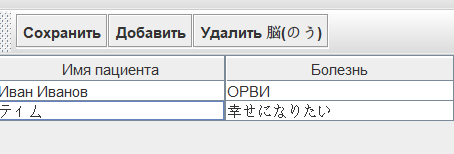
JPanel && JTextField:  
 

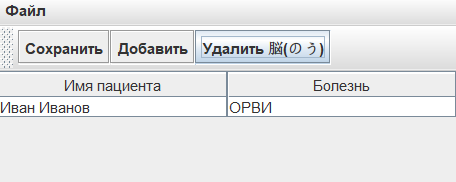
JComboBox:



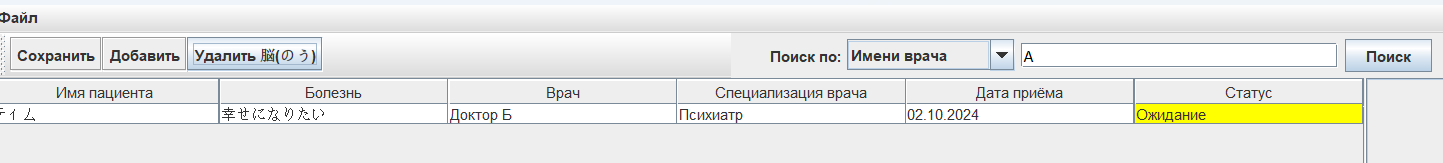
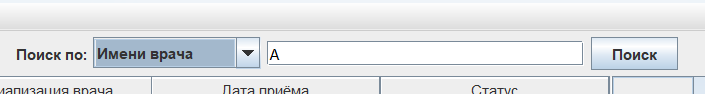
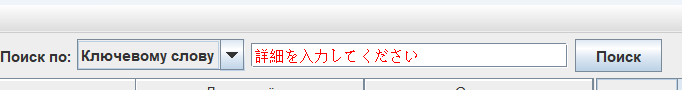
**Работоспособность слушателей:**

deleteButton:



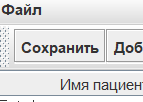


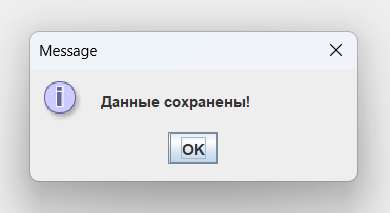
searchField && searchButton:



// работает как поиск по ключевому слову, так и изменение поля поиска при начале ввода

saveButton:

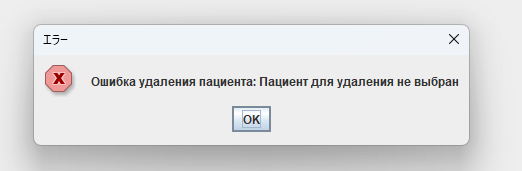




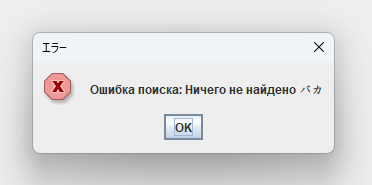
// только вывод сообщения

**Работоспособность исключений:**

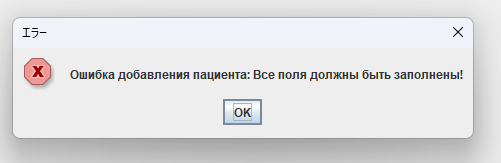
**//** deleteButton



// searchButton



// addButton

**Работоспособность загрузки/выгрузки данных из файла:**

**//** представлен порядок действий

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Ссылки:**

>>репозиторий: https://github.com/iconLti/LTprojects/tree/master/OOP/Java%20projects/Hospital-lab05

>> видео отчет:  
Google Disk: https://drive.google.com/drive/folders/1YRKOxpkXtTLnzTo8e\_P50Sp0jCBNFT9a?usp=sharing

**Текст программы:**

// ClinicApp.java

*import* javax.swing.SwingUtilities;  
  
*/\*\*  
 \* Основной класс приложения, содержащий точку входа.  
 \** ***@author*** *Tim Loktionov 3311  
 \** ***@version*** *1.00  
 \*/  
public class* ClinicApp {  
 */\*\*  
 \* Главный метод запуска программы.  
 \* Вызывает метод создания и отображения интерфейса.  
 \*  
 \** ***@param args*** *аргументы командной строки (не используются).  
 \*/  
 public static void* main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> *new* GUI().buildAndShowGUI());  
 }  
}

// GUI.java

*import* javax.swing.\*;  
*import* java.awt.\*;  
*import* java.io.\*;  
*import* javax.swing.table.DefaultTableModel;  
*import* javax.swing.table.*TableCellRenderer*;  
  
*/\*\*  
 \* Основной класс, отвечающий за построение интерфейса приложения "Клиника".  
 \* Приложение предназначено для управления списком пациентов.  
 \* Включает добавление, удаление пациентов, сохранение данных и поиск по имени.  
 \*/  
public class* GUI {  
 *// Объявление компонентов  
 private* JFrame frame;  
 *private* JMenuBar menuBar;  
 *private* JMenu fileMenu;  
 *private* JMenuItem openItem, saveItem;  
 *private* JToolBar toolBar;  
 *private* JButton saveButton, addButton, deleteButton;  
 *private* JButton searchButton;  
 *private* JComboBox<String> searchType;  
 *private* JComboBox<String> sortType;  
 *private* JTextField searchField;  
 *private* JTable dataTable;  
 *private* JScrollPane tableScrollPane;  
 *private* DefaultTableModel tableModel;  
  
 */\*\*  
 \* Метод для построения и отображения графического интерфейса.  
 \* Создает основное окно приложения, меню, панель инструментов,  
 \* элементы для поиска и таблицу данных.  
 \*/  
 public void* buildAndShowGUI() {  
 frame = *new* JFrame("Клиника - Список пациентов");  
 frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 frame.setSize(1200, 640);  
  
 *// Создание меню* menuBar = *new* JMenuBar();  
 fileMenu = *new* JMenu("Файл");  
 openItem = *new* JMenuItem("Открыть");  
 saveItem = *new* JMenuItem("Сохранить");  
 fileMenu.add(openItem);  
 fileMenu.add(saveItem);  
 menuBar.add(fileMenu);  
 frame.setJMenuBar(menuBar);  
  
 *// Панель инструментов* toolBar = *new* JToolBar();  
 saveButton = *new* JButton("Сохранить");  
 addButton = *new* JButton("Добавить");  
 deleteButton = *new* JButton("Удалить 脳(のう)");  
 toolBar.add(saveButton);  
 toolBar.add(addButton);  
 toolBar.add(deleteButton);  
  
  
 *// Панель поиска* JPanel searchPanel = *new* JPanel();  
 searchType = *new* JComboBox<>(*new* String[]{"Ключевому слову", "Имени пациента", "Имени врача"});  
 searchField = *new* JTextField(25);  
 searchButton = *new* JButton("Поиск");  
  
  
 *// Текст подсказка* String placeholder = "詳細を入力してください";  
 searchField.setText(placeholder);  
 searchField.setForeground(Color.***RED***); *// цвет текста* searchField.addFocusListener(Listeners.*getSearchFieldFocusListener*(searchField, placeholder));  
  
 searchPanel.add(*new* JLabel("Поиск по:"));  
 searchPanel.add(searchType);  
 searchPanel.add(searchField);  
 searchPanel.add(searchButton);  
  
  
 *// Контейнер для обеих частей* JPanel topPanel = *new* JPanel(*new* GridLayout(1, 2)); *// Одна строка, два столбца* topPanel.add(toolBar);  
 topPanel.add(searchPanel);  
 frame.add(topPanel, BorderLayout.***NORTH***);  
  
  
 *// Таблица с данными* String[] columns = {"Имя пациента", "Болезнь", "Врач", "Специализация врача", "Дата приёма", "Статус"};  
 tableModel = *new* DefaultTableModel(*new* Object[][]{}, columns);  
  
 dataTable = *new* JTable(tableModel) {  
 @Override  
 *public* Component prepareRenderer(*TableCellRenderer* renderer, *int* row, *int* column) {  
 Component cell = *super*.prepareRenderer(renderer, row, column);  
 *// Проверяем, что это колонка "Статус" (индекс 5)  
 if* (column == 5) {  
 String status = (String) getValueAt(row, column);  
 *switch* (status) {  
 *case* "Принят":  
 cell.setBackground(Color.***GREEN***);  
 *break*;  
 *case* "Ожидание":  
 cell.setBackground(Color.***YELLOW***);  
 *break*;  
 *case* "Отменен":  
 *case* "Отменён":  
 cell.setBackground(Color.***RED***);  
 *break*;  
 *default*:  
 cell.setBackground(Color.***WHITE***); *// Фон для остальных статусов  
 break*;  
 }  
 } *else* {  
 cell.setBackground(Color.***WHITE***); *// Для остальных колонок устанавливаем белый фон* }  
 *return* cell;  
 }  
 };  
 tableScrollPane = *new* JScrollPane(dataTable);  
 frame.add(tableScrollPane, BorderLayout.***CENTER***);  
  
 *// Сортировка* sortType = *new* JComboBox<>(*new* String[]{"По имени", "По дате"});  
 frame.add(sortType, BorderLayout.***EAST***);  
  
  
 *// Слушатели (Action)  
 // поиск* searchButton.addActionListener(Listeners.*getSearchListener*(dataTable, searchField, searchType, frame));  
 *//кнопки* saveButton.addActionListener(Listeners.*getSaveDataListener*(frame, tableModel));  
 addButton.addActionListener(Listeners.*getAddPatientListener*(tableModel));  
 deleteButton.addActionListener(Listeners.*getDeletePatientListener*(tableModel, dataTable, frame));  
 *//сортировка* sortType.addActionListener(Listeners.*getSortTypeActionListener*(sortType, frame));  
 *// Слушатели для меню* openItem.addActionListener(Listeners.*getLoadDataListener*(tableModel, frame));  
 saveItem.addActionListener(Listeners.*getSaveToPathDataListener*(frame, tableModel));  
  
 *// Визуализация* frame.setVisible(*true*);  
 }  
}

// Listeners.java

*import* javax.swing.\*;  
*import* javax.swing.table.DefaultTableModel;  
*import* java.awt.\*;  
*import* java.awt.event.\*;  
*import* java.io.\*;  
  
*/\*\*  
 \* Класс, содержащий слушатели для различных действий в приложении.  
 \*/  
public class* Listeners {  
  
 */\*\*  
 \* Создает слушатель для добавления нового пациента.  
 \*  
 \** ***@param*** *tableModel модель таблицы, в которую будет добавлен пациент  
 \** ***@return*** *ActionListener для добавления нового пациента  
 \*/  
 public static ActionListener* getAddPatientListener(DefaultTableModel tableModel) {  
 *return* e -> {  
 *try* {  
 String name = JOptionPane.*showInputDialog*("Введите имя пациента:");  
 String disease = JOptionPane.*showInputDialog*("Введите название болезни:");  
 String doctor = JOptionPane.*showInputDialog*("Введите имя врача:");  
 String specialization = JOptionPane.*showInputDialog*("Введите специализацию врача:");  
 String date = JOptionPane.*showInputDialog*("Введите дату приёма:");  
 String status = JOptionPane.*showInputDialog*("Введите статус:");  
  
 *// Проверка, что поля не пустые и не равны null, так как сама программа добавляет пустые строки  
 if* (name != *null* && !name.trim().isEmpty() &&  
 disease != *null* && !disease.trim().isEmpty() &&  
 doctor != *null* && !doctor.trim().isEmpty() &&  
 specialization != *null* && !specialization.trim().isEmpty() &&  
 date != *null* && !date.trim().isEmpty() &&  
 status != *null* && !status.trim().isEmpty()) {  
  
 tableModel.addRow(*new* Object[]{name, disease, doctor, specialization, date, status});  
 } *else* {  
 *throw new* IllegalArgumentException("Все поля должны быть заполнены!");  
 }  
 } *catch* (Exception ex) {  
 JOptionPane.*showMessageDialog*(*null*, "Ошибка добавления пациента: "  
 + ex.getMessage(), " エラー", JOptionPane.***ERROR\_MESSAGE***);  
 }  
 };  
 }  
  
  
  
 */\*\*  
 \* Создает слушатель для удаления пациента.  
 \*  
 \** ***@param*** *tableModel модель таблицы, из которой будет удален пациент  
 \** ***@param*** *dataTable таблица, отображающая пациентов  
 \** ***@param*** *frame окно, в котором отображаются сообщения  
 \** ***@return*** *ActionListener для удаления пациента  
 \*/  
 public static ActionListener* getDeletePatientListener(DefaultTableModel tableModel, JTable dataTable, JFrame frame) {  
 *return* e -> {  
 *try* {  
 *int* selectedRow = dataTable.getSelectedRow();  
 *if* (selectedRow != -1) {  
 tableModel.removeRow(selectedRow);  
 } *else* {  
 *throw new* IllegalArgumentException("Пациент для удаления не выбран");  
 }  
 } *catch* (Exception ex) {  
 JOptionPane.*showMessageDialog*(frame, "Ошибка удаления пациента: " + ex.getMessage(),  
 " エラー", JOptionPane.***ERROR\_MESSAGE***);  
 }  
 };  
 }  
  
 */\*\*  
 \* Создает слушатель для сохранения данных.  
 \*  
 \** ***@param*** *frame окно, в котором отображаются сообщения  
 \** ***@return*** *ActionListener для сохранения данных  
 \*/  
 public static ActionListener* getSaveDataListener(JFrame frame) {  
 *return* e -> JOptionPane.*showMessageDialog*(frame, "Данные сохранены!");  
 }  
  
 */\*\*  
 \* Создает слушатель для поиска пациента по имени или врачу.  
 \*  
 \** ***@param*** *dataTable таблица, в которой производится поиск  
 \** ***@param*** *searchField поле ввода для текста поиска  
 \** ***@param*** *searchType комбобокс для выбора типа поиска  
 \** ***@param*** *frame окно, в котором отображаются сообщения  
 \** ***@return*** *ActionListener для поиска пациента  
 \*/  
 public static ActionListener* getSearchListener(JTable dataTable, JTextField searchField,  
 JComboBox<String> searchType, JFrame frame) {  
 *return* e -> {  
 *try* {  
 String searchText = searchField.getText().toLowerCase();  
 *int* searchColumn = searchType.getSelectedIndex() == 1 ? 0 : 2; *// 0 - имя пациента, 1 - имя врача  
  
 boolean* found = *false*;  
 *for* (*int* i = 0; i < dataTable.getRowCount(); i++) {  
 String value = dataTable.getValueAt(i, searchColumn).toString().toLowerCase();  
 *if* (value.contains(searchText)) {  
 dataTable.setRowSelectionInterval(i, i);  
 found = *true*;  
 *break*;  
 }  
 }  
 *if* (!found) {  
 *throw new* IllegalArgumentException("Ничего не найдено バカ");  
 }  
 } *catch* (Exception ex) {  
 JOptionPane.*showMessageDialog*(frame, "Ошибка поиска: " + ex.getMessage(),  
 " エラー", JOptionPane.***ERROR\_MESSAGE***);  
 }  
  
 };  
 }  
  
 */\*\*  
 \* Создает слушатель для сортировки пациентов.  
 \*  
 \** ***@param*** *sortType комбобокс для выбора типа сортировки  
 \** ***@param*** *frame окно, в котором отображаются сообщения  
 \** ***@return*** *ActionListener для сортировки пациентов  
 \*/  
 public static ActionListener* getSortTypeActionListener(JComboBox<String> sortType, JFrame frame) {  
 *return* e -> {  
 String selectedSort = (String) sortType.getSelectedItem();  
 *if* ("По имени".equals(selectedSort)) {  
 JOptionPane.*showMessageDialog*(frame, "Сортировка по имени");  
 } *else if* ("По дате".equals(selectedSort)) {  
 JOptionPane.*showMessageDialog*(frame, "Сортировка по дате");  
 }  
 };  
 }  
  
 */\*\*  
 \* Создает слушатель для управления поведением поля поиска.  
 \*  
 \** ***@param*** *searchField поле ввода для текста поиска  
 \** ***@param*** *placeholder текст-заполнитель для поля поиска  
 \** ***@return*** *FocusAdapter для управления поведением поля поиска  
 \*/  
 public static* FocusAdapter getSearchFieldFocusListener(JTextField searchField, String placeholder) {  
 *return new* FocusAdapter() {  
 @Override  
 *public void* focusGained(FocusEvent e) {  
 *if* (searchField.getText().equals(placeholder)) {  
 searchField.setText("");  
 searchField.setForeground(Color.***BLACK***);  
 }  
 }  
  
 @Override  
 *public void* focusLost(FocusEvent e) {  
 *if* (searchField.getText().isEmpty()) {  
 searchField.setForeground(Color.***RED***);  
 searchField.setText(placeholder);  
 }  
 }  
 };  
 }  
  
  
 *// Work with file //  
 private static* File *loadedFile*; *// Храним ссылку на загруженный файл  
  
 public static void* setLoadedFile(File file) {  
 *loadedFile* = file; *// Метод для установки файла при загрузке* }  
  
 */\*\*  
 \* Создает слушатель для выгрузки данных из файла  
 \*  
 \** ***@param*** *tableModel ячейки таблицы  
 \** ***@param frame*** *окно, в котором отображаются сообщения  
 \** ***@return*** *ActionListener для выгрузки данных  
 \*/  
 public static ActionListener* getLoadDataListener(DefaultTableModel tableModel, JFrame frame) {  
 *return* e -> {  
 JFileChooser fileChooser = *new* JFileChooser();  
 *int* result = fileChooser.showOpenDialog(frame);  
  
 *if* (result == JFileChooser.***APPROVE\_OPTION***) {  
 File file = fileChooser.getSelectedFile();  
 *try* (BufferedReader br = *new* BufferedReader(*new* FileReader(file))) {  
 String line;  
 tableModel.setRowCount(0);  
 *while* ((line = br.readLine()) != *null*) {  
 String[] data = line.split(";");  
 tableModel.addRow(data);  
 }  
 *// Сохраняем файл для дальнейшего использования при сохранении данных* Listeners.*setLoadedFile*(file);  
 JOptionPane.*showMessageDialog*(frame, "Данные успешно загружены");  
 } *catch* (IOException ex) {  
 JOptionPane.*showMessageDialog*(frame, "Ошибка загрузки в файл: " + ex.getMessage(),  
 " エラー", JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
 };  
 }  
  
 */\*\*  
 \* Создает слушатель для сохранения данных в тот же файл  
 \*  
 \** ***@param tableModel*** *ячейки таблицы  
 \** ***@param frame*** *окно, в котором отображаются сообщения  
 \** ***@return*** *ActionListener для сохранения данных  
 \*/  
 public static ActionListener* getSaveDataListener(JFrame frame, DefaultTableModel tableModel) {  
 *return* e -> {  
 *if* (*loadedFile* != *null*) {  
 *try* (BufferedWriter bw = *new* BufferedWriter(*new* FileWriter(*loadedFile*))) {  
 *for* (*int* row = 0; row < tableModel.getRowCount(); row++) {  
 *for* (*int* col = 0; col < tableModel.getColumnCount(); col++) {  
 bw.write(String.*valueOf*(tableModel.getValueAt(row, col)));  
 *if* (col < tableModel.getColumnCount() - 1) {  
 bw.write(";");  
 }  
 }  
 bw.newLine(); *// Переход на новую строку* }  
 JOptionPane.*showMessageDialog*(frame, "Данные успешно сохранены!");  
 } *catch* (IOException ex) {  
 JOptionPane.*showMessageDialog*(frame, "Ошибка сохранения файла: " + ex.getMessage(),  
 "Ошибка", JOptionPane.***ERROR\_MESSAGE***);  
 }  
 } *else* {  
 JOptionPane.*showMessageDialog*(frame, "Файл для сохранения не загружен!",  
 "Ошибка", JOptionPane.***ERROR\_MESSAGE***);  
 }  
 };  
 }  
  
 */\*\*  
 \* Создает слушатель для сохранения данных в файл в формате "сохранить как"  
 \*  
 \** ***@param tableModel*** *ячейки таблицы  
 \** ***@param frame*** *окно, в котором отображаются сообщения  
 \** ***@return*** *ActionListener для сохранения данных  
 \*/  
 public static ActionListener* getSaveToPathDataListener(JFrame frame, DefaultTableModel tableModel) {  
 *return* e -> {  
 JFileChooser fileChooser = *new* JFileChooser();  
 *int* result = fileChooser.showSaveDialog(frame);  
  
 *if* (result == JFileChooser.***APPROVE\_OPTION***) {  
 File file = fileChooser.getSelectedFile();  
 *try* (PrintWriter pw = *new* PrintWriter(*new* FileWriter(file))) {  
 *for* (*int* row = 0; row < tableModel.getRowCount(); row++) {  
 *for* (*int* col = 0; col < tableModel.getColumnCount(); col++) {  
 pw.print(tableModel.getValueAt(row, col));  
 *if* (col < tableModel.getColumnCount() - 1) {  
 pw.print(";");  
 }  
 }  
 pw.println();  
 }  
 JOptionPane.*showMessageDialog*(frame, "Данные успешно сохранены!");  
 } *catch* (IOException ex) {  
 JOptionPane.*showMessageDialog*(frame, "Ошибка сохранения файла: " + ex.getMessage(), "Ошибка", JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
 };  
 }  
}