**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №3**

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

Тема: «**Обработка событий**»

|  |  |  |
| --- | --- | --- |
| Студент гр. 4311 |  | Левицкий М. Ю. |
| Преподаватель |  | Лазарев Э.А. |

Санкт-Петербург

2025

# ЦЕЛЬ

Знакомство со способами подключения слушателей событий к графическим компонентам пользовательского интерфейса.

* 1. Создайте новый проект, который будет дублировать проект лабораторной работы № 2.
* 2. Выявите на экранной форме, разработанной в лабораторной работе № 2, события, в ответ на которые потребуется реакция приложения.
* 3. К двум-трем разнотипным компонентам графического интерфейса пользователя напишите код слушателей. Слушатели должны реализовать полностью или частично свою функциональность, вывести на экран результат своей работы или информационное сообщение. В код слушателей должны быть вставлены комментарии документации, отражающие их работу.
* 4. Запустите приложение и снимите с экрана скриншоты, иллюстрирующие работу слушателей.
* 5. Сгенерируйте документацию с помощью Javadoc и просмотрите ее в браузере.

**ОПИСАНИЕ ДЕЙСТВИЙ , КОТОРЫЕ ДОЛЖЕН РЕАЛИЗОВАТЬ СЛУШАТЕЛЬ**

1.Слушатель кнопки "Добавить" (ActionListener)  
2.Слушатель кнопки "Удалить" (ActionListener)  
3.Слушатель кнопки "Поиск" (ActionListener)  
4.Слушатель комбобокса фамилий (ActionListener)  
5.Слушатель выбора в таблице (ListSelectionListener)

**ИСХОДНЫЕ ТЕКСТЫ СЛУШАТЕЛЕЙ**

|  |
| --- |
| /\*\*  Слушатель для кнопки "Добавить".  Реагирует на событие клика мыши, извлекает данные из полей ввода  и добавляет нового клиента в таблицу.  \*/  addButton.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String name = clientNameField.getText().trim();  String phone = phoneField.getText().trim();  String address = addressField.getText().trim();  String newspaper = newspaperField.getText().trim();  // Проверяем, что все поля заполнены  if (name.isEmpty() || phone.isEmpty() || address.isEmpty() || newspaper.isEmpty()) {  JOptionPane.showMessageDialog(parent,  "Заполните все поля данных клиента!",  "Ошибка ввода",  JOptionPane.WARNING\_MESSAGE);  return;  }  // Добавляем новую строку в таблицу  model.addRow(new Object[]{name, phone, address, newspaper});  // Обновляем комбобокс с фамилиями  updateSurnameComboBox(model, surnameComboBox);  // Очищаем поля ввода  clearInputFields();  JOptionPane.showMessageDialog(parent,  "Клиент '" + name + "' успешно добавлен!",  "Добавление клиента",  JOptionPane.INFORMATION\_MESSAGE);  }  });  /\*\*  Слушатель для кнопки "Удалить".  Реагирует на событие клика мыши, проверяет выделенную строку в таблице  и удаляет выбранного клиента после подтверждения.  \*/  deleteButton.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  int selectedRow = clientsTable.getSelectedRow();  if (selectedRow == -1) {  JOptionPane.showMessageDialog(parent,  "Выберите клиента для удаления из таблицы!",  "Ошибка удаления",  JOptionPane.WARNING\_MESSAGE);  return;  }  String clientName = (String) model.getValueAt(selectedRow, 0);  int confirm = JOptionPane.showConfirmDialog(parent,  "Вы уверены, что хотите удалить клиента '" + clientName + "'?",  "Подтверждение удаления",  JOptionPane.YES\_NO\_OPTION);  if (confirm == JOptionPane.YES\_OPTION) {  model.removeRow(selectedRow);  updateSurnameComboBox(model, surnameComboBox);  JOptionPane.showMessageDialog(parent,  "Клиент '" + clientName + "' удален!",  "Удаление завершено",  JOptionPane.INFORMATION\_MESSAGE);  }  }  });  /\*\*  Слушатель для кнопки "Поиск".  Реагирует на событие клика мыши, выполняет поиск клиента по введенному тексту  и выделяет найденные совпадения в таблице.  \*/  searchButton.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String searchText = searchField.getText().trim().toLowerCase();  if (searchText.isEmpty()) {  JOptionPane.showMessageDialog(parent,  "Введите текст для поиска!",  "Ошибка поиска",  JOptionPane.WARNING\_MESSAGE);  return;  }  boolean found = false;  // Снимаем выделение со всех строк  clientsTable.clearSelection();  // Поиск по всем строкам и столбцам таблицы  for (int row = 0; row < model.getRowCount(); row++) {  for (int col = 0; col < model.getColumnCount(); col++) {  String cellValue = model.getValueAt(row, col).toString().toLowerCase();  if (cellValue.contains(searchText)) {  // Выделяем найденную строку  clientsTable.addRowSelectionInterval(row, row);  // Прокручиваем таблицу к найденной строке  clientsTable.scrollRectToVisible(clientsTable.getCellRect(row, 0, true));  found = true;  break; // Переходим к следующей строке  }  }  }  if (!found) {  JOptionPane.showMessageDialog(parent,  "Клиенты по запросу '" + searchText + "' не найдены!",  "Результат поиска",  JOptionPane.INFORMATION\_MESSAGE);  } else {  JOptionPane.showMessageDialog(parent,  "Найдены клиенты по запросу: '" + searchText + "'",  "Поиск завершен",  JOptionPane.INFORMATION\_MESSAGE);  }  }  });  /\*\*  Слушатель для комбобокса выбора фамилии.  Реагирует на событие выбора элемента, автоматически заполняет поля ввода  данными выбранного клиента и выделяет соответствующую строку в таблице.  \*/  comboBox.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String selectedSurname = (String) comboBox.getSelectedItem();  if (selectedSurname != null) {  // Ищем выбранную фамилию в таблице  for (int i = 0; i < model.getRowCount(); i++) {  if (selectedSurname.equals(model.getValueAt(i, 0))) {  // Заполняем поля ввода данными найденного клиента  clientNameField.setText(selectedSurname);  phoneField.setText(model.getValueAt(i, 1).toString());  addressField.setText(model.getValueAt(i, 2).toString());  newspaperField.setText(model.getValueAt(i, 3).toString());  // Выделяем строку в таблице  clientsTable.setRowSelectionInterval(i, i);  clientsTable.scrollRectToVisible(clientsTable.getCellRect(i, 0, true));  JOptionPane.showMessageDialog(parent,  "Загружены данные клиента: " + selectedSurname,  "Данные загружены",  JOptionPane.INFORMATION\_MESSAGE);  break;  }  }  }  }  });  /\*\*  Слушатель для таблицы клиентов.  Реагирует на событие выбора строки мыши, автоматически обновляет комбобокс  и выводит информацию о выбранном клиенте.  \*/  table.getSelectionModel().addListSelectionListener(e -> {  if (!e.getValueIsAdjusting()) {  int selectedRow = table.getSelectedRow();  if (selectedRow != -1) {  String selectedName = (String) table.getValueAt(selectedRow, 0);  // Обновляем выбор в комбобоксе  surnameComboBox.setSelectedItem(selectedName);  // Можно выводить дополнительную информацию в консоль для отладки  System.out.println("Выбран клиент: " + selectedName);  }  }  }); |

# КОНТРОЛЬНЫЙ ПРИМЕР

Описание проверки работоспособности приложения:

Проверяем работу удаления элемента и выпадающего списка фамилий клентов

|  |
| --- |
|  |

**Скриншот 1 для контрольного примера**

|  |
| --- |
|  |

**Скриншот 2 для контрольного примера**

|  |
| --- |
|  |

**Скриншот 3 для контрольного примера**

|  |
| --- |
|  |

**Скриншот 4 для контрольного примера**

|  |
| --- |
|  |

**Скриншот 5 для контрольного примера**

# ТЕКСТ ДОКУМЕНТАЦИИ

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

# Код программы

|  |
| --- |
| **package** APP;  **import** javax.swing.\*;  **import** java.awt.\*;  **import** java.awt.event.\*;  **import** javax.swing.table.DefaultTableModel;  /\*\*  \* Главный класс приложения "Почта России" для управления клиентами.  \* Содержит графический интерфейс для отображения и управления списком клиентов.  \*  \* **@author** Mikhail  \* **@version** 1.1  \*/  **public** **class** Program {    // Объявляем компоненты как поля класса для доступа из слушателей  **private** JTextField clientNameField;  **private** JTextField phoneField;  **private** JTextField addressField;  **private** JTextField newspaperField;  **private** JTextField searchField;  **private** DefaultTableModel model;  **private** JTable clientsTable;  **private** JComboBox<String> surnameComboBox;    /\*\*  \* Создает и отображает главное окно приложения с интерфейсом управления клиентами.  \*/    **public** **void** show() {  JFrame mainFrame = **new** JFrame("Pochta Rossii");  mainFrame.setSize(800, 600);  mainFrame.setLocationRelativeTo(**null**);  mainFrame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  mainFrame.setLayout(**new** BorderLayout());    // Создание модели и таблицы  String[] columns = {"Клиент", "Телефон", "Адрес", "Газета"};  Object[][] data = {  {"Ivanov", "451-50-70", "Kommunorský 22", "Rastishka"},  {"Frolov", "789-63-45", "Lesnaya 44", "Basketbolchik"},  {"Petrov", "225-25-52", "Sodovaya 13", "Modelist"}  };    model = **new** DefaultTableModel(data, columns) {  @Override  **public** **boolean** isCellEditable(**int** row, **int** column) {  **return** **false**;  }  };    clientsTable = **new** JTable(model);  JScrollPane scroll = **new** JScrollPane(clientsTable);    // Создание панели инструментов  JToolBar toolBar = **new** JToolBar();  toolBar.setFloatable(**false**);    // Создание кнопок  JButton addButton = createButton("ADD.png", "Добавить");  JButton editButton = createButton("EDIT.png", "Редактировать");  JButton deleteButton = createButton("Recycle.jpg", "Удалить");  JButton searchButton = createButton("SEARCH.png", "Поиск");    // Добавляем слушателей для кнопок  setupButtonListeners(addButton, editButton, deleteButton, searchButton, mainFrame);    toolBar.add(addButton);  toolBar.add(editButton);  toolBar.add(deleteButton);  toolBar.addSeparator();    surnameComboBox = **new** JComboBox<>();  updateSurnameComboBox(model, surnameComboBox);    // Добавляем слушатель для комбобокса  setupComboBoxListener(surnameComboBox, mainFrame);    toolBar.add(**new** JLabel("Фамилия:"));  toolBar.add(surnameComboBox);    // Создание панелей  JPanel inputPanel = createInputPanel();  JPanel searchPanel = createSearchPanel(searchButton);    // Компоновка  mainFrame.add(toolBar, BorderLayout.***NORTH***);  mainFrame.add(scroll, BorderLayout.***CENTER***);    JPanel southPanel = **new** JPanel(**new** BorderLayout());  southPanel.add(inputPanel, BorderLayout.***NORTH***);  southPanel.add(searchPanel, BorderLayout.***SOUTH***);  mainFrame.add(southPanel, BorderLayout.***SOUTH***);    // Добавляем слушатель для таблицы  setupTableListener(clientsTable, mainFrame);    mainFrame.setVisible(**true**);  }    /\*\*  \* Настраивает слушателей событий для кнопок интерфейса.  \*  \* **@param** addButton кнопка добавления клиента  \* **@param** editButton кнопка редактирования клиента  \* **@param** deleteButton кнопка удаления клиента  \* **@param** searchButton кнопка поиска клиента  \* **@param** parent родительское окно для диалоговых сообщений  \*/  **private** **void** setupButtonListeners(JButton addButton, JButton editButton,  JButton deleteButton, JButton searchButton,  JFrame parent) {    /\*\*  \* Слушатель для кнопки "Добавить".  \* Реагирует на событие клика мыши, извлекает данные из полей ввода  \* и добавляет нового клиента в таблицу.  \*/  addButton.addActionListener(**new** ActionListener() {  @Override  **public** **void** actionPerformed(ActionEvent e) {  String name = clientNameField.getText().trim();  String phone = phoneField.getText().trim();  String address = addressField.getText().trim();  String newspaper = newspaperField.getText().trim();    // Проверяем, что все поля заполнены  **if** (name.isEmpty() || phone.isEmpty() || address.isEmpty() || newspaper.isEmpty()) {  JOptionPane.*showMessageDialog*(parent,  "Заполните все поля данных клиента!",  "Ошибка ввода",  JOptionPane.***WARNING\_MESSAGE***);  **return**;  }    // Добавляем новую строку в таблицу  model.addRow(**new** Object[]{name, phone, address, newspaper});    // Обновляем комбобокс с фамилиями  updateSurnameComboBox(model, surnameComboBox);    // Очищаем поля ввода  clearInputFields();    JOptionPane.*showMessageDialog*(parent,  "Клиент '" + name + "' успешно добавлен!",  "Добавление клиента",  JOptionPane.***INFORMATION\_MESSAGE***);  }  });    /\*\*  \* Слушатель для кнопки "Удалить".  \* Реагирует на событие клика мыши, проверяет выделенную строку в таблице  \* и удаляет выбранного клиента после подтверждения.  \*/  deleteButton.addActionListener(**new** ActionListener() {  @Override  **public** **void** actionPerformed(ActionEvent e) {  **int** selectedRow = clientsTable.getSelectedRow();    **if** (selectedRow == -1) {  JOptionPane.*showMessageDialog*(parent,  "Выберите клиента для удаления из таблицы!",  "Ошибка удаления",  JOptionPane.***WARNING\_MESSAGE***);  **return**;  }    String clientName = (String) model.getValueAt(selectedRow, 0);    **int** confirm = JOptionPane.*showConfirmDialog*(parent,  "Вы уверены, что хотите удалить клиента '" + clientName + "'?",  "Подтверждение удаления",  JOptionPane.***YES\_NO\_OPTION***);    **if** (confirm == JOptionPane.***YES\_OPTION***) {  model.removeRow(selectedRow);  updateSurnameComboBox(model, surnameComboBox);  JOptionPane.*showMessageDialog*(parent,  "Клиент '" + clientName + "' удален!",  "Удаление завершено",  JOptionPane.***INFORMATION\_MESSAGE***);  }  }  });    /\*\*  \* Слушатель для кнопки "Поиск".  \* Реагирует на событие клика мыши, выполняет поиск клиента по введенному тексту  \* и выделяет найденные совпадения в таблице.  \*/  searchButton.addActionListener(**new** ActionListener() {  @Override  **public** **void** actionPerformed(ActionEvent e) {  String searchText = searchField.getText().trim().toLowerCase();    **if** (searchText.isEmpty()) {  JOptionPane.*showMessageDialog*(parent,  "Введите текст для поиска!",  "Ошибка поиска",  JOptionPane.***WARNING\_MESSAGE***);  **return**;  }    **boolean** found = **false**;  // Снимаем выделение со всех строк  clientsTable.clearSelection();    // Поиск по всем строкам и столбцам таблицы  **for** (**int** row = 0; row < model.getRowCount(); row++) {  **for** (**int** col = 0; col < model.getColumnCount(); col++) {  String cellValue = model.getValueAt(row, col).toString().toLowerCase();  **if** (cellValue.contains(searchText)) {  // Выделяем найденную строку  clientsTable.addRowSelectionInterval(row, row);  // Прокручиваем таблицу к найденной строке  clientsTable.scrollRectToVisible(clientsTable.getCellRect(row, 0, **true**));  found = **true**;  **break**; // Переходим к следующей строке  }  }  }    **if** (!found) {  JOptionPane.*showMessageDialog*(parent,  "Клиенты по запросу '" + searchText + "' не найдены!",  "Результат поиска",  JOptionPane.***INFORMATION\_MESSAGE***);  } **else** {  JOptionPane.*showMessageDialog*(parent,  "Найдены клиенты по запросу: '" + searchText + "'",  "Поиск завершен",  JOptionPane.***INFORMATION\_MESSAGE***);  }  }  });  }    /\*\*  \* Настраивает слушатель событий для комбобокса выбора фамилии.  \*  \* **@param** comboBox комбобокс с фамилиями клиентов  \* **@param** parent родительское окно для диалоговых сообщений  \*/  **private** **void** setupComboBoxListener(JComboBox<String> comboBox, JFrame parent) {  /\*\*  \* Слушатель для комбобокса выбора фамилии.  \* Реагирует на событие выбора элемента, автоматически заполняет поля ввода  \* данными выбранного клиента и выделяет соответствующую строку в таблице.  \*/  comboBox.addActionListener(**new** ActionListener() {  @Override  **public** **void** actionPerformed(ActionEvent e) {  String selectedSurname = (String) comboBox.getSelectedItem();  **if** (selectedSurname != **null**) {  // Ищем выбранную фамилию в таблице  **for** (**int** i = 0; i < model.getRowCount(); i++) {  **if** (selectedSurname.equals(model.getValueAt(i, 0))) {  // Заполняем поля ввода данными найденного клиента  clientNameField.setText(selectedSurname);  phoneField.setText(model.getValueAt(i, 1).toString());  addressField.setText(model.getValueAt(i, 2).toString());  newspaperField.setText(model.getValueAt(i, 3).toString());    // Выделяем строку в таблице  clientsTable.setRowSelectionInterval(i, i);  clientsTable.scrollRectToVisible(clientsTable.getCellRect(i, 0, **true**));    JOptionPane.*showMessageDialog*(parent,  "Загружены данные клиента: " + selectedSurname,  "Данные загружены",  JOptionPane.***INFORMATION\_MESSAGE***);  **break**;  }  }  }  }  });  }    /\*\*  \* Настраивает слушатель событий для таблицы клиентов.  \*  \* **@param** table таблица с данными клиентов  \* **@param** parent родительское окно для диалоговых сообщений  \*/  **private** **void** setupTableListener(JTable table, JFrame parent) {  /\*\*  \* Слушатель для таблицы клиентов.  \* Реагирует на событие выбора строки мыши, автоматически обновляет комбобокс  \* и выводит информацию о выбранном клиенте.  \*/  table.getSelectionModel().addListSelectionListener(e -> {  **if** (!e.getValueIsAdjusting()) {  **int** selectedRow = table.getSelectedRow();  **if** (selectedRow != -1) {  String selectedName = (String) table.getValueAt(selectedRow, 0);  // Обновляем выбор в комбобоксе  surnameComboBox.setSelectedItem(selectedName);    // Можно выводить дополнительную информацию в консоль для отладки  System.***out***.println("Выбран клиент: " + selectedName);  }  }  });  }    /\*\*  \* Очищает поля ввода данных клиента.  \*/  **private** **void** clearInputFields() {  clientNameField.setText("");  phoneField.setText("");  addressField.setText("");  newspaperField.setText("");  }    /\*\*  \* Создает кнопку с иконкой и подсказкой.  \*  \* **@param** iconName имя файла иконки  \* **@param** tooltip текст подсказки  \* **@return** созданная кнопка  \*/  **private** JButton createButton(String iconName, String tooltip) {  **try** {  JButton button = **new** JButton(**new** ImageIcon("./images/" + iconName));  button.setToolTipText(tooltip);  **return** button;  } **catch** (Exception e) {  **return** **new** JButton(tooltip);  }  }    /\*\*  \* Создает панель для ввода данных клиента.  \*  \* **@return** панель с полями ввода  \*/  **private** JPanel createInputPanel() {  JPanel panel = **new** JPanel(**new** GridLayout(2, 4, 5, 5));  panel.setBorder(BorderFactory.*createTitledBorder*("Данные клиента"));    String[] labels = {"Имя клиента:", "Телефон:", "Адрес:", "Газета:"};  **for** (**int** i = 0; i < labels.length; i++) {  panel.add(**new** JLabel(labels[i]));  JTextField textField = **new** JTextField();  panel.add(textField);    // Сохраняем ссылки на текстовые поля  **switch**(i) {  **case** 0: clientNameField = textField; **break**;  **case** 1: phoneField = textField; **break**;  **case** 2: addressField = textField; **break**;  **case** 3: newspaperField = textField; **break**;  }  }    **return** panel;  }    /\*\*  \* Создает панель для поиска клиентов.  \*  \* **@param** searchButton кнопка поиска  \* **@return** панель поиска  \*/  **private** JPanel createSearchPanel(JButton searchButton) {  JPanel panel = **new** JPanel(**new** BorderLayout(5, 5));  panel.setBorder(BorderFactory.*createTitledBorder*("Поиск клиента"));    searchField = **new** JTextField();  panel.add(**new** JLabel("Поиск:"), BorderLayout.***WEST***);  panel.add(searchField, BorderLayout.***CENTER***);  panel.add(searchButton, BorderLayout.***EAST***);    **return** panel;  }    /\*\*  \* Обновляет комбобокс фамилиями из таблицы.  \*  \* **@param** model модель таблицы  \* **@param** comboBox комбобокс для обновления  \*/  **private** **void** updateSurnameComboBox(DefaultTableModel model, JComboBox<String> comboBox) {  String selected = (String) comboBox.getSelectedItem();  comboBox.removeAllItems();    **for** (**int** i = 0; i < model.getRowCount(); i++) {  comboBox.addItem((String) model.getValueAt(i, 0));  }    **if** (selected != **null** && comboBox.getItemCount() > 0) {  comboBox.setSelectedItem(selected);  }  }    /\*\*  \* Главный метод приложения.  \*  \* **@param** args аргументы командной строки  \*/  **public** **static** **void** main(String[] args) {  SwingUtilities.*invokeLater*(**new** Runnable() {  @Override  **public** **void** run() {  **new** Program().show();  }  });  }  } |

# ВЫВОДЫ

В результате выполнения я изучил способы подключения слушателей событий к графическим компонентам пользовательского интерфейса. Получил навыки работы с различными типами событий,освоил модель "источник-слушатель" и основные интерфейсы слушателей.Выполнил практическое применение методов работы с событиями.