Теория и технология программирования

Программирование на языке Java

Раздел 7. Редакторы текста

Глухих Михаил Игоревич, к.т.н., доц.

mailto: glukhikh@mail.ru

Простой текстовый редактор

- Для построения простого текстового редактора может быть использован компонент JEditorPane - наследник JTextComponent
- Редактор содержит внутри себя **документ**, который может быть получен функцией getDocument и установлен функцией setDocument
- Документ в данном случае отвечает за внутреннее содержание - текст, а редактор - за его отображение и взаимодействие с пользователем

Простой текстовый редактор

```
private JScrollPane scrollPane;
private JEditorPane editorPane;

private MainFrame() {
    super("Текстовый редактор");
    setSize(800, 600);
    editorPane = new JEditorPane();
    scrollPane = new JScrollPane(editorPane);
    add(scrollPane);
    setVisible(true);
    setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
}
```

Работа с файлами

- Основные функции:
  - создание нового документа (New)
  - открытие документа из файла (Open)
  - сохранение текущего документа (Save)
  - сохранение в заданном файле (Save as…)
- Для их выполнения существуют четыре команды меню

Работа с файлами

- У редактора есть два состояния:
  - файл еще не был сохранен (безымянный или новый файл)
  - файл уже был сохранен
- Описать их можно с помощью поля File file
  - Если файл еще не был сохранен, в поле хранится null
  - Если уже был сохранен, поле хранит ссылку на конкретный объект типа File (его можно получить с помощью диалога JFileChooser)

Создание документа

- Сброс текущего файла
  - file = null;
- Сброс текста в редакторе
  - editorPane.setText("");
- Изменение заголовка окна (как правило, имя редактируемого файла включается в заголовок)
  - changeTitle();

Открытие файла

- Создание диалога
  - JFileChooser fileChooser = new JFileChooser(file);
- Задание файлового фильтра
  - fileChooser.addChoosableFileFilter(new TextFileFilter());
- Активация диалога
  - int result = fileChooser.showOpenDialog(this);
- Открытие выбранного файла
  - if (result==JFileChooser.APPROVE_OPTION) {
  - openFile(fileChooser.getSelectedFile());
  - }

Открытие файла

- Для загрузки содержимого файла в текстовый редактор необходимо создать объект класса FileReader, после чего вызвать функцию:
  - editorPane.read(reader, null);
- После того, как файл успешно открыт, можно переустановить текущий файл и заголовок окна

Сохранение файла

```
private void onSave() {
    if (file == null)
        onSaveAs();
    else
        saveFile(file);
}

private void onSaveAs() {
    JFileChooser fileChooser = new JFileChooser(file);
    fileChooser.addChoosableFileFilter(new TextFileFilter());
    int result = fileChooser.showSaveDialog(this);
    if (result==JFileChooser.APPROVE_OPTION) {
        saveFile(fileChooser.getSelectedFile());
    }
}
```

Сохранение файла

- Для сохранения файла применяется следующая функция:
  - editorPane.write(writer);
- Где writer - объект класса FileWriter

Undo и Redo (отменить/вернуть)

- Частая функция редактора - отмена одного или нескольких последних действий (undo), а также возвращение их обратно (redo)
- Для этой цели служит класс UndoManager:
  - addEdit - запомнить очередное **действие** (объект класса UndoableEdit)
  - canUndo, undo - отмена действия
  - canRedo, redo - возвращение действия
  - die - очистка всех запомненных действий (когда?)

Undo и Redo (отменить/вернуть)

- Как получить отменяемые действия UndoableEdit?
- Для этой цели **документу** можно установить UndoableEditListener:
  - document = editorPane.getDocument();
  - document.addUndoableEditListener(
    editListener);
  - …
  - editListener = new UndoableEditListener() {
  - public void undoableEditHappened(
    UndoableEditEvent e) {
  - undoManager.addEdit(e.getEdit());
  - }
  - };

Реализация undo и redo

```
    private void onUndo() {
        if (undoManager.canUndo())
            undoManager.undo();
    }

    private void onRedo() {
        if (undoManager.canRedo())
            undoManager.redo();
    }
```

Контроль наличия изменений

- Можно использовать тот же UndoManager - если есть действия для отмены, значит, файл был изменен
- При создании нового документа, сохранении и открытии файла UndoManager необходимо сбрасывать (die())

Контроль наличия изменений

```
private void onQuit() {
    if (undoManager.canUndo()) {
        int result = JOptionPane.showConfirmDialog(
            this, "Сохранить текущий файл?");
        if (result==JOptionPane.YES_OPTION) {
            onSave();
        } else if (result==JOptionPane.NO_OPTION) {
        } else {
            return;
        }
    }
    System.exit(0);
}
```

Многооконный редактор

- Многооконный редактор позволяет редактировать несколько документов одновременно
- Для этого каждый из них нужно открыть в своем окне
- Один из способов это сделать - использование панели рабочего стола JDesktopPane

Панель рабочего стола

- Панель JDesktopPane обычно используется для управления несколькими внутренними окнами
- Внутренние окна описываются компонентом JInternalFrame
- У панели отсутствует менеджер размещения - поэтому все размеры внутренних окон указываются вручную

Основные возможности рабочего стола

- Получение внутреннего фрейма, выбранного в данный момент
  - desktopPane.getSelectedFrame()
  - может вернуть null, если выбранного фрейма нет
- Добавление нового внутреннего фрейма
  - JInternalFrame frame = new JInternalFrame();
  - frame.setSize(width, height);
  - desktopPane.add(frame);
- Удаление существующего внутреннего фрейма
  - desktopPane.remove(frame);

Основные возможности внутреннего фрейма

- Слушатель событий InternalFrameEvent
  - frame.addInternalFrameListener(
    new InternalFrameAdapter() {
    public void internalFrameClosing
    (InternalFrameEvent ev) {
    …
    }
    }
    );

Переработка редактора

- Ранее, панель прокрутки вместе с панелью редактора являлись полями класса
  MainFrame extends JFrame
- Теперь же, они должны стать полями класса
  DocumentFrame extends JInternalFrame
- В связи с этим, ряд других полей и методов должны «переехать» из MainFrame в DocumentFrame

Переработка редактора - поля

- private JScrollPane scrollPane;
- private JTextPane textPane;
- private HTMLDocument document;
- private JMenuBar menuBar;
- private JMenu fileMenu, editMenu;
- private JMenuItem newMenu, openMenu, saveMenu, …;
- private JMenuItem colorMenu;
- private ActionListener newListener, openListener, …;
- private UndoableEditListener editListener;
- private WindowListener windowListener;
- private UndoManager undoManager;
- private File file = null;

Переработка редактора - методы

- private void initMenu() {}
- private void changeTitle() {}
- private void onNew() {}
- private void onOpen() {}
- private void onEdit(UndoableEditEvent e) {}
- private void onUndo() {}
- private void onRedo() {}
- private void onQuit() {}

Проблема

- Ранее вся функциональность, связанная с управлением документом, находилась в MainFrame
- Теперь она в DocumentFrame
- Что делать, если нам захочется еще каким-либо образом изменить способ управления?

Варианты решения

- Реализовать управление документом в рамках класса-расширения панели JPanel
- Вообще не привязывать управление документом к определенному визуальному классу

Look and Feel

- Совокупность свойств компонентов, влияющих на их отображение (Look) и поведение (Feel)
- Базовый класс LookAndFeel
  - MotifLookAndFeel (Solaris)
  - WindowsLookAndFeel (Windows)
  - MacLookAndFeel (Mac OS/2)
  - MetalLookAndFeel (Java)

Управление Look and Feel

- Проще всего использовать класс UIManager
  - UIManager.setLookAndFeel(className);
- Используемые имена классов
  - UIManager. getSystemLookAndFeelClassName()
  - UIManager. getCrossPlatformLookAndFeelClassName()
  - "com.sun.java.swing.plaf.motif.MotifLookAndFeel"

Смена Look and Feel на-лету

```
viewMenu = new JMenu("Вид");
menuBar.add(viewMenu);
viewGroup = new ButtonGroup();
UIManager.LookAndFeelInfo[] info =
                UIManager.getInstalledLookAndFeels();
viewItems = new JRadioButtonMenuItem[info.length];
ViewMenuListener viewMenuListener = new ViewMenuListener();
for (int i=0; i<info.length; i++) {
    viewItems[i] = new JRadioButtonMenuItem(info[i].getName());
    viewItems[i].setActionCommand(info[i].getClassName());
    viewItems[i].addActionListener(viewMenuListener);
    viewMenu.add(viewItems[i]);
    viewGroup.add(viewItems[i]);
}
```

Слушатель пунктов меню Вид

```
public class ViewMenuListener
    implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        try {
            UIManager.setLookAndFeel(
                e.getActionCommand());
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Слушатель изменения LF свойств

```
private MainFrame() {
  // ...
  UIManager.addPropertyChangeListener(
    new PropertyChangeListener() {
      public void propertyChange(PropertyChangeEvent evt) {
        onLookAndFeelChanged();
      }
    });
}
private void onLookAndFeelChanged() {
  SwingUtilities.updateComponentTreeUI(this);
}
```

Итоги

- Рассмотрены варианты создания текстового редактора
- Рассмотрены компоненты JDesktopPane, JInternalFrame, JEditorPane
- Рассмотрены Look & Feel Java