Praktische GUI Klassen

AutoSuggest

AutoSuggest.java

```
import java.awt.EventQueue;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JComboBox;
import javax.swing.JTextField;

public class AutoSuggest<E> extends JComboBox<E> {

    private final JComboBox<E> comboBox = this;
    private final JTextField textField = (JTextField)
getEditor().getEditorComponent();
    private final List<String> suggestions = new LinkedList<>();
    private boolean hide_flag = false;

    public AutoSuggest() {
        super();

        setEditable(true);

        textField.addKeyListener(new KeyAdapter() {
            @Override
            public void keyTyped(KeyEvent e) {
                EventQueue.invokeLater(() -> {
                    String text = textField.getText();
                    if (text.length() == 0) {
                        comboBox.hidePopup();
                        setModel(new
DefaultComboBoxModel(suggestions.toArray()), "");
                    } else {
                        DefaultComboBoxModel m =
getSuggestedModel(suggestions, text);
                        if (m.getSize() == 0 || hide_flag) {
                            comboBox.hidePopup();
                            hide_flag = false;
                        } else {
                            setModel(m, text);
                            comboBox.showPopup();
```

```
                        }
                    }
                });
            }

            @Override
            public void keyPressed(KeyEvent e) {
                String text = textField.getText();
                int code = e.getKeyCode();
                switch (code) {
                    case KeyEvent.VK_ENTER:
                        if (!suggestions.contains(text)) {
                            suggestions.add(text);
                            Collections.sort(suggestions);
                            setModel(getSuggestedModel(suggestions,
text), text);
                        }
                        hide_flag = true;
                        break;
                    case KeyEvent.VK_ESCAPE:
                        hide_flag = true;
                        break;
                    case KeyEvent.VK_RIGHT:
                        for (int i = 0; i < suggestions.size(); i++) {
                            String str = suggestions.get(i);
                            if (str.startsWith(text)) {
                                comboBox.setSelectedIndex(-1);
                                textField.setText(str);
                                return;
                            }
                        }
                        break;
                    default:
                        break;
                }
            }
        });

        setModel(new DefaultComboBoxModel(suggestions.toArray()), "");
    }

    public void update(Set<String> terms) {
        textField.setText("");

        suggestions.clear();
        suggestions.addAll(terms);

        setModel(new DefaultComboBoxModel(suggestions.toArray()), "");
    }
```

```
    private void setModel(DefaultComboBoxModel defaultComboBoxModel,
String text) {
        comboBox.setModel(defaultComboBoxModel);
        comboBox.setSelectedIndex(-1);
        textField.setText(text);
    }

    private static DefaultComboBoxModel getSuggestedModel(List<String>
list, String text) {
        DefaultComboBoxModel m = new DefaultComboBoxModel();
        list.stream().filter((s) ->
(s.toLowerCase().startsWith(text.toLowerCase()))).forEach((s) -> {
            m.addElement(s);
        });
        return m;
    }

    public JTextField getTextField() {
        return textField;
    }

    public String getText() {
        return textField.getText();
    }

}
```

JHintTextField

JHintTextField.java

```
import java.awt.Color;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Insets;
import java.awt.RenderingHints;
import javax.swing.JTextField;

public class JHintTextField extends JTextField {

    private String hint = "";

    public JHintTextField() {
    }

    public JHintTextField(String hint) {
        setHint(hint);
```

```
    }

    public final void setHint(String hint) {
        if (hint != null) {
            this.hint = hint;
        } else {
            this.hint = "";
        }
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        if (getText().length() == 0) {
            int h = getHeight();
            ((Graphics2D)
g).setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
            Insets ins = getInsets();
            FontMetrics fm = g.getFontMetrics();
            int c0 = getBackground().getRGB();
            int c1 = getForeground().getRGB();
            int m = 0xfefefefe;
            int c2 = ((c0 & m) >>> 1) + ((c1 & m) >>> 1);
            g.setColor(new Color(c2, true));
            g.drawString(hint, ins.left, h / 2 + fm.getAscent() / 2 -
1);
        }
    }
}
```

FocusHighlighter

FocusHighlighter.java

```
import java.awt.Color;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import javax.swing.UIManager;

public class FocusHighlighter implements FocusListener {

    @Override
    public void focusGained(FocusEvent e) {
        e.getComponent().setBackground(Color.YELLOW);
    }
```

```
    @Override
    public void focusLost(FocusEvent e) {
e.getComponent().setBackground(UIManager.getColor("TextField.background
"));
    }
}
```

DisposeWithESC

DisposeWithESC.java

```
import java.awt.event.KeyEvent;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.KeyStroke;

public class DisposeWithESC {

    private DisposeWithESC() {
    }

    public static void attach(JDialog jDialog) {
        jDialog.getRootPane().registerKeyboardAction(e -> {
            jDialog.dispose();
        }, KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0),
JComponent.WHEN_IN_FOCUSED_WINDOW);
    }

}
```

LimitDocumentFilter

LimitDocumentFilter.java

```
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.DocumentFilter;

public class LimitDocumentFilter extends DocumentFilter {

    public final int limit;

    public LimitDocumentFilter(int limit) {
        if (limit <= 0) {
```

```
            throw new IllegalArgumentException("Limit can not be <=
0!");
        }
        this.limit = limit;
    }

    @Override
    public void replace(FilterBypass fb, int offset, int length, String
text, AttributeSet attrs) throws BadLocationException {
        int currentLength = fb.getDocument().getLength();
        int overLimit = (currentLength + text.length()) - limit -
length;
        if (overLimit > 0) {
            text = text.substring(0, text.length() - overLimit);
        }
        if (text.length() >= 0) {
            super.replace(fb, offset, length, text, attrs);
        }
    }
}
```

TableSearchDocumentListener

TableSearchDocumentListener.java

```
import javax.swing.JTextField;
import javax.swing.RowFilter;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;

public class TableSearchDocumentListener implements DocumentListener {

    private final JTextField searchTextField;
    private final TableRowSorter tableRowSorter;

    public TableSearchDocumentListener(JTextField searchTextField,
TableRowSorter tableRowSorter) {
        this.searchTextField = searchTextField;
        this.tableRowSorter = tableRowSorter;
    }

    @Override
    public void insertUpdate(DocumentEvent e) {
        updateTableRowSorter();
    }
```

```
    @Override
    public void removeUpdate(DocumentEvent e) {
        updateTableRowSorter();
    }

    @Override
    public void changedUpdate(DocumentEvent e) {
        updateTableRowSorter();
    }

    private void updateTableRowSorter() {
        String searchFieldText = searchTextField.getText();

        if (searchFieldText != null) {
            String[] keywords = searchFieldText.split(" ");
            tableRowSorter.setRowFilter(new KeywordFilter(keywords));
        }
    }

    public class KeywordFilter extends RowFilter<TableModel, Integer> {

        private final String[] keywords;

        // Constructor for filtering a single keyword
        public KeywordFilter(String keyword) {
            this.keywords = new String[1];
            this.keywords[0] = keyword.toLowerCase();
        }

        // Constructor for filtering an array of keywords
        // Only the rows that contain every keyword are displayed
        public KeywordFilter(String[] keywords) {
            this.keywords = keywords;
            // convert all keywords to lower case
            for (int i = 0; i < keywords.length; i++) {
                keywords[i] = keywords[i].toLowerCase();
            }
        }

        @Override
        public boolean include(RowFilter.Entry entry) {
            // Does the row contain all keywords? Assume true and set
to false if
            // one is missing
            boolean containsAll = true;
            for (int i = 0; containsAll == true && i < keywords.length;
i++) {
                // Is the keyword contained in one of the columns?
Assume false and
                // set to true if it is found
```

```
                boolean containsThis = false;
                for (int j = 0; containsThis == false && j <
entry.getValueCount(); j++) {
                    if
(entry.getStringValue(j).toLowerCase().contains(keywords[i])) {
                        containsThis = true;
                    }
                }
                if (!containsThis) {
                    containsAll = false;
                }
            }

            return containsAll;
        }

    }

}
```