```java
package view;

import interfaces.Controller;
import interfaces.UserChoices;
import interfaces.ViewInterface;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.File;
import java.io.IOException;
import java.nio.file.FileVisitResult;
import java.nio.file.FileVisitor;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.attribute.BasicFileAttributes;
import java.nio.file.SimpleFileVisitor;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.regex.PatternSyntaxException;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.text.BadLocationException;
import javax.swing.text.DefaultCaret;
import javax.swing.text.Document;

import net.miginfocom.swing.MigLayout;

import com.google.common.base.Joiner;
import com.google.common.base.Predicate;
import com.google.common.collect.Iterables;

public class InteractionPanel extends JPanel implements ViewInterface, UserChoices {

    private String lang, style, content;
    private String prompt = "Type a regex, ex [cpp$] [,java$] ...";
    private boolean manualInsert = false;
    private Controller control;
    private JTextArea dirContent;
    private File dir;
    private final Map<String, String> srcFiles;

    public InteractionPanel() {
```

```
63          super(new MigLayout("fill", "[]5::5[]", "[]5::5[]5::5[]"));
64          setBorder(BorderFactory.createRaisedSoftBevelBorder());
65          setBackground(new Color(Integer.parseInt("5E303B", 16)));
66          srcFiles = new TreeMap<String, String>();
67      }
68
69      @Override
70      public List<String> getFiles() {
71          List<String> selectedFiles = Arrays.asList(dirContent.getText().split("\n"));
72          for (int t = 0; t < selectedFiles.size(); t++) {
73              selectedFiles.set(t, Paths
74                      .get(srcFiles.get(selectedFiles.get(t)), selectedFiles.get(t)).toString());
75          }
76          return selectedFiles;
77      }
78
79      @Override
80      public String getLanguage() {
81          return lang;
82      }
83
84      @Override
85      public String getStyle() {
86          return style;
87      }
88
89      @Override
90      public void initComponents() {
91          JButton button = new JButton("Convert!");
92          button.addActionListener(control);
93          add(button, "south, tag ok, spanx, center, height 30::");
94
95          addProgressReport();
96          addInteraction();
97      }
98
99      @Override
100     public void showMessage(String msg, int msgType) {
101         JOptionPane.showMessageDialog(this, msg, "Message", msgType);
102     }
103
104     @Override
105     public File getRootFolder() {
106         return dir;
107     }
108
109     @Override
110     public void setController(Controller listener) {
111         control = listener;
112     }
113
114     /* ------------------Private methods begin--------------------- */
115     private void addProgressReport() {
116
117         JTextArea textArea = new JTextArea(10, 10);
118         textArea.setBorder(BorderFactory.createLoweredSoftBevelBorder());
119         textArea.setBackground(new Color(Integer.parseInt("D3DBBD", 16)));
120         textArea.setVisible(true);
121         textArea.setLineWrap(true);
122         textArea.setWrapStyleWord(true);
123         textArea.setEditable(false);
124         ((DefaultCaret) textArea.getCaret()).setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
```

```
125
126         JScrollPane scrollPane = new JScrollPane(textArea);
127         scrollPane.setBorder(BorderFactory.createTitledBorder("Progress"));
128
129         add(scrollPane, "width 40%::, spany, growy, pushy");
130     }
131
132     /**
133      * Used to select the correct language from the list of possible languages
134      *
135      * @return The combo box used for the selection
136      */
137     private JComboBox<String> getLangSelect() {
138         JComboBox<String> langSelect =
139                 new JComboBox<String>(control.getLanguages().toArray(new String[0]));
140         langSelect.setBorder(BorderFactory.createTitledBorder("Language"));
141         langSelect.addItemListener(new ItemListener() {
142             @Override
143             public void itemStateChanged(ItemEvent e) {
144                 if (e.getStateChange() == ItemEvent.SELECTED)
145                     lang = (String) e.getItem();
146             }
147         });
148         return langSelect;
149     }
150
151     /**
152      * Used to select the correct style from the list of possible styles
153      *
154      * @return The combo box used for the selection
155      */
156     private JComboBox<String> getStyleSelect() {
157         JComboBox<String> styleSelect =
158                 new JComboBox<String>(control.getStyles().toArray(new String[0]));
159         styleSelect.setBorder(BorderFactory.createTitledBorder("Format Style"));
160         styleSelect.addItemListener(new ItemListener() {
161             @Override
162             public void itemStateChanged(ItemEvent e) {
163                 if (e.getStateChange() == ItemEvent.SELECTED)
164                     style = (String) e.getItem();
165             }
166         });
167         return styleSelect;
168     }
169
170     /**
171      * @return a button for opening a folder in which the files to be read are contained
172      */
173     private JButton getFolderButton() {
174         JButton fSelect = new JButton("Select Folder");
175         fSelect.addActionListener(new ActionListener() {
176             private final JFileChooser fileChooser = new JFileChooser(".");
177
178             @Override
179             public void actionPerformed(ActionEvent e) {
180                 fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
181                 int action = fileChooser.showOpenDialog(InteractionPanel.this);
182                 if (action == JFileChooser.APPROVE_OPTION) {
183                     dir = fileChooser.getSelectedFile();
184                     fileChooser.setCurrentDirectory(fileChooser.getCurrentDirectory());
185                     srcFiles.clear();
186                     try {
```

```
                        Files.walkFileTree(Paths.get(dir.toURI()), walkDirectoryTree);
                        content = Joiner.on('\n').join(srcFiles.keySet());
                        dirContent.setText(content);
                    } catch (IOException e1) {
                    }
                }
            }
        });
        return fSelect;
    }

    /**
     * @return textfield that enables users to filter only files that user wants
     */
    private JTextField getFilesFilter() {
        JTextField textField = new JTextField(prompt);
        textField.setBorder(BorderFactory.createTitledBorder("Filter File extensions"));
        textField.setBackground(new Color(Integer.parseInt("4764EF", 16)));
        textField.addFocusListener(new FocusListener() {
            @Override
            public void focusGained(FocusEvent arg) {
                JTextField textField = (JTextField) arg.getSource();
                if (textField.getText().equals(prompt)) {
                    manualInsert = true;
                    textField.setText("");
                }
            }

            @Override
            public void focusLost(FocusEvent arg) {
                JTextField textField = (JTextField) arg.getSource();
                if (textField.getText().isEmpty()) {
                    manualInsert = true;
                    textField.setText(prompt);
                }
            }
        });

        textField.getDocument().addDocumentListener(new DocumentListener() {
            @Override
            public void changedUpdate(DocumentEvent arg) {}

            @Override
            public void insertUpdate(DocumentEvent arg) {
                filter(arg);
            }

            @Override
            public void removeUpdate(DocumentEvent arg) {
                filter(arg);
            }
        });
        return textField;
    }

    private void addInteraction() {
        JComboBox<String> langSelect = getLangSelect();
        lang = langSelect.getItemAt(0);

        JComboBox<String> styleSelect = getStyleSelect();
        style = styleSelect.getItemAt(0);

```

```
249         JButton folderButton = getFolderButton();
250 
251         JTextArea textArea = new JTextArea(10, 10);
252         textArea.setBackground(new Color(Integer.parseInt("5F2D23", 16)));
253         textArea.setForeground(new Color(Integer.parseInt("12C4B5", 16)));
254         textArea.setBorder(BorderFactory.createLoweredSoftBevelBorder());
255         textArea.setVisible(true);
256         textArea.setEditable(false);
257         dirContent = textArea;
258 
259         JScrollPane scrollPane = new JScrollPane(textArea);
260         scrollPane.setBorder(BorderFactory.createTitledBorder("Content"));
261 
262         JTextField filesFilter = getFilesFilter();
263 
264         add(langSelect, "width 29%::, height pref + 5::, top, split2");
265         add(styleSelect, "width 29%::, height pref + 5::, top, wrap");
266         add(folderButton, "top, width 29%::, center, height 30::, wrap");
267         add(scrollPane, "spanx, grow, push, center");
268         add(filesFilter, "growx, center, height pref + 5::");
269     }
270 
271     /**
272      * Compiles a pattern entered by the user to ensure it is a valid regex
273      *
274      * @param regex the pattern entered by the user
275      * @return A pattern object to be used for filtering files
276      */
277     private Pattern compileRegex(String regex) {
278         try {
279             Pattern p = Pattern.compile(regex);
280             return p;
281         } catch (PatternSyntaxException ex) {
282         }
283         return null;
284     }
285 
286     /**
287      * Filters the user's input and shows only files that match the regex entered by the user
288      *
289      * @param evt the document event which signals that the user wrote something
290      * @throws BadLocationException
291      */
292     private void filter(DocumentEvent evt) {
293         // Only check for input from user
294         if (!manualInsert && dirContent.getDocument().getLength() > 0) {
295             try {
296                 Document document = evt.getDocument();
297                 String text = document.getText(0, document.getLength()).trim();
298                 Pattern p = compileRegex(text);
299                 if (p == null)
300                     return;
301 
302                 final Matcher matcher = p.matcher("");
303                 String matchedFiles =
304                         Joiner.on('\n').join(
305                                 Iterables.filter(srcFiles.keySet(), new Predicate<String>() {
306                                     @Override
307                                     public boolean apply(String input) {
308                                         return matcher.reset(input).find();
309                                     }
310                                 }));
```

```
311
312                 if (!matchedFiles.isEmpty()) {
313                     dirContent.setText(matchedFiles);
314                 } else
315                     dirContent.setText(content);
316             } catch (BadLocationException ble) {
317             }
318         } else
319             manualInsert = false;
320     }
321
322     /**
323      * A simple file visitor that enables one to walk a directory tree
324      */
325     private final FileVisitor<Path> walkDirectoryTree = new SimpleFileVisitor<Path>() {
326
327         // TODO User might want to choose files based on some attribute of the directory
328         // So it might help to let directory names be included in the output.
329         //
330         @Override
331         public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
332             srcFiles.put(file.getFileName().toString(), file.getParent().toString());
333             return FileVisitResult.CONTINUE;
334         }
335     };
336 }
```