We were able to get the code running, and have completed 6 change requests.

Names and NSIDs: Corey Hickson (crh208) Benjamin Hingston (bvh895) Evan Salter (evs162)
Project: jEdit
Change Number: 1
Date: 03/31/2018

Modify the Splash Window

1. Change Request:

Add the names and emails of your group members to it. And add moving text as the same effect shown in "About jEdit" dialog. Adjust the scrolling speed so that all text can be shown.

2. Concept Location:

I used IntelliJ Find-in-Path to search "splash screen" in the source code. From finding *showSplashScreen()* I used IntelliJ's go-to-definition (cmd-click) to *GUIUtilities*; there I found the JComponent *SplashScreen*. Searching for "jEdit is brought to you" I found scrolling text in the about dialog in *jedit_gui.props*. There I looked up "about.text" that lead me to *AboutDialog*.

Table 1. The list of all the classes visited during concept location.

#	File name	Tool used	Located?	Comments
1	jEdit.java	Find in Path	Propagating	Main class of the editor. Inits the editor and progresses the splash screen through GUIUtilities.
2	GUIUtilities.java	Go To Definition	Propagating	Creates splash screen and main functions for GUIs.
3	SplashScreen.java	Go To Definition	Changed	Component for displaying splash screen.
4	jedit_gui.props	Find in Path	Unchanged	Contains text for scrolling dialog.
5	AboutDialog.java	Find in Path	Unchanged	Displays the dialog with scrolling text animation.

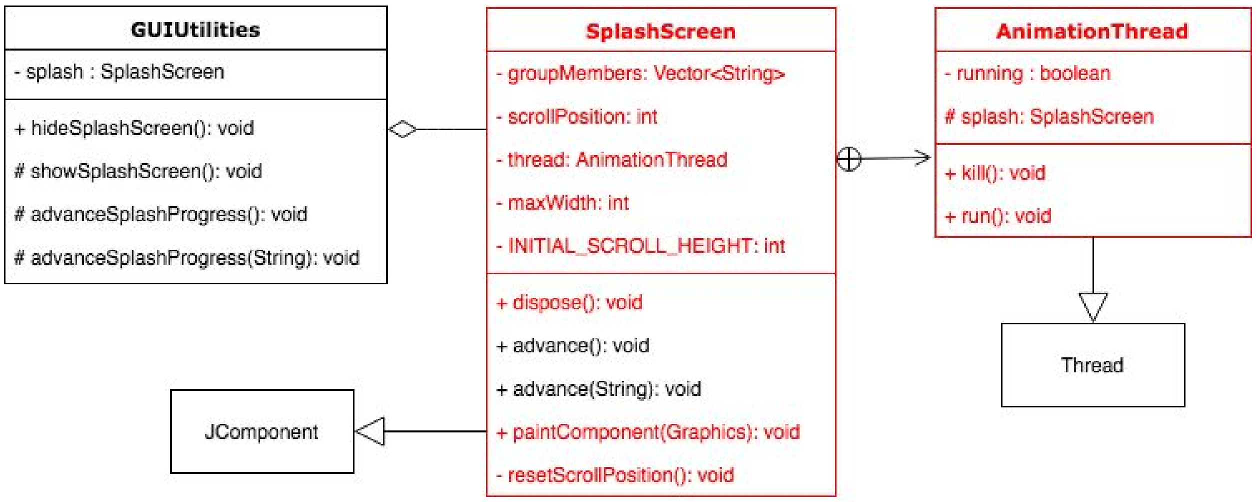
3. Impact Analysis:

Table 2. The list of all the classes visited during impact analysis.

#	Class name	Tool used	Impacted?	Comments
1	SplashScreen	Go To Definition	Yes, changed	Component for displaying splash screen.
2	GUIUtilities	Find Usages	No, unchanged	Has utility functions for creating and progressing splash screen.

4. Learning process:

Figure 1 Example of Class Diagram


GUIUtilities
- splash : SplashScreen
+ hideSplashScreen(): void
# showSplashScreen(): void
# advanceSplashProgress(): void
# advanceSplashProgress(String): void
SplashScreen
- groupMembers: Vector<String>
- scrollPosition: int
- thread: AnimationThread
- maxWidth: int
- INITIAL_SCROLL_HEIGHT: int
+ dispose(): void
+ advance(): void
+ advance(String): void
+ paintComponent(Graphics): void
- resetScrollPosition(): void
AnimationThread
- running : boolean
# splash: SplashScreen
+ kill(): void
+ run(): void
Thread
JComponent

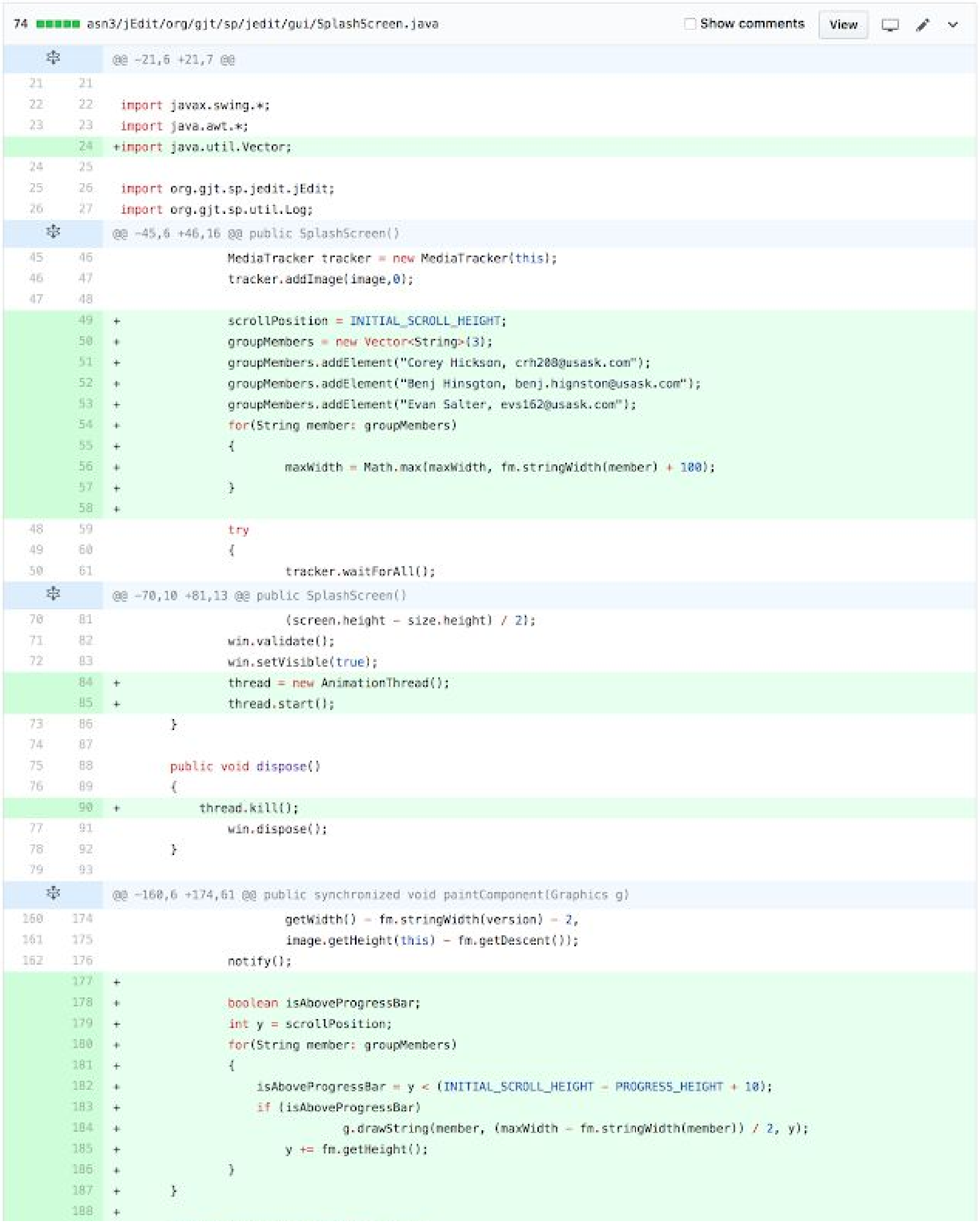
5. Description of the implementation:

To display our team info I added changes to the constructor of *SplashScreen*. I added *groupMembers* vector to hold the team info. I added *AnimationThread*, an inner class to *SplashScreen* to handle the scrolling animation similar to About dialog.

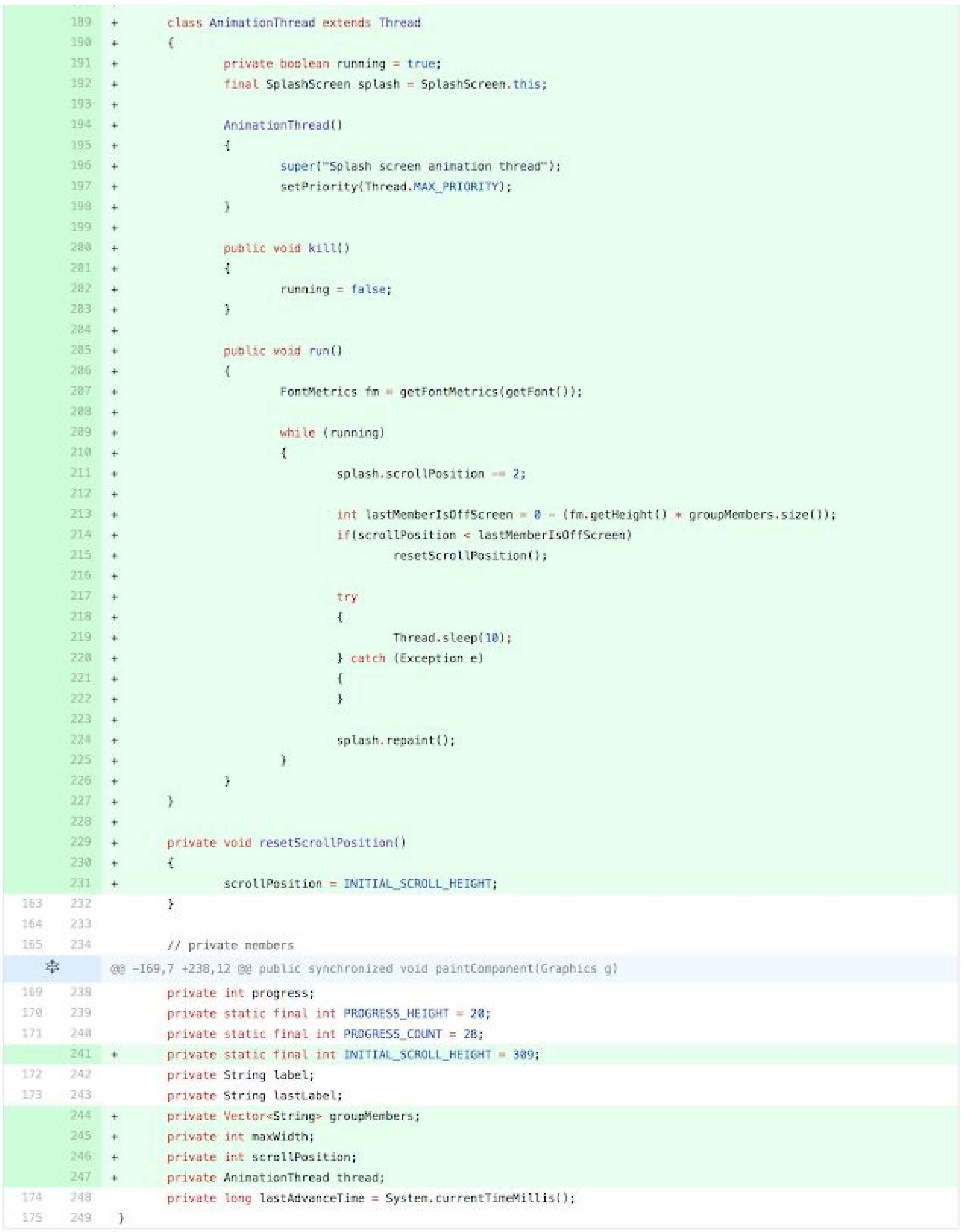
Changes and additions to **org.gjt.sp.jedit.gui.SpashScreen**:

- method **SplashScreen()**: initialize group info and scroll position, start animation thread
- method **dispose()**: destroy animation thread
- method **paintComponent(Graphics)**: logic to paint team info
- method **resetScrollPosition()**: added to reset scrolling text
- inner class **AnimationThread**
    - method **kill()**: stops animation thread from happening
    - method **run()**: updates scroll position and repaints splash screen
    - attribute **splash**: static reference to outer class SplashScreen

7. Highlighted Source Code:



```
@@ -21,6 +21,7 @@

 import javax.swing.*;
 import java.awt.*;
+import java.util.Vector;

 import org.gjt.sp.jedit.jEdit;
 import org.gjt.sp.util.Log;
@@ -45,6 +46,16 @@ public SplashScreen()
                MediaTracker tracker = new MediaTracker(this);
                tracker.addImage(image,0);

+               scrollPosition = INITIAL_SCROLL_HEIGHT;
+               groupMembers = new Vector<String>(3);
+               groupMembers.addElement("Corey Hickson, crh208@usask.com");
+               groupMembers.addElement("Benj Hinsgton, benj.hignston@usask.com");
+               groupMembers.addElement("Evan Salter, evs162@usask.com");
+               for(String member: groupMembers)
+               {
+                       maxWidth = Math.max(maxWidth, fm.stringWidth(member) + 100);
+               }
+
                try
                {
                        tracker.waitForAll();
@@ -70,10 +81,13 @@ public SplashScreen()
                        (screen.height - size.height) / 2);
                win.validate();
                win.setVisible(true);
+               thread = new AnimationThread();
+               thread.start();
        }

        public void dispose()
        {
+           thread.kill();
                win.dispose();
        }

@@ -160,6 +174,61 @@ public synchronized void paintComponent(Graphics g)
                        getWidth() - fm.stringWidth(version) - 2,
                        image.getHeight(this) - fm.getDescent());
                notify();
+
+               boolean isAboveProgressBar;
+               int y = scrollPosition;
+               for(String member: groupMembers)
+               {
+                   isAboveProgressBar = y < (INITIAL_SCROLL_HEIGHT - PROGRESS_HEIGHT + 10);
+                   if (isAboveProgressBar)
+                               g.drawString(member, (maxWidth - fm.stringWidth(member)) / 2, y);
+                       y += fm.getHeight();
+               }
+       }
+
```



```diff
+	class AnimationThread extends Thread
+	{
+		private boolean running = true;
+		final SplashScreen splash = SplashScreen.this;
+
+		AnimationThread()
+		{
+			super("Splash screen animation thread");
+			setPriority(Thread.MAX_PRIORITY);
+		}
+
+		public void kill()
+		{
+			running = false;
+		}
+
+		public void run()
+		{
+			FontMetrics fm = getFontMetrics(getFont());
+
+			while (running)
+			{
+				splash.scrollPosition -= 2;
+
+				int lastMemberIsOffScreen = 0 - (fm.getHeight() * groupMembers.size());
+				if(scrollPosition < lastMemberIsOffScreen)
+					resetScrollPosition();
+
+				try
+				{
+					Thread.sleep(10);
+				} catch (Exception e)
+				{
+				}
+
+				splash.repaint();
+			}
+		}
+	}
+
+	private void resetScrollPosition()
+	{
+		scrollPosition = INITIAL_SCROLL_HEIGHT;
 	}
 
 	// private members
@@ -169,7 +238,12 @@ public synchronized void paintComponent(Graphics g)
 	private int progress;
 	private static final int PROGRESS_HEIGHT = 20;
 	private static final int PROGRESS_COUNT = 28;
+	private static final int INITIAL_SCROLL_HEIGHT = 309;
 	private String label;
 	private String lastLabel;
+	private Vector<String> groupMembers;
+	private int maxWidth;
+	private int scrollPosition;
+	private AnimationThread thread;
 	private long lastAdvanceTime = System.currentTimeMillis();
 }
```

Names and NSIDs: Corey Hickson (crh208) Benjamin Hingston (bvh895) Evan Salter (evs162)
Project: jEdit
Change Number: 2
Date: 03/27/2018

Zoom the text editor

1. Change Request*:*

This change request asked that I add the ability to increase and decrease the size of the text in the editor. This is to be implemented through the use of menu items in the View menu, labelled “zoom+” and “zoom-”.

2. Concept Location:

During concept location, I mainly used the Search functionality of Eclipse. This is because the menu items aren’t defined in .java files. The menu is built in View.java, but it uses a helper to create a menu bar called “view.mbar”. In order to find that, I had to search for that string to find jedit_gui.props, which actually defined the menu bar items. However, once it came time to actually implement the functionality, I could use more features like “Go to Definition” and “Find Symbol”, because I was navigating through .java files at that point.

Table 1. The list of all the classes visited during concept location.

#	File name	Tool used	Located?	Comments
1	View.java	Search	Unchanged	This is related to managing a “view” (i.e. window). It builds the menu bar but doesn’t contain which menu items to add
2	basics.xml	Search	Unchanged	This lists the toolbar menu items for documentation, but has nothing to do with the toolbar implementation
3	jedit_gui.props	Search	Changed	Defines the toolbar menu items
4	actions.xml	Search	Changed	Defines the action that the toolbar menu items execute
5	TextArea.java	Find Symbol	Changed	Defines how the text editor should display text (i.e. font)
6	Font.java	Go-to-definition (ctrl-click)	Propagating	Told me what parameters are passed to the constructor of a Font object
7	jedit_keys.props	Search	Changed	Sets the keyboard shortcuts for menu items

3. Impact Analysis:

Table 2. The list of all the classes visited during impact analysis.

#	Class name	Tool used	Impacted?	Comments
1	TextArea	Find Symbol	Yes. I added new functions in this file to support the new feature.	Defines how the text editor should display text (i.e. font)
2	View	Search	Yes. This references the files in jedit_gui.props, jedit_keys.props, and actions.xml to build the menubar items	
3	EditPane	"Find References" on TextArea	No. Only new functions were added to zoom. These are only called for the TextArea in the main editor view. However, this functionality is now able to be implemented in other TextArea instances if required.	The pane of the UI that holds the TextArea that I edited.
4	TextAreaInputHandler	"Find References" on TextArea		Processes input events in TextAreas
5	Registers	"Find References" on TextArea		Manages clipboard-like text registers.
6	org.gjt.sp.jedit.textarea.*	"Find References" on TextArea	No. There are a lot of usages of TextArea in this package, but without using the new functions I added, there will be no effect.	All sorts of supporting functionality for the TextArea class.

4. Learning process:

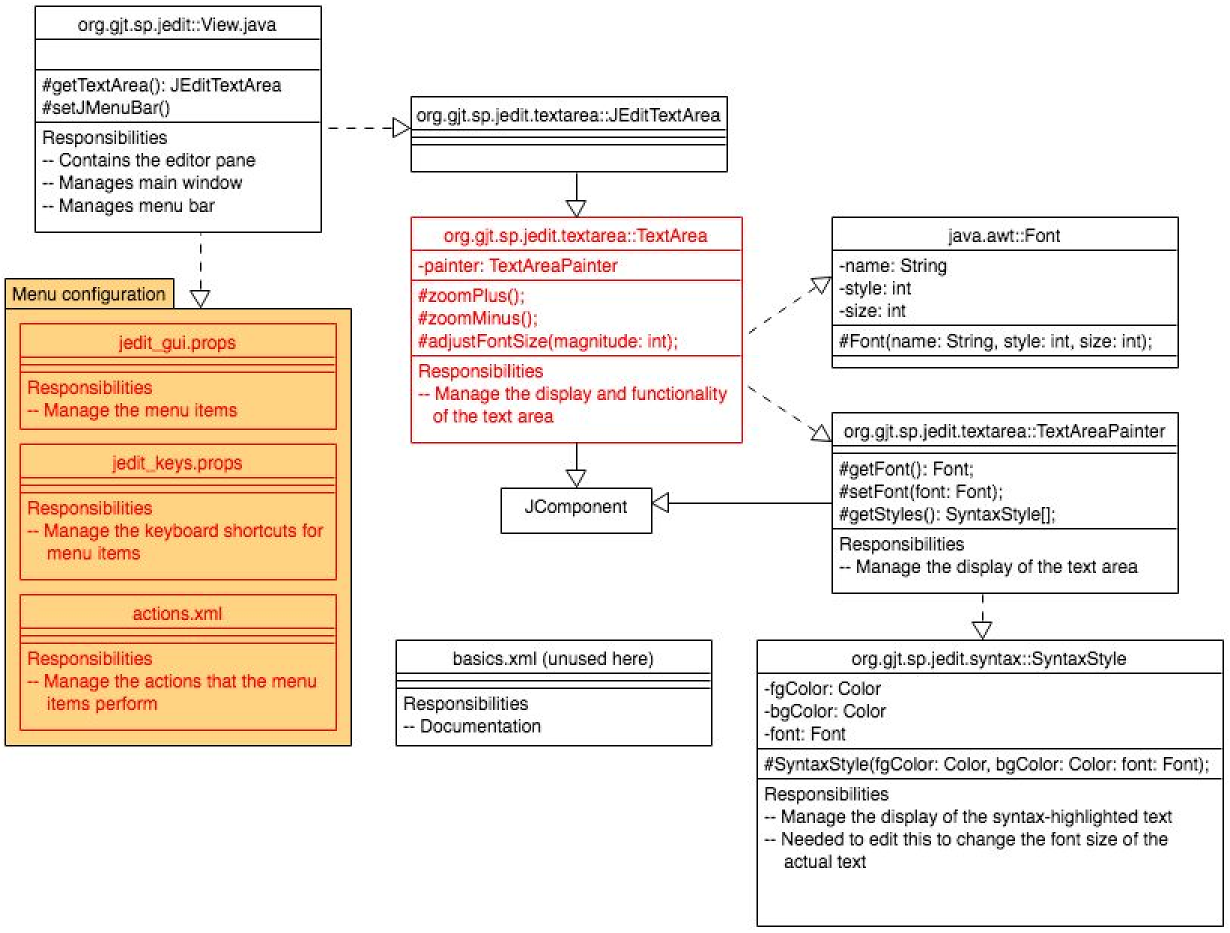

org.gjt.sp.jedit::View.java
#getTextArea(): JEditTextArea
#setJMenuBar()
Responsibilities
-- Contains the editor pane
-- Manages main window
-- Manages menu bar
org.gjt.sp.jedit.textarea::JEditTextArea
org.gjt.sp.jedit.textarea::TextArea
-painter: TextAreaPainter
#zoomPlus();
#zoomMinus();
#adjustFontSize(magnitude: int);
Responsibilities
-- Manage the display and functionality of the text area
java.awt::Font
-name: String
-style: int
-size: int
#Font(name: String, style: int, size: int);
Menu configuration
jedit_gui.props
Responsibilities
-- Manage the menu items
jedit_keys.props
Responsibilities
-- Manage the keyboard shortcuts for menu items
actions.xml
Responsibilities
-- Manage the actions that the menu items perform
JComponent
org.gjt.sp.jedit.textarea::TextAreaPainter
#getFont(): Font;
#setFont(font: Font);
#getStyles(): SyntaxStyle[];
Responsibilities
-- Manage the display of the text area
basics.xml (unused here)
Responsibilities
-- Documentation
org.gjt.sp.jedit.syntax::SyntaxStyle
-fgColor: Color
-bgColor: Color
-font: Font
#SyntaxStyle(fgColor: Color, bgColor: Color: font: Font);
Responsibilities
-- Manage the display of the syntax-highlighted text
-- Needed to edit this to change the font size of the actual text

Figure 1 Example of the class diagram

5. Description of the implementation:

To implement the functionality of adjusting the size of the text in the editor, I added the following 3 functions to the org.gjt.sp.jedit.textarea.TextArea class:

- adjustFontSize(int magnitude): Adjusts the font size by `magnitude` px (can be positive or negative, to increase or decrease the size)
- zoomPlus(): Increases the font size by 3 pixels
- zoomMinus(): Decreases the font size by 3 pixels, ensuring it doesn't go smaller than 1px

To create the menu items in the view menu, I made the following changes:

- jedit_gui.props
  - Create new menu identifiers called `zoom-plus` and `zoom-minus`
  - Set the menu item labels using e.g. `zoom-plus.label=zoom+`
- jedit_keys.props
  - Added keyboard shortcuts to allow you to zoom in using Ctrl+= and Ctrl+-
- actions.xml

- Set the actions for each of the new menu items

6. **Sources:**

7. **Highlighted Source Code:**

org/gjt/sp/jedit/actions.xml:
```
<ACTION NAME="new-plain-view">
     <CODE>
          jEdit.newView(view,buffer,true);
     </CODE>
</ACTION>

<ACTION NAME="zoom-plus">
     <CODE>
          textArea.zoomPlus();
     </CODE>
</ACTION>

<ACTION NAME="zoom-minus">
     <CODE>
          textArea.zoomMinus();
     </CODE>
</ACTION>

<ACTION NAME="new-view">
     <CODE>
          jEdit.newView(view);
     </CODE>
</ACTION>
```

org/gjt/sp/jedit/jedit_gui.props:

```
#{{{ View menu
view=new-view \
     new-plain-view \
     close-view \
     - \
     zoom-plus \
     zoom-minus \
     - \
     prev-buffer \
     next-buffer \
     recent-buffer \
     show-buffer-switcher \
     - \
     toggle-line-numbers \
     - \
     %scrolling \
     %splitting \
     %docking
view.label=$View
new-view.label=New $View
new-plain-view.label=Ne$w Plain View
zoom-plus.label=zoom+
zoom-minus.label=zoom-
close-view.label=$Close View
prev-buffer.label=Go to $Previous Buffer
next-buffer.label=Go to $Next Buffer
recent-buffer.label=Go to $Recent Buffer
show-buffer-switcher.label=Show $Buffer Switcher
toggle-line-numbers.label=$Line Numbers
```

org/gjt/sp/jedit/jedit_keys.props:

```
recent-buffer.shortcut=C+BACK_QUOTE
select-block.shortcut=C+OPEN_BRACKET
match-bracket.shortcut=C+CLOSE_BRACKET
expand-abbrev.shortcut=C+SEMICOLON
quick-search.shortcut=C+COMMA
hypersearch.shortcut=C+PERIOD
scroll-up-line.shortcut=C+QUOTE
scroll-down-line.shortcut=C+SLASH
toggle-multi-select.shortcut=C+BACK_SLASH
zoom-plus.shortcut=C+EQUALS
zoom-minus.shortcut=C+MINUS
#}}}
```

org/gjt/sp/jedit/textarea/TextArea.java:

```
//{{{ adjustFontSize() method
/**
 * Adjusts the current font size by the given magnitude
  * @param magnitude the amount to adjust the font size by (can
be positive or negative)
 */
private void adjustFontSize(int magnitude) {
     Font oldFont = painter.getFont();
     int newSize = Math.max(1, oldFont.getSize() + magnitude);
     Font     newFont     =     new     Font(oldFont.getName(),
oldFont.getStyle(), newSize);
     painter.setFont(newFont);
     painter.getStyles()[0]   =   new   SyntaxStyle(Color.black,
Color.white, newFont);
} //}}}

//{{{ zoomPlus() method
/**
 * Increases the font size of the editor
 */
public void zoomPlus() {
     adjustFontSize(3);
} //}}}

//{{{ zoomMinus() method
/**
 * Decreases the font size of the editor
 */
public void zoomMinus() {
     adjustFontSize(-3);
} //}}}
```

Names and NSIDs: Corey Hickson (crh208), Benj Hingston (bvh895), Evan Salter (evs162)
Project: jEdit
Change Number: *4*
Date: *03/19/2018*

Add timestamps to log

1. Change Request*:*

"Locate where the activity.log is. Currently there are no timestamps in the log file. Add timestamps to all kinds of messages."

2. Concept Location:

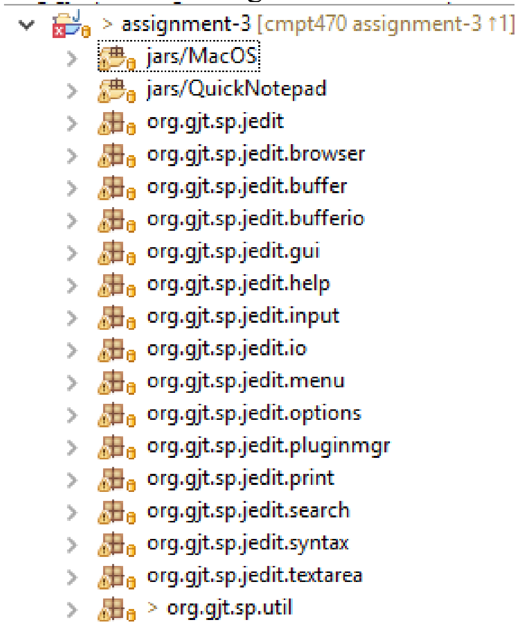
I searched for Log in the main file (jEdit) and came across a Log class. I went to the Log class and discovered where it was writing the log messages.

Table 1. The list of all the classes visited during concept location.

#	File name	Tool used	Located?	Comments
1	jEdit.java	Given in the spec	Propagating	Uses the Log class
2	Log.java	Find Class	Changed	Contains the code we need to change

3. Impact Analysis:

I searched for usages of the Log class and came up with a large list of packages, each with their own set of classes that had uses of Log within them.



> assignment-3 [cmpt470 assignment-3 ↑1]
jars/MacOS
jars/QuickNotepad
org.gjt.sp.jedit
org.gjt.sp.jedit.browser
org.gjt.sp.jedit.buffer
org.gjt.sp.jedit.bufferio
org.gjt.sp.jedit.gui
org.gjt.sp.jedit.help
org.gjt.sp.jedit.input
org.gjt.sp.jedit.io
org.gjt.sp.jedit.menu
org.gjt.sp.jedit.options
org.gjt.sp.jedit.pluginmgr
org.gjt.sp.jedit.print
org.gjt.sp.jedit.search
org.gjt.sp.jedit.syntax
org.gjt.sp.jedit.textarea
> org.gjt.sp.util

Figure 1 List of packages found with the Log class

This makes sense because pretty much every event is logged so almost every spot would need it. I've just included a few of those spots in the table below, since all of them will be impacted, but in the same way (they will not include the Timestamp in the log).

Table 2. The list of all the classes visited during impact analysis.

#	Class name	Tool used	Impacted?	Comments
1	jEdit	Find usages in Workspace	Yes	This is where a lot of the init is setup so if there's any problems, they'll likely come up in here
2	BrowserIORequest	Find usages in Workspace	Yes	This is an example of a spot which will do the Log.Log() and have the Timestamp

4. Learning process:

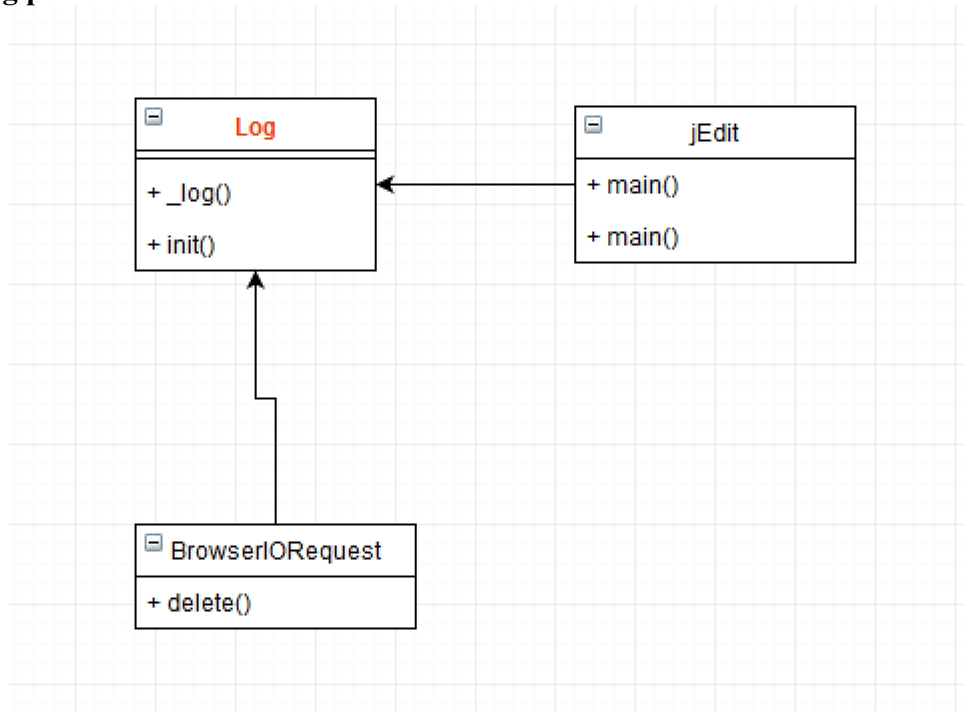


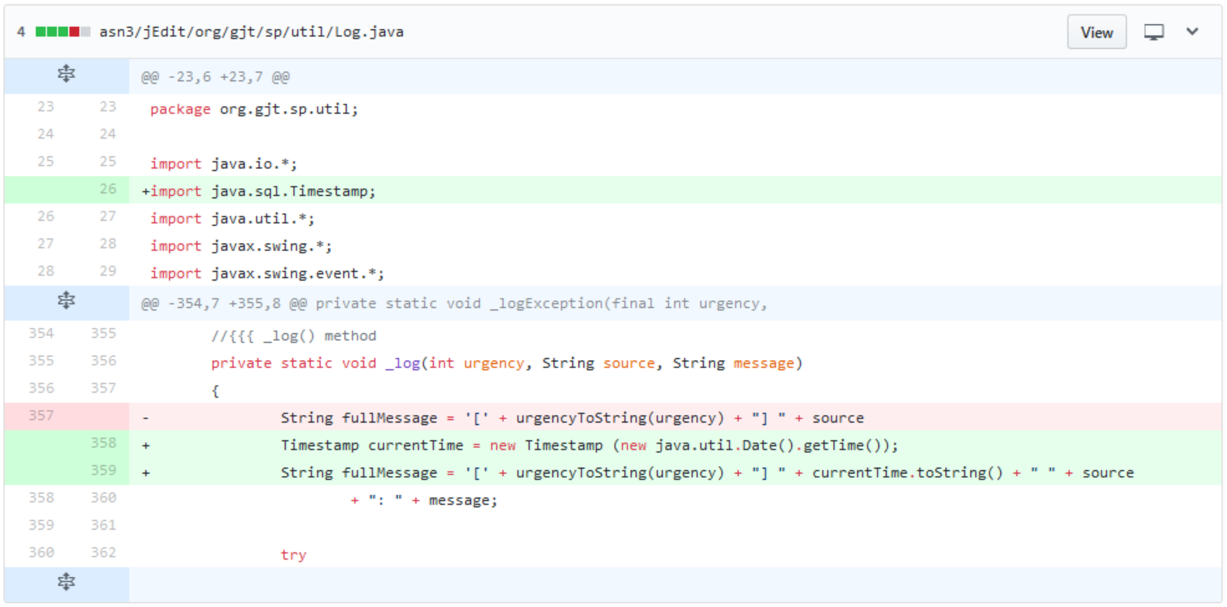
Figure 2 Example of the class diagram

5. Description of the implementation:

In the private _log method, each time it's called, it now creates a Timestamp and adds it to the fullMessage that is printed out when logging happens.

6. Sources: None

7. Highlighted Source Code:



```
@@ -23,6 +23,7 @@
 package org.gjt.sp.util;

 import java.io.*;
+import java.sql.Timestamp;
 import java.util.*;
 import javax.swing.*;
 import javax.swing.event.*;
@@ -354,7 +355,8 @@ private static void _logException(final int urgency,
        //{{{ _log() method
        private static void _log(int urgency, String source, String message)
        {
-               String fullMessage = '[' + urgencyToString(urgency) + "] " + source
+               Timestamp currentTime = new Timestamp (new java.util.Date().getTime());
+               String fullMessage = '[' + urgencyToString(urgency) + "] " + currentTime.toString() + " " + source
                        + ": " + message;

                try
```

Names and NSIDs: Corey Hickson (crh208), Benj Hingston (bvh895), Evan Salter (evs162)
Project: jEdit
Change Number: *5*
Date: *03/30/2018*

Duplicate data when creating a new view

1. **Change Request*:***
Currently clicking View | New View will create a new view for the same data; which means that modification in one view will affect the other one. Add menu item New View&Buffer under menu View to allow the user to duplicate data for the current shown view only.

2. **Concept Location:**
For this, I began by searching for the menu item "New Plain View" string. This took me to jedit_gui.props where it was kept, and I added in a spot under the menu for a New View and Buffer. I found a new-plain-view as well, and searching for that led me to actions.xml. In here I realized that I would need a new action for the new-view-and-buffer.
After this, I had to figure out how the actions were used, so I looked at others and this took me into jEdit.java. In here I saw some of the example menu options and added in one for the newViewAndBuffer. Once here, I used existing functions to create the new function.

Table 1. The list of all the classes visited during concept location.

#	File name	Tool used	Located?	Comments
1	Jedit_gui.props	Project search (New Plain View)	Changed	Contains menu structure
2	Actions.xml	Project search (new-plain-view)	Changed	Contains the actual actions which point to Java code, the menus use this
3	jEdit.java	Seen in actions.xml	Changed	Added in a newViewAndBuffer function

3. **Impact Analysis:**
There were two classes that were used a lot in this change so I went to them. However, since it was the changed feature using these classes, and not these classes using the changed feature, they won't be affected.

Table 2. The list of all the classes visited during impact analysis.

#	Class name	Tool used	Impacted?	Comments
1	Buffer.java	Following the class from jEdit.java	No	This was looked at to better understand how the buffers worked
2	View.java	Following the class from jEdit.java	No	This was looked at to understand how the views worked

4. **Learning process:**

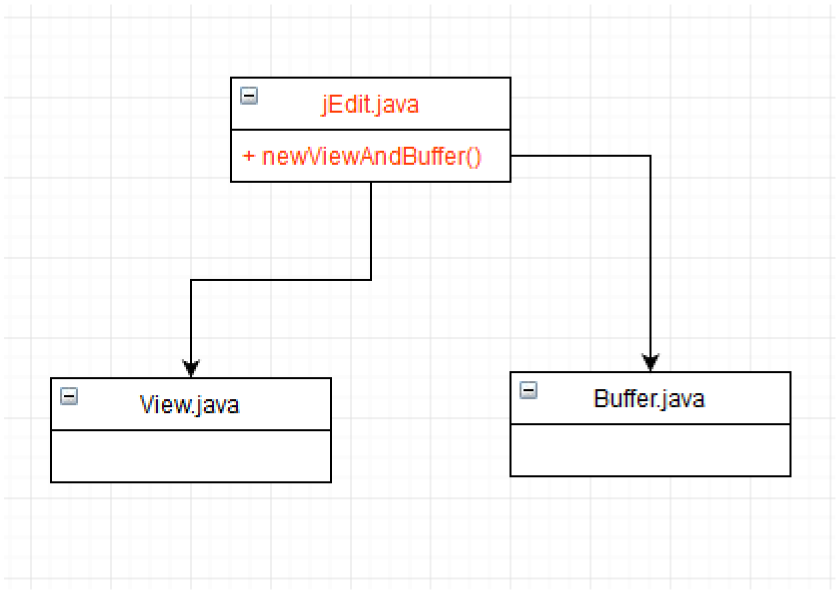


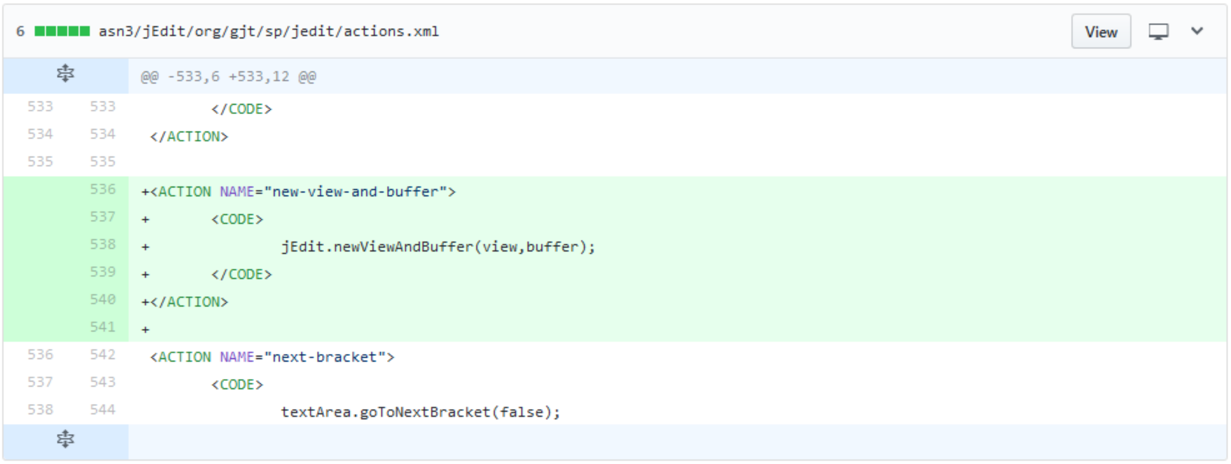
Figure 2 Example of the class diagram

5. **Description of the implementation:**

We added a new menu item into the jedit_gui.props and then gave it the newViewAndBuffer action in actions.xml. We created that action in jEdit.java. Inside that action, we make a new Segment, a new view, and a new buffer. We put the old buffer text into the segment, then put that segment into the new buffer. The result is we have a new view and a new buffer but with the same text as before.

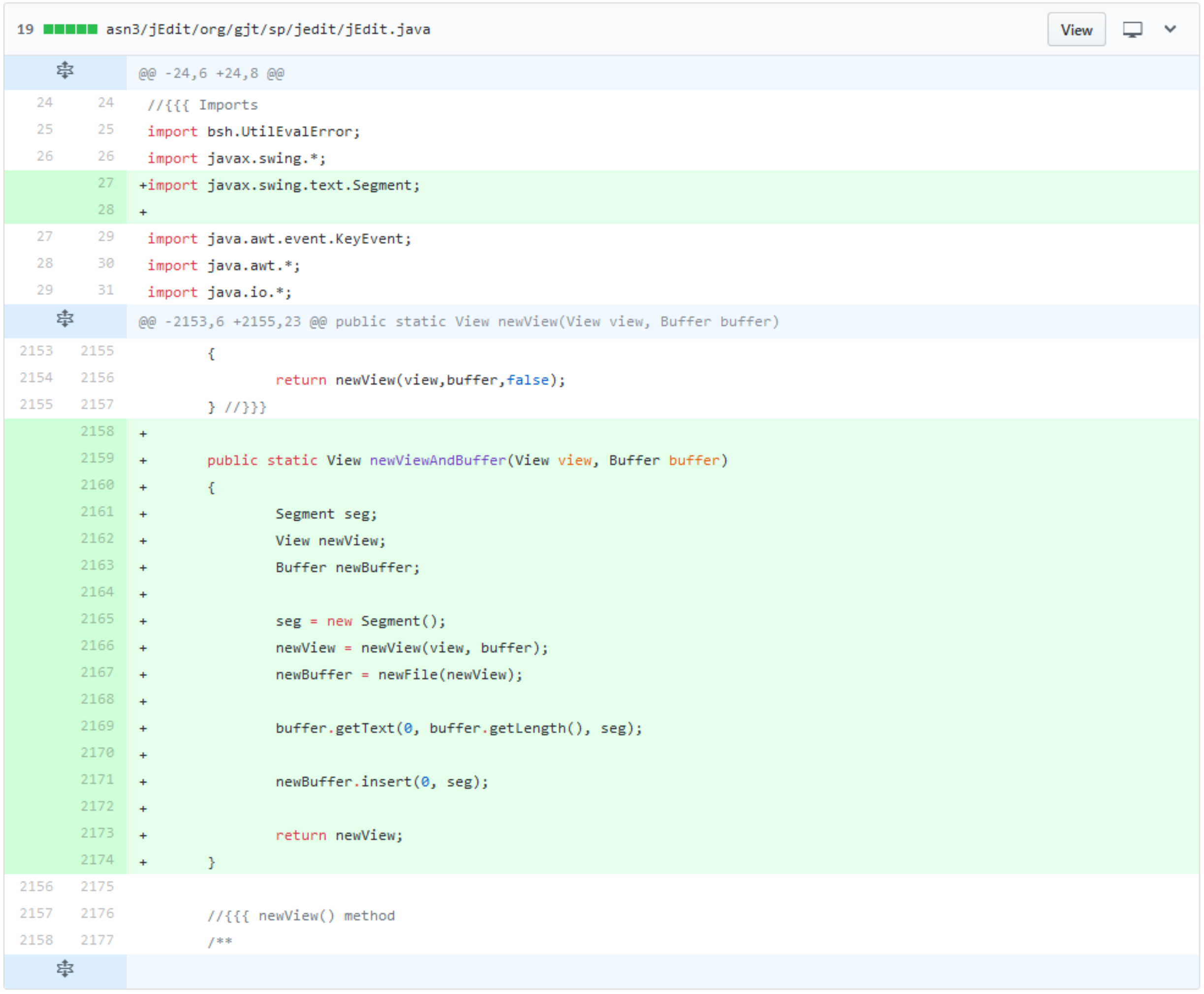
6. **Sources:** None

7. Highlighted Source Code:



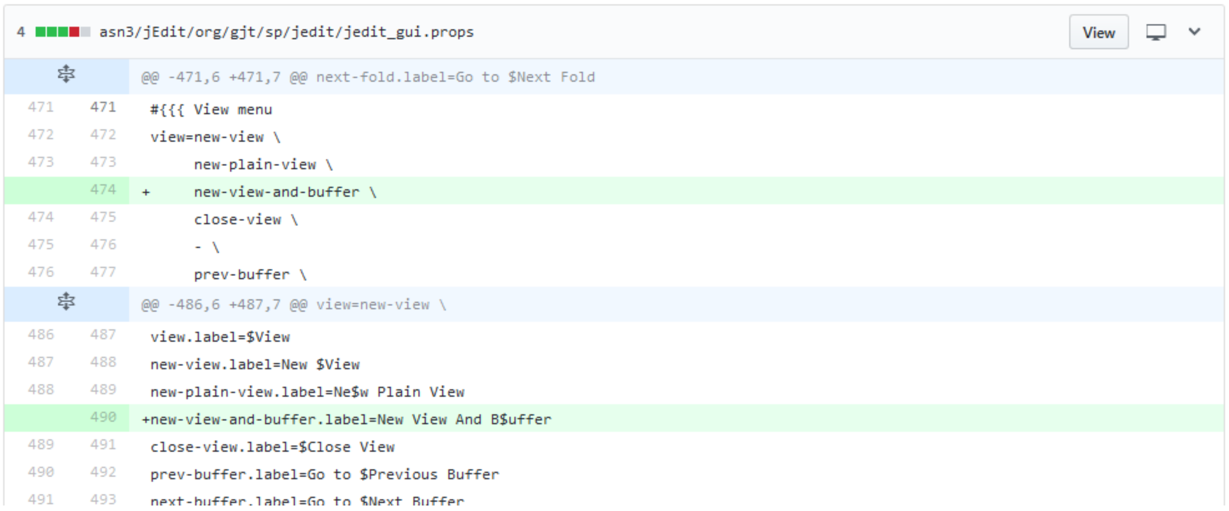
6 asn3/jEdit/org/gjt/sp/jedit/actions.xml

```diff
@@ -533,6 +533,12 @@
 	</CODE>
 </ACTION>
 
+<ACTION NAME="new-view-and-buffer">
+	<CODE>
+		jEdit.newViewAndBuffer(view,buffer);
+	</CODE>
+</ACTION>
+
 <ACTION NAME="next-bracket">
 	<CODE>
 		textArea.goToNextBracket(false);
```



19 asn3/jEdit/org/gjt/sp/jedit/jEdit.java

```diff
@@ -24,6 +24,8 @@
 //{{{ Imports
 import bsh.UtilEvalError;
 import javax.swing.*;
+import javax.swing.text.Segment;
+
 import java.awt.event.KeyEvent;
 import java.awt.*;
 import java.io.*;
@@ -2153,6 +2155,23 @@ public static View newView(View view, Buffer buffer)
 	{
 		return newView(view,buffer,false);
 	} //}}}
+
+	public static View newViewAndBuffer(View view, Buffer buffer)
+	{
+		Segment seg;
+		View newView;
+		Buffer newBuffer;
+
+		seg = new Segment();
+		newView = newView(view, buffer);
+		newBuffer = newFile(newView);
+
+		buffer.getText(0, buffer.getLength(), seg);
+
+		newBuffer.insert(0, seg);
+
+		return newView;
+	}
 
 	//{{{ newView() method
 	/**
```



```
4 asn3/jEdit/org/gjt/sp/jedit/jedit_gui.props
```

```diff
@@ -471,6 +471,7 @@ next-fold.label=Go to $Next Fold
 #{{{ View menu
 view=new-view \
      new-plain-view \
+     new-view-and-buffer \
      close-view \
      - \
      prev-buffer \
@@ -486,6 +487,7 @@ view=new-view \
 view.label=$View
 new-view.label=New $View
 new-plain-view.label=Ne$w Plain View
+new-view-and-buffer.label=New View And B$uffer
 close-view.label=$Close View
 prev-buffer.label=Go to $Previous Buffer
 next-buffer.label=Go to $Next Buffer
```

Names and NSIDs: Corey Hickson (crh208) Benjamin Hingston (bvh895) Evan Salter (evs162)
Project: jEdit
Change Number: 6
Date: 03/31/2018

Show/Hide Whitespace

1. Change Request:

Currently jEdit shows a red dot at the end of every line. Newline is the only whitespace symbol that jEdit shows. Add menu item Show/Hide whitespace under menu View to allow the user to choose whether all whitespace symbols (newlines, blanks, and tabs) will be shown. At this stage you do not have to worry about editing of the text with whitespace showing.

2. Concept Location:

In the View menu item I saw "Line Numbers" so I searched for that and found it in *View.java*.

Table 1. The list of all the classes visited during concept location.

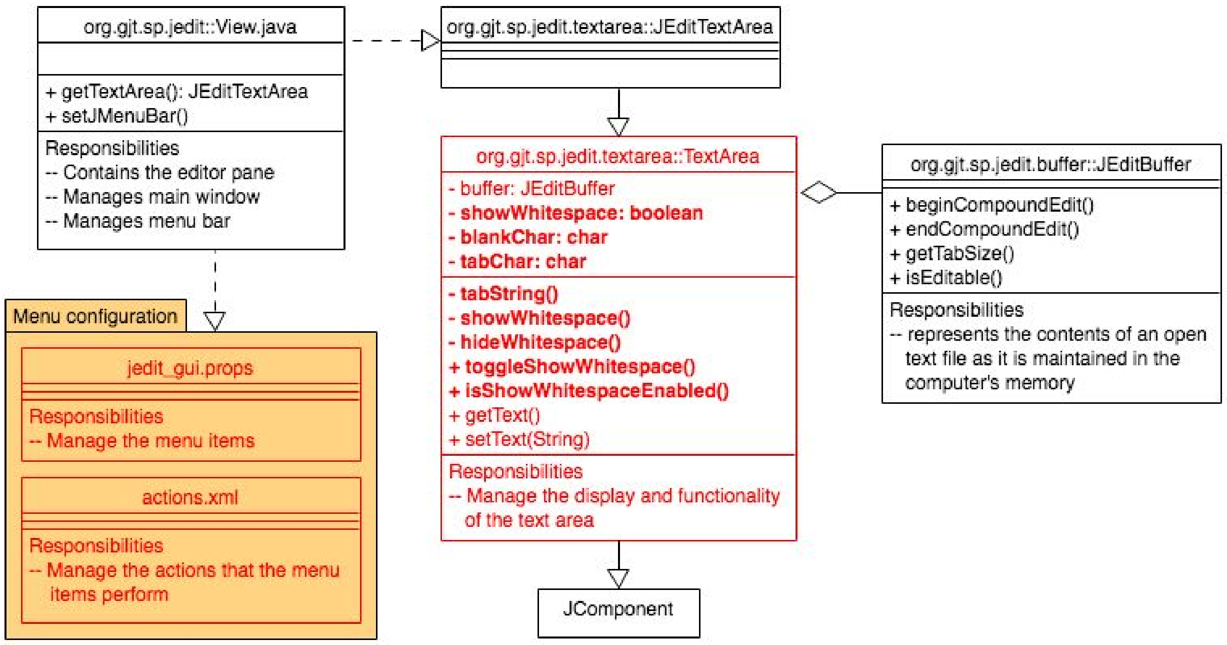
#	File name	Tool used	Located?	Comments
1	View.java	Find in Path	Unchanged	This is related to managing a "view" (i.e. window). It builds the menu bar but doesn't contain which menu items to add.
2	basic.xml	Find in Path	Unchanged	Used only for documentation only.
3	jedit.props	Find in Path	Unchanged	visual global settings.
4	jedit_gui.props	Find in Path	Changed	Defines the toolbar menu.
5	actions.xml	Find in Path	Changed	Binds menu items to executions.
6	TextArea.java	Find Class	Changed	Defines how the text editor should display text.
7	TextAreaPainter.java	Go To Definition	Unchanged	Displays text of current buffer.
8	JEditBuffer.java	Go To Definition	Propagating	Represents contents of open file.

3. Impact Analysis:

Table 2. The list of all the classes visited during impact analysis.

#	Class name	Tool used	Impacted?	Comments
1	TextArea	Find in Path	Yes, changed	Defines how the text editor should display text.
2	JEditBuffer	Go to Definition	No, unchanged	Represents contents of open file.
3	View	Find Class	Yes, unchanged	This references the files in jedit_gui.props and actions.xml to build the menubar items.
4	org.gjt.sp.jedit.textarea.*	Find Usages on TextArea	No, unchanged	new public functions will only be used in jedit_gui.props and actions.xml

4. Learning process:


org.gjt.sp.jedit::View.java
+ getTextArea(): JEditTextArea
+ setJMenuBar()
Responsibilities
-- Contains the editor pane
-- Manages main window
-- Manages menu bar
org.gjt.sp.jedit.textarea::JEditTextArea
org.gjt.sp.jedit.textarea::TextArea
- buffer: JEditBuffer
- showWhitespace: boolean
- blankChar: char
- tabChar: char
- tabString()
- showWhitespace()
- hideWhitespace()
+ toggleShowWhitespace()
+ isShowWhitespaceEnabled()
+ getText()
+ setText(String)
Responsibilities
-- Manage the display and functionality of the text area
org.gjt.sp.jedit.buffer::JEditBuffer
+ beginCompoundEdit()
+ endCompoundEdit()
+ getTabSize()
+ isEditable()
Responsibilities
-- represents the contents of an open text file as it is maintained in the computer's memory
Menu configuration
jedit_gui.props
Responsibilities
-- Manage the menu items
actions.xml
Responsibilities
-- Manage the actions that the menu items perform
JComponent

5. Description of the implementation:

To implement the functionality of toggling visible whitespace in the text in the editor, I added the following *TextArea*:

- attribute **showWhitespace**: holds value to have spaces and tabs visible
- attribute **blankChar**: unique unicode character for spaces
- attribute **tabChar**: unique unicode character for tabs
- method **tabString()**: returns proper amount of tabChar dependant on tab size of buffer

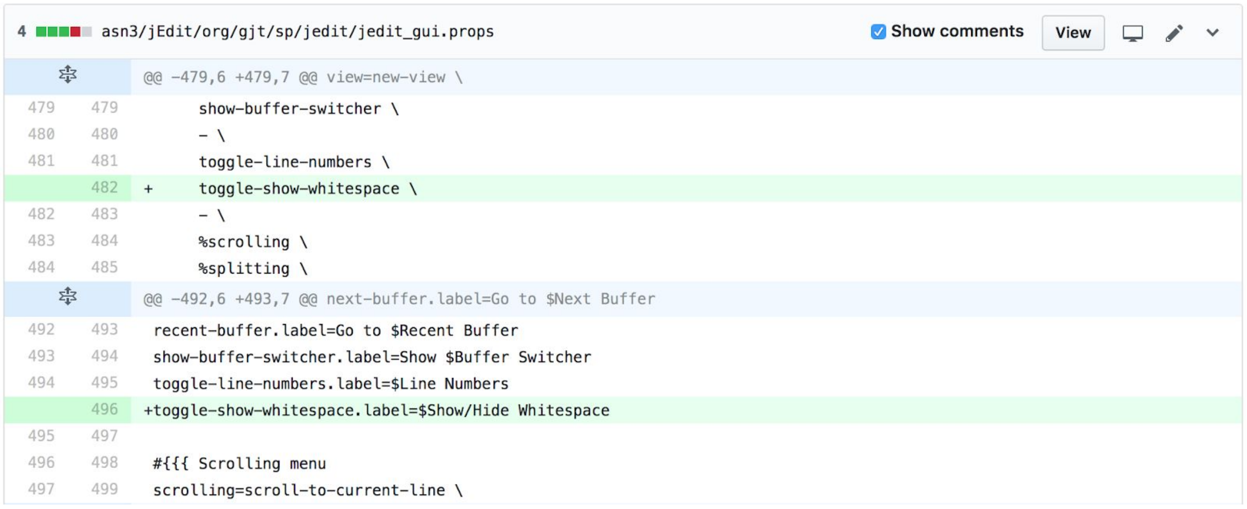
- method **showWhitespace()**: makes blanks and tabs visible in current buffer
- method **hideWhitespace()**: hides blanks and tabs in current buffer
- method **toggleShowWhitespace()**: control to show/hide whitespace
- method **isShowWhitespaceEnabled()**: checks if feature is enabled

To display new View menu option I added:
- **jedit_gui.props**: Create new menu identifier called "Show/Hide Whitespace"
- **actions.xml**: Set the actions for new menu item

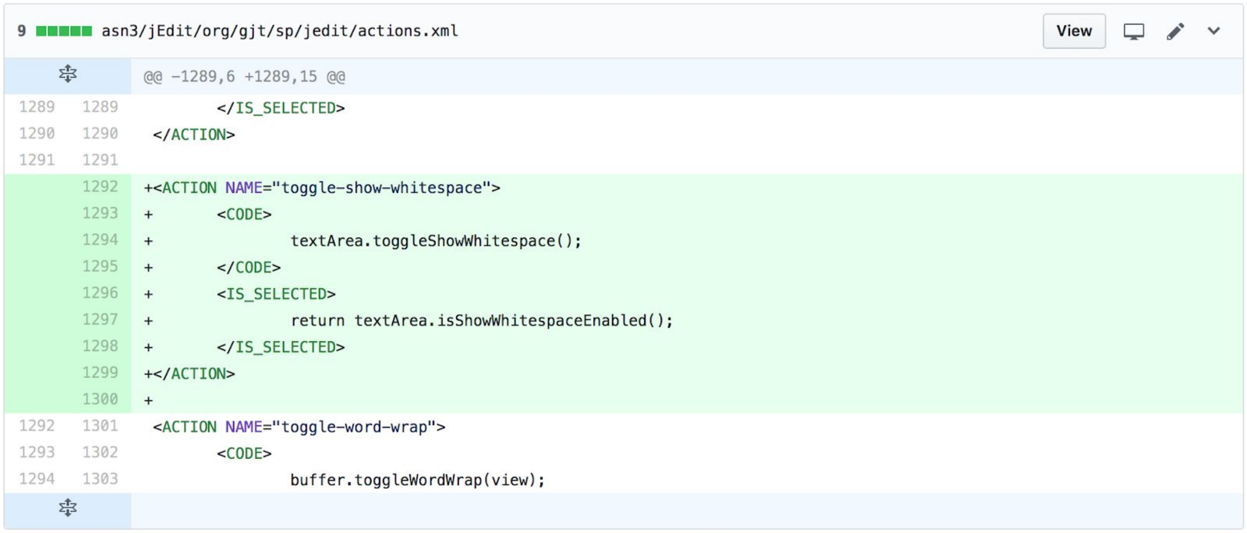
6. Sources:

https://gist.github.com/umidjons/10859940 helped me write *tabString( )*

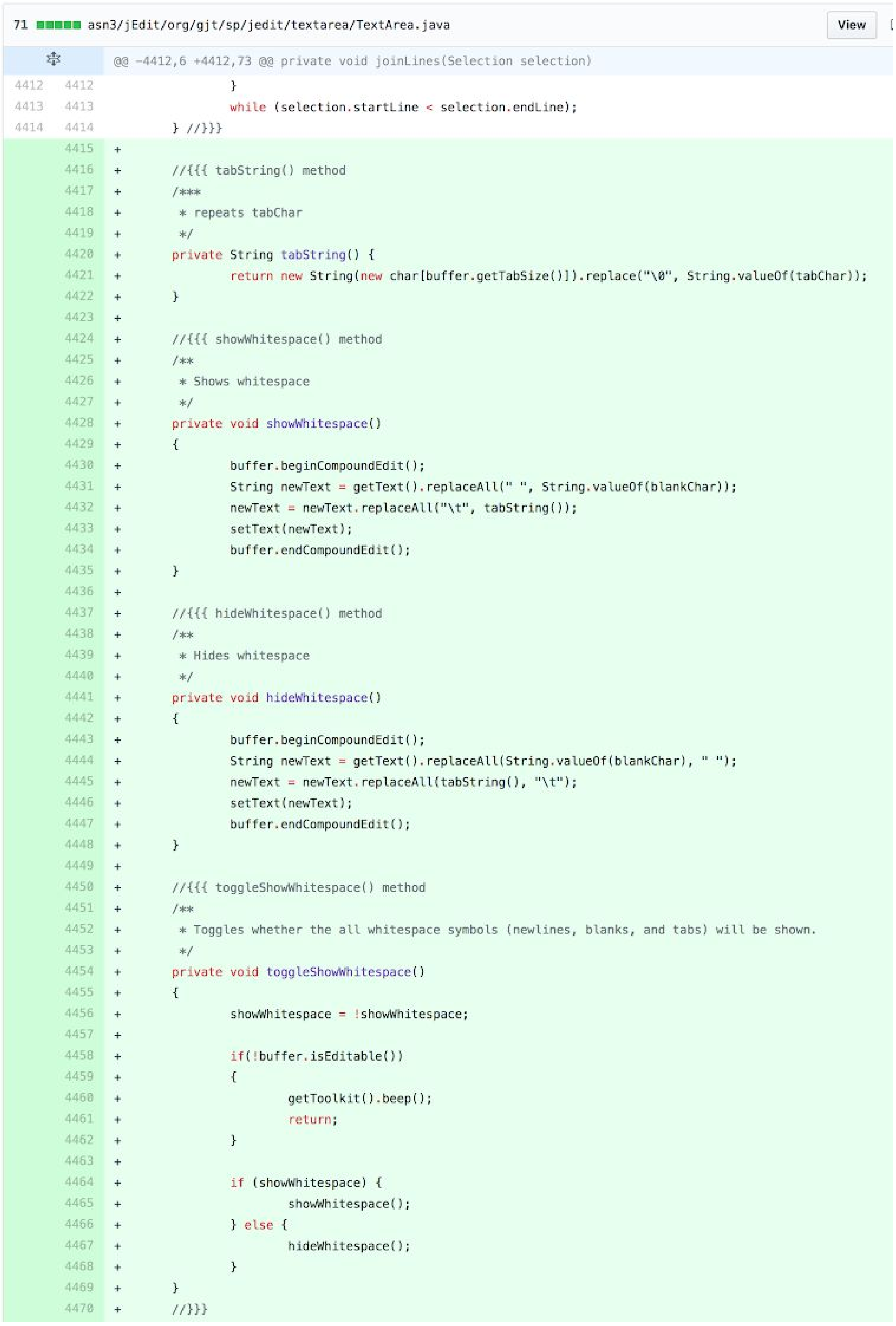
7. Highlighted Source Code:



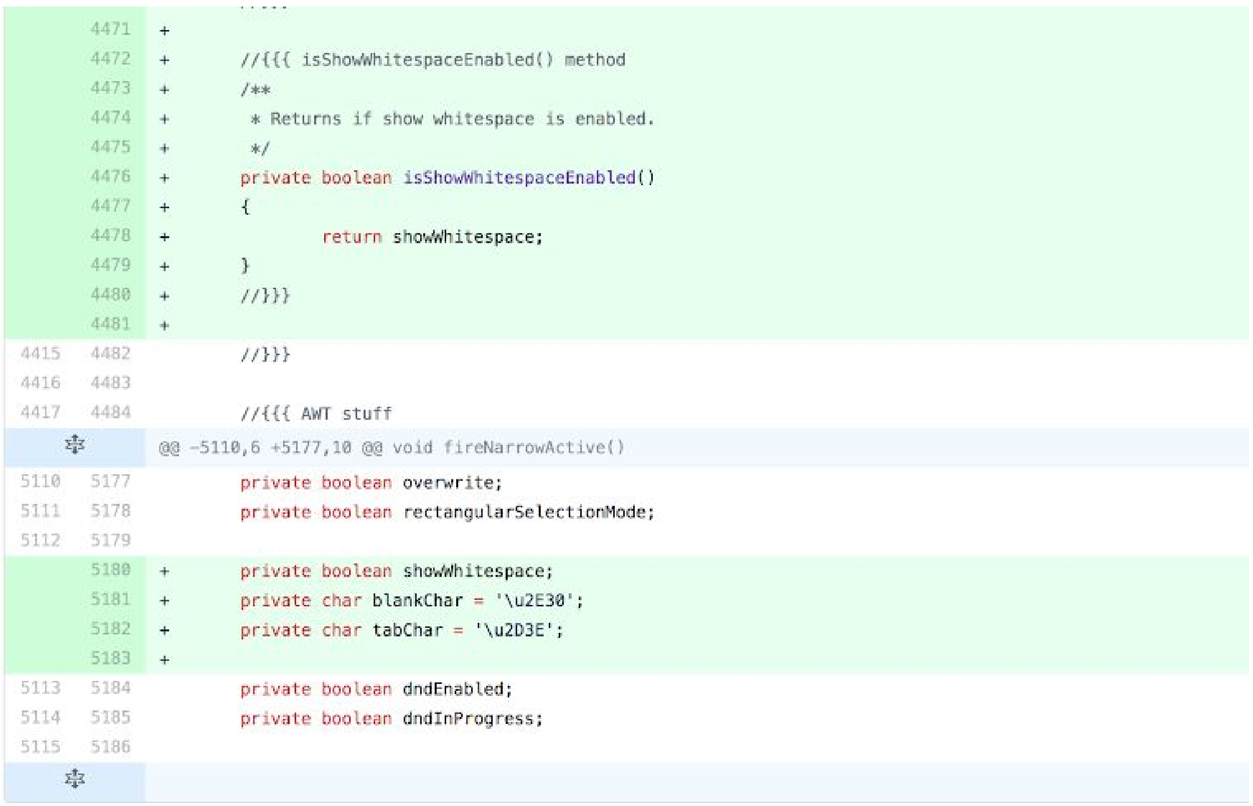
```
4 asn3/jEdit/org/gjt/sp/jedit/jedit_gui.props
@@ -479,6 +479,7 @@ view=new-view \
 	show-buffer-switcher \
 	- \
 	toggle-line-numbers \
+	toggle-show-whitespace \
 	- \
 	%scrolling \
 	%splitting \
@@ -492,6 +493,7 @@ next-buffer.label=Go to $Next Buffer
 recent-buffer.label=Go to $Recent Buffer
 show-buffer-switcher.label=Show $Buffer Switcher
 toggle-line-numbers.label=$Line Numbers
+toggle-show-whitespace.label=$Show/Hide Whitespace
 
 #{{{ Scrolling menu
 scrolling=scroll-to-current-line \
```



```
9 asn3/jEdit/org/gjt/sp/jedit/actions.xml
@@ -1289,6 +1289,15 @@
 		</IS_SELECTED>
 </ACTION>
 
+<ACTION NAME="toggle-show-whitespace">
+	<CODE>
+		textArea.toggleShowWhitespace();
+	</CODE>
+	<IS_SELECTED>
+		return textArea.isShowWhitespaceEnabled();
+	</IS_SELECTED>
+</ACTION>
+
 <ACTION NAME="toggle-word-wrap">
 	<CODE>
 		buffer.toggleWordWrap(view);
```



```
71 ■■■■■ asn3/jEdit/org/gjt/sp/jedit/textarea/TextArea.java

@@ -4412,6 +4412,73 @@ private void joinLines(Selection selection)
                        }
                        while (selection.startLine < selection.endLine);
                } //}}}
+
+               //{{{ tabString() method
+               /***
+                * repeats tabChar
+                */
+               private String tabString() {
+                       return new String(new char[buffer.getTabSize()]).replace("\0", String.valueOf(tabChar));
+               }
+
+               //{{{ showWhitespace() method
+               /**
+                * Shows whitespace
+                */
+               private void showWhitespace()
+               {
+                       buffer.beginCompoundEdit();
+                       String newText = getText().replaceAll(" ", String.valueOf(blankChar));
+                       newText = newText.replaceAll("\t", tabString());
+                       setText(newText);
+                       buffer.endCompoundEdit();
+               }
+
+               //{{{ hideWhitespace() method
+               /**
+                * Hides whitespace
+                */
+               private void hideWhitespace()
+               {
+                       buffer.beginCompoundEdit();
+                       String newText = getText().replaceAll(String.valueOf(blankChar), " ");
+                       newText = newText.replaceAll(tabString(), "\t");
+                       setText(newText);
+                       buffer.endCompoundEdit();
+               }
+
+               //{{{ toggleShowWhitespace() method
+               /**
+                * Toggles whether the all whitespace symbols (newlines, blanks, and tabs) will be shown.
+                */
+               private void toggleShowWhitespace()
+               {
+                       showWhitespace = !showWhitespace;
+
+                       if(!buffer.isEditable())
+                       {
+                               getToolkit().beep();
+                               return;
+                       }
+
+                       if (showWhitespace) {
+                               showWhitespace();
+                       } else {
+                               hideWhitespace();
+                       }
+               }
+               //}}}
```



```
+
+       //{{{ isShowWhitespaceEnabled() method
+       /**
+        * Returns if show whitespace is enabled.
+        */
+       private boolean isShowWhitespaceEnabled()
+       {
+               return showWhitespace;
+       }
+       //}}}
+
        //}}}

        //{{{ AWT stuff
@@ -5110,6 +5177,10 @@ void fireNarrowActive()
        private boolean overwrite;
        private boolean rectangularSelectionMode;

+       private boolean showWhitespace;
+       private char blankChar = '\u2E30';
+       private char tabChar = '\u2D3E';
+
        private boolean dndEnabled;
        private boolean dndInProgress;

```

Names and NSIDs: Corey Hickson (crh208) Benjamin Hingston (bvh895) Evan Salter (evs162)
Project: jEdit
Change Number: *7*
Date: 04/02/2018

Search list

1. **Change Request*:***
   Add a listbox to the search dialog that shows the last 5 strings you have searched for. When an item is selected, it should fill the search box with that value, so you are able to search for that string using the "Find" button. A search history function already exists, but requires you to use the page up/page down buttons on your keyboard to scroll through.

2. **Concept Location:**

Table 1. The list of all the classes visited during concept location.

#	File name	Tool used	Located?	Comments
1	jedit_gui.props	Search for "Return value of a BeanShell snippet"	Changed	Contains the labels for GUI elements
2	SearchDialog.java	Search for "beanshell-replace-btn" (found in jedit_gui.props)	Changed	Responsible for building the search dialog box, and all of the components within it
3	HistoryTextArea.java	Go To Definition (Ctrl+Click in Eclipse)	Changed	A TextArea that keeps track of the history, allowing the user to scroll through entries using page up/page down
4	HistoryText.java	Go To Definition (Ctrl+Click in Eclipse)	Propagating	Manages the history functionality of the above class (HistoryTextArea)

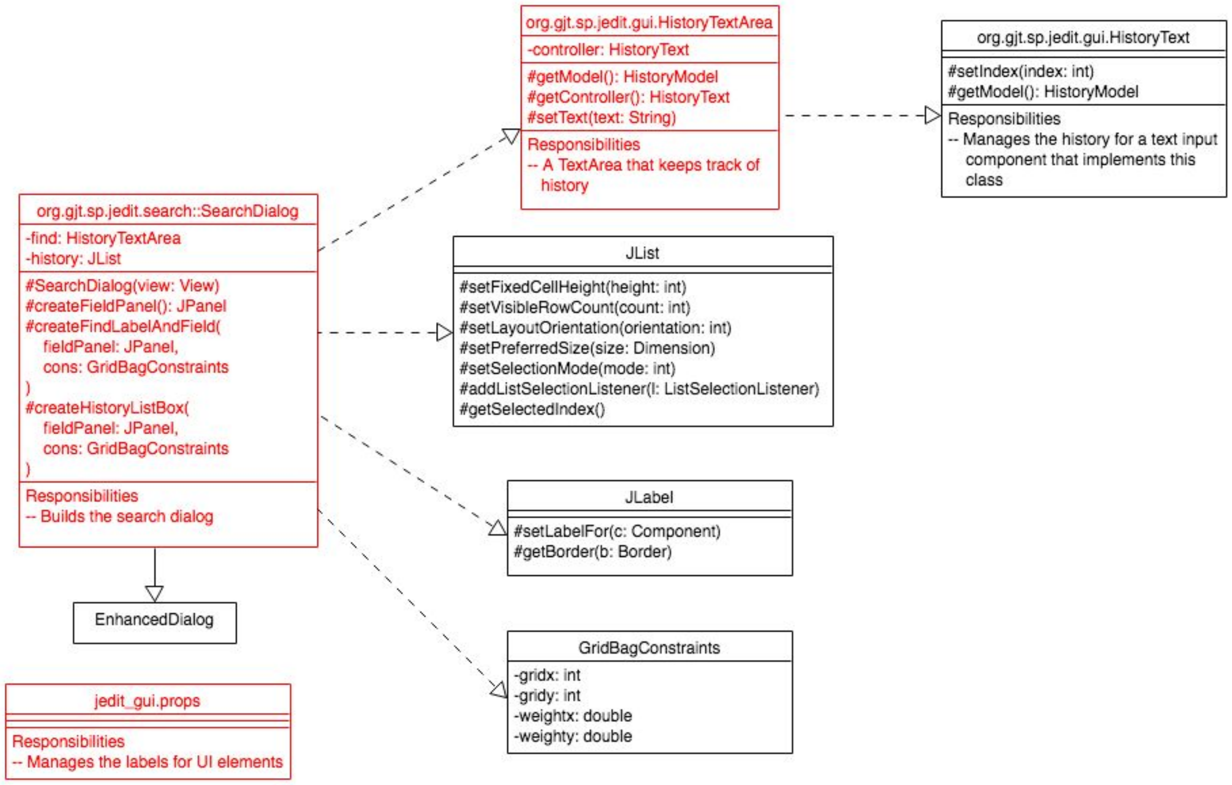
3. **Impact Analysis:**

Table 2. The list of all the classes visited during impact analysis.

#	Class name	Tool used	Impacted?	Comments
1	SearchDialog	Search for "beanshell-replace-btn" (found in jedit_gui.props)	Yes. I added a new GUI element to the window it controls	This class manages the UI for the Search dialog.
2	HistoryTextArea	Go To Definition (Ctrl+Click in Eclipse)	Yes. I added a new function that	Keeps track of the history for a text area that implements it.

			returns the controller (HistoryText instance). However, it is only called from SearchDialog, so no other uses will be impacted.	

4. Learning process:



org.gjt.sp.jedit.gui.HistoryTextArea
-controller: HistoryText
#getModel(): HistoryModel
#getController(): HistoryText
#setText(text: String)
Responsibilities
-- A TextArea that keeps track of history
org.gjt.sp.jedit.gui.HistoryText
#setIndex(index: int)
#getModel(): HistoryModel
Responsibilities
-- Manages the history for a text input component that implements this class
org.gjt.sp.jedit.search::SearchDialog
-find: HistoryTextArea
-history: JList
#SearchDialog(view: View)
#createFieldPanel(): JPanel
#createFindLabelAndField(fieldPanel: JPanel, cons: GridBagConstraints)
#createHistoryListBox(fieldPanel: JPanel, cons: GridBagConstraints)
Responsibilities
-- Builds the search dialog
JList
#setFixedCellHeight(height: int)
#setVisibleRowCount(count: int)
#setLayoutOrientation(orientation: int)
#setPreferredSize(size: Dimension)
#setSelectionMode(mode: int)
#addListSelectionListener(l: ListSelectionListener)
#getSelectedIndex()
JLabel
#setLabelFor(c: Component)
#getBorder(b: Border)
EnhancedDialog
GridBagConstraints
-gridx: int
-gridy: int
-weightx: double
-weighty: double
jedit_gui.props
Responsibilities
-- Manages the labels for UI elements

Figure 1 Example of the class diagram

5. Description of the implementation:

The first place I made changes was in jedit_gui.props. This is where the labels for different GUI elements are kept, so I needed to add a “History:” label for the new history element.

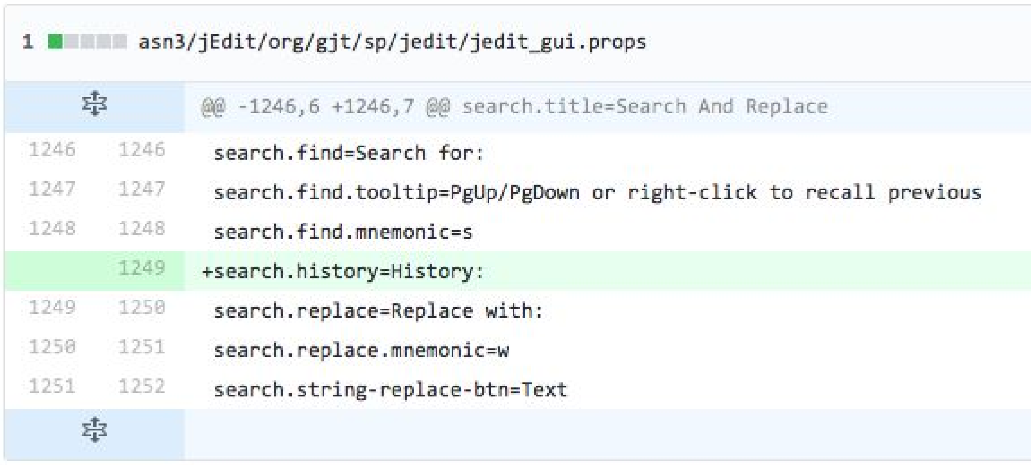
Next, I created the createHistoryListBox() method in SearchDialog.java. This creates all of the elements needed to display the history, including a JLabel and a JList. This function is called in createFieldPanel() so the history box actually appears.

This history of the search box is handled by a private variable on the find (HistoryTextArea) variable called controller (HistoryText). I needed to use some functions on this controller in order to allow selecting items from the history, so I had to add a getController() function to the HistoryTextArea class. I call this from SearchDialog, allowing me to maniputlate the controller.

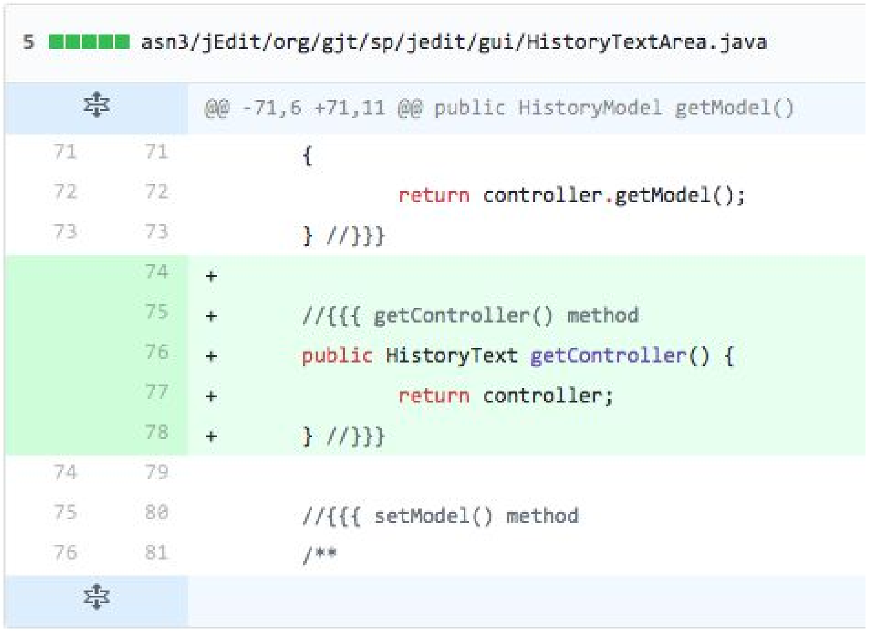
6. Sources:

- Information on using JList: https://docs.oracle.com/javase/tutorial/uiswing/components/list.html

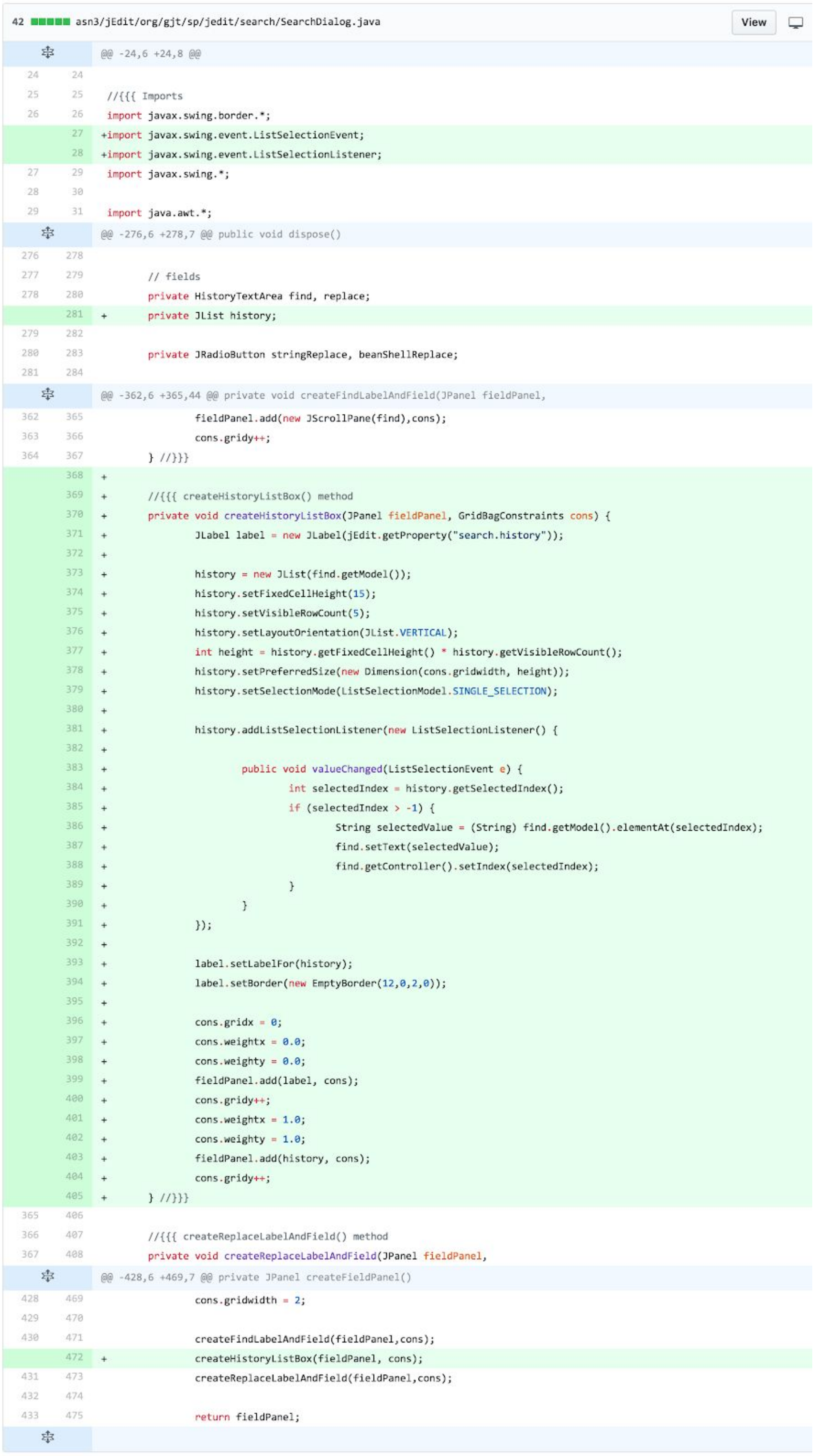
7. Highlighted Source Code:



```
1 asn3/jEdit/org/gjt/sp/jedit/jedit_gui.props

@@ -1246,6 +1246,7 @@ search.title=Search And Replace
 search.find=Search for:
 search.find.tooltip=PgUp/PgDown or right-click to recall previous
 search.find.mnemonic=s
+search.history=History:
 search.replace=Replace with:
 search.replace.mnemonic=w
 search.string-replace-btn=Text
```



```
5 asn3/jEdit/org/gjt/sp/jedit/gui/HistoryTextArea.java

@@ -71,6 +71,11 @@ public HistoryModel getModel()
        {
                return controller.getModel();
        } //}}}
+
+       //{{{ getController() method
+       public HistoryText getController() {
+               return controller;
+       } //}}}

        //{{{ setModel() method
        /**
```



```
42 ■■■■■ asn3/jEdit/org/gjt/sp/jedit/search/SearchDialog.java                View
@@ -24,6 +24,8 @@

 //{{{ Imports
 import javax.swing.border.*;
+import javax.swing.event.ListSelectionEvent;
+import javax.swing.event.ListSelectionListener;
 import javax.swing.*;

 import java.awt.*;
@@ -276,6 +278,7 @@ public void dispose()

        // fields
        private HistoryTextArea find, replace;
+       private JList history;

        private JRadioButton stringReplace, beanShellReplace;

@@ -362,6 +365,44 @@ private void createFindLabelAndField(JPanel fieldPanel,
                fieldPanel.add(new JScrollPane(find),cons);
                cons.gridy++;
        } //}}}
+
+       //{{{ createHistoryListBox() method
+       private void createHistoryListBox(JPanel fieldPanel, GridBagConstraints cons) {
+               JLabel label = new JLabel(jEdit.getProperty("search.history"));
+
+               history = new JList(find.getModel());
+               history.setFixedCellHeight(15);
+               history.setVisibleRowCount(5);
+               history.setLayoutOrientation(JList.VERTICAL);
+               int height = history.getFixedCellHeight() * history.getVisibleRowCount();
+               history.setPreferredSize(new Dimension(cons.gridwidth, height));
+               history.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
+
+               history.addListSelectionListener(new ListSelectionListener() {
+
+                       public void valueChanged(ListSelectionEvent e) {
+                               int selectedIndex = history.getSelectedIndex();
+                               if (selectedIndex > -1) {
+                                       String selectedValue = (String) find.getModel().elementAt(selectedIndex);
+                                       find.setText(selectedValue);
+                                       find.getController().setIndex(selectedIndex);
+                               }
+                       }
+               });
+
+               label.setLabelFor(history);
+               label.setBorder(new EmptyBorder(12,0,2,0));
+
+               cons.gridx = 0;
+               cons.weightx = 0.0;
+               cons.weighty = 0.0;
+               fieldPanel.add(label, cons);
+               cons.gridy++;
+               cons.weightx = 1.0;
+               cons.weighty = 1.0;
+               fieldPanel.add(history, cons);
+               cons.gridy++;
+       } //}}}

        //{{{ createReplaceLabelAndField() method
        private void createReplaceLabelAndField(JPanel fieldPanel,
@@ -428,6 +469,7 @@ private JPanel createFieldPanel()
                cons.gridwidth = 2;

                createFindLabelAndField(fieldPanel,cons);
+               createHistoryListBox(fieldPanel, cons);
                createReplaceLabelAndField(fieldPanel,cons);

                return fieldPanel;
```