Change request log - #je-3

1 Team

Team Name: Swathy & Rachit

Rachit Dalal : Change request log tracking

Swathy Hari Prasad : Code inspection, Identification of Concept Location, Impact Analysis, Actualization, Validation and Reverse Engineering

2 Change Request

Change Request ID: #je-3

- In the HyperSearch feature, highlight all the occurrences of the search string provided based on, if Ignore Case is on or off.

Github for #je-3 :

https://github.com/SwathyAravind/cs515-001-s20-Swathy-Rachit-jedit/tree/jedit_%23je-3_Swathy-Rachit

3 Concept Location

Step #	Description	Rationale
1	jEdit application is run via Eclipse to see it's working and understand the functionality of different options especially where the new feature addition is required.	To get a better understanding of the related code and its functionalities, to identify the concept location of this change request.
2	The Eclipse Search feature (Ctrl + Shift + R or Open Resource) is used to fetch at least one class where HyperSearch is used or referred by entering *hypersearch* in the search field. We concentrated on the below classes: ➢ HyperSearchResult.java ➢ HyperSearchResults.java	To list the classes where HyperSearch is referred to and we chose these classes where search results are saved.
3	From the listed classes, HyperSearchResult*.java* and HyperSearchResults*.java* which were more related to the feature that we want to add and are inspected further.	To find out where the initial changes are to be made for this change request.
4	By inspecting the classes more, we found that HyperSearchResults.java is where the search string is stored in tree node and tree traversal is also done here. Along with this highlighting the first node is also done in the inner class HighlightingTree.java This is where we identified the concept location.	We analyzed the different pattern matching techniques used here. Also we found out that results are stored in this class along with highlighting. Further analyzing, we found that all the results are stored correctly, but highlighting is done only on the first word of each line. Thus we got the exact concept location where we need the modification.

Time spent (in minutes): 150

4 Impact Analysis

Step #	Description	Rationale
1	HyperSearchResults.java is the first class that we have decided to modify and now we are checking the change propagation. So from StaHyperSearchResultstusBar.java the following classes are referenced. ➢ DefaultMutableTreeNode.java ➢ DefaultTreeModel.java ➢ HighlightingTree.java (inner class) ➢ HyperSearchTreeNodeCallback.java	By carefully going through the code, we found out that the given classes are referenced. After analyzing the initially found classes we will reduce the number of impacted classes.
2	Of the above found classes, most of them java.swing.tree classes referenced to store the data in a tree node like DefaultMutableTreeNode.java and DefaultTreeModel.java. They are not impacted.	They are java.swing.tree classes and they are not impacted.
3	HighlightingTree.java, the inner class of HyperSearchResults.java is where we need the changes. Since this class is not used anywhere, there are no other changes or impacts.	Since HighlightingTree.java class is not used anywhere, there are no other changes or impacts.
4		

Time spent (in minutes): 30

5 Prefactoring (optional)- Not Done

Step #	Description	Rationale
1		

Time spent (in minutes): 0

6 Actualization

Step #	Description	Rationale
1	We found that there is only a small change that is required. in searchDone() method in HyperSearchResults.java, all the results for the search string are stored. But in HighlightingTree.java, highlighting is done only for the first word and then once that is done, the matching word is made null and so the rest of the results are not highlighted. So we just need to comment out that null in the finally{} block.	in HighlightingTree.java, highlighting is done only for the first word and then once that is done, the matching word is made null and so the rest of the results are not highlighted.

Time spent (in minutes): 15

7 Postfactoring

Step #	Description	Rationale
1		

Time spent (in minutes): 0

8 Validation

Sample text used: Given at the bottom of this document

Step #	Description	Rationale
1	Open JEdit Application. Paste the text. ● Inputs : Select “it” from the text and perform hyper search with IgnoreCase ON in Search menu ● Expected Output : should show all the 89 occurrences highlighted. ● Actual Output : All 89 occurrences are highlighted. ● Test Result : Passed	Checks if all the matching occurrences are highlighted.
2	Open JEdit Application. Paste the text. ● Inputs : Select “It” from the text and perform hyper search with IgnoreCase OFF in Search menu ● Expected Output : should show only 2 occurrences highlighted. ● Actual Output : All 2 occurrences are highlighted. ● Test Result : Passed	Checks if all the matching occurrences are highlighted and is Case sensitive.
3	Open JEdit Application. Paste the text. ● Inputs : In the HyperSearch Bar, add “it” and search. ● Expected Output : should show all the 89 occurrences highlighted. ● Actual Output : All 89 occurrences are highlighted. ● Test Result : Passed	Checks if all the matching occurrences are highlighted while using HyperSearchBar.

Time spent (in minutes): 15

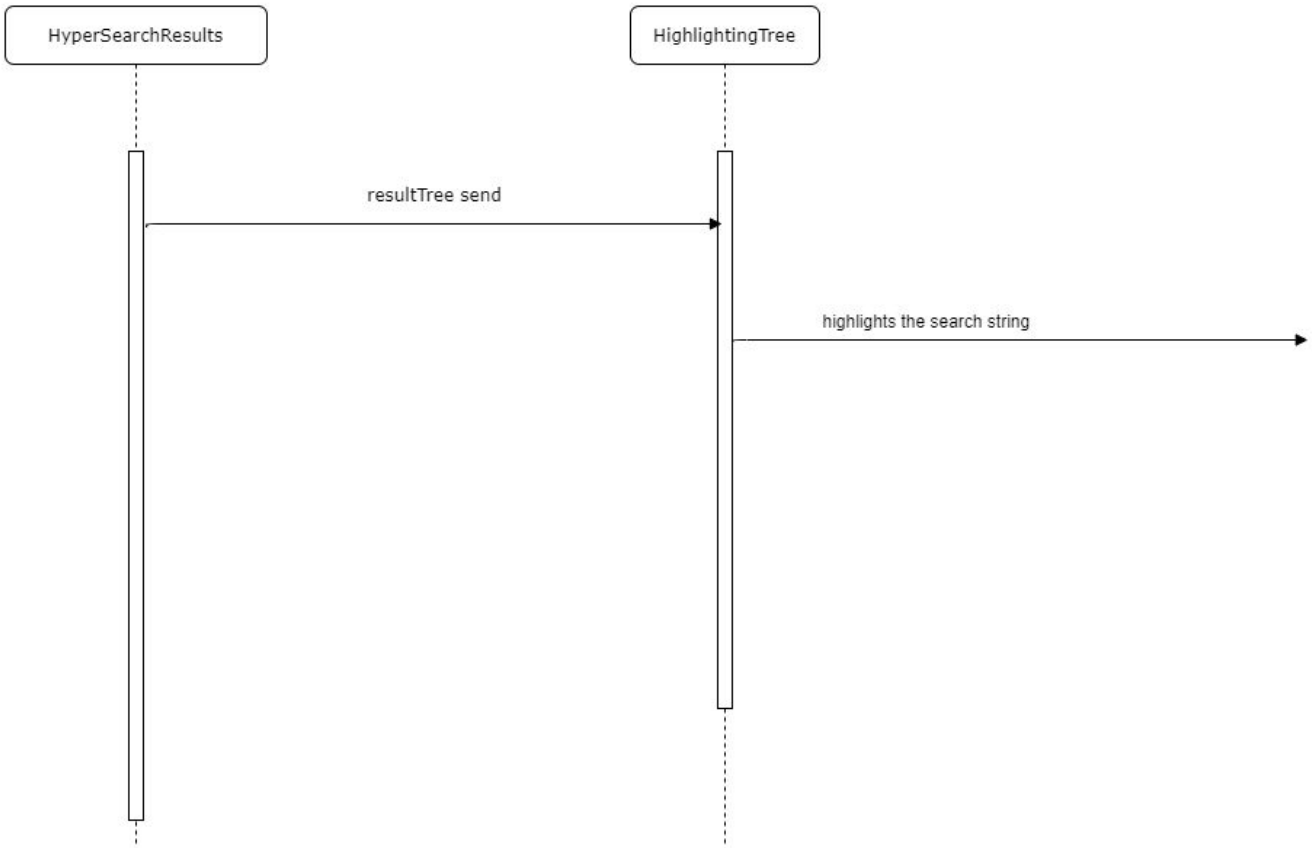
9 Timing

Summarize the time spent on each phase.

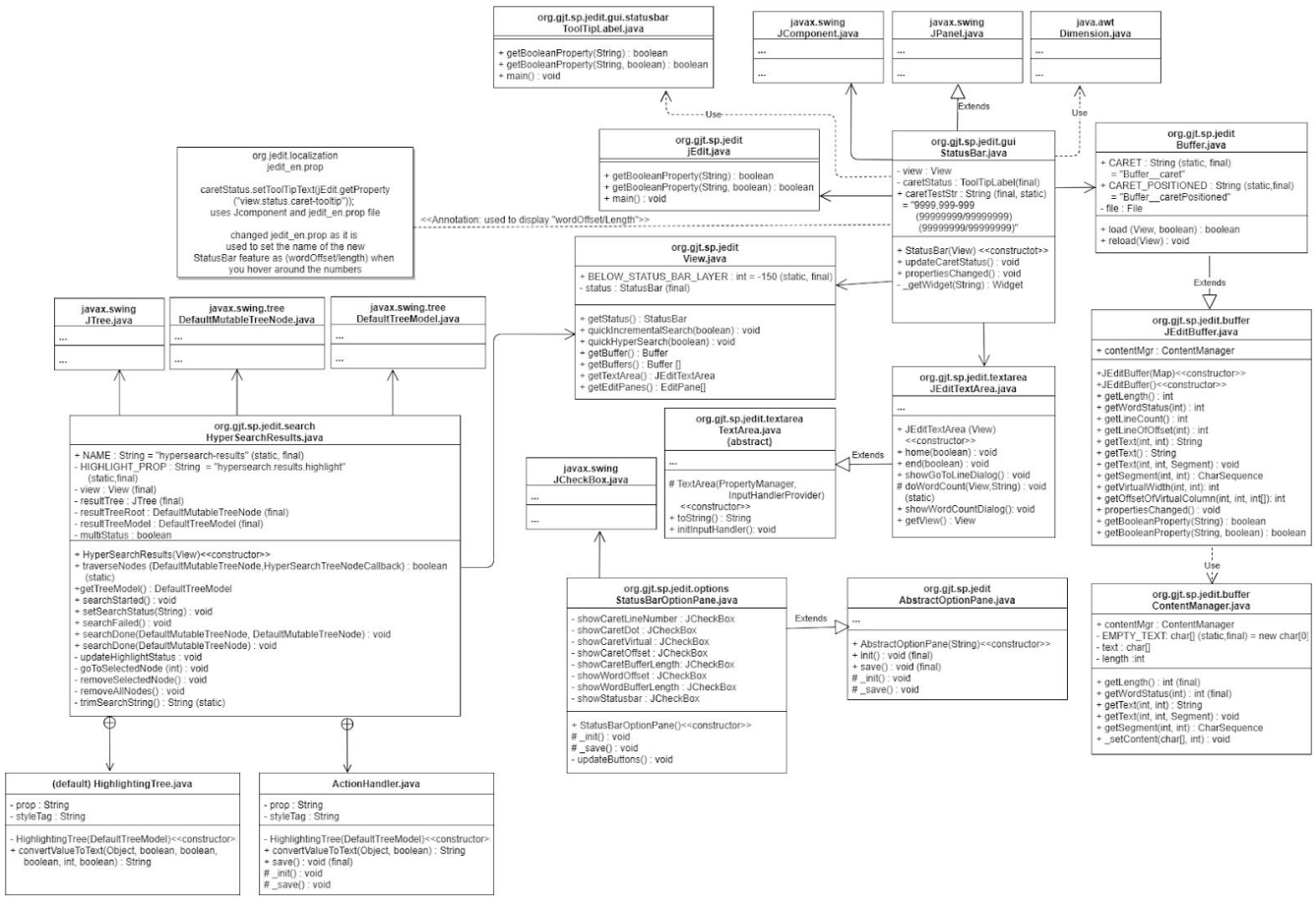
Phase Name	Time (in minutes)
Concept location	150
Impact Analysis	30
Prefactoring	
Actualization	15
Postfactoring	15
Verification	
Total	210

10 Reverse engineering

UML Sequence Diagram



UML Class Diagram



11 Conclusions

Finding Concept location for this feature was comparatively difficult. Although we found the class where changes are to be made, finding the exact concept location was difficult. But once it was found, Impact Analysis and Actualization was very easy. Validation was also easier.

The following classes, methods and attributes where modified:

- org.gjt.sp.jedit.search.HyperSearchResults.java
  - ➢ Inside this class, the inner class HighlightingTree was modified to comment out null in the finally {} block.

Sample Text used for Validation:

Features

jEdit includes syntax highlighting that provides native support for over 200 file formats. Support for additional formats can be added manually using XML files. It supports UTF-8 and many other encodings.

It has extensive code folding and text folding capabilities as well as text wrapping that takes indents into account.

The application is highly customizable and can be extended with macros written in BeanShell, Jython, JavaScript and some other scripting languages.

Plug-ins

There are over 150 available jEdit plug-ins for many different application areas.

Plug-ins are used to customize the application for individual use and can make it into an advanced XML/HTML editor, or an integrated development environment (IDE), with compiler, code completion,

context-sensitive help, debugging, visual differentiation and language-specific tools.

The plug-ins are downloaded via an integrated plug-in manager which finds and installs them along with any dependencies. The plugin manager will track new versions and can download associated updates automatically.[5]

Some available plug-ins include:

Spell checker using Aspell

Syntax and style checkers for various languages[6]

Text auto-complete

Accents plugin that converts character abbreviations for accented characters as they are typed.

XML plugin that is used for editing XML, HTML, JavaScript and CSS files. In the case of XML, the plug-in does validation. For XML, HTML and CSS, it uses auto-completion popups for elements, attributes and entities.[7]

Reception

In general jEdit has received positive reviews from developers.

Rob Griffiths wrote in April 2002 for MAC OS X HINTS saying he was "very impressed" and naming it "pick of the week". He cited its file memory upon reopening, its ability to notice if an open file was

changed on disk by another program, syntax coloring, including that users can create their own colour schemes, split windows feature, show line number feature, convertible tabs to soft-tabs and view sidebars.

He also praised its customization possibilities using the extensive preferences panel and the "on the fly" search engine, which searches while typing. Griffiths noted that the application has a few drawbacks,

such as that it is "a bit slow at scrolling a line at a time" and that because it is a Java application it doesn't have the full Aqua interface.[8]

Also reviewing the application in April 2002, Daniel Steinberg writing for O'Reilly Media said "The strength of jEdit for Java developers comes from the plug-ins contributed by the community...For

the most part, there's nothing here that couldn't be done with BBEdit or even with Emacs or vi. jEdit packages the capabilities much more nicely and makes it easy to call often-used functionality using the plug-ins.

Where I saw NetBeans as overkill, others may see jEdit as underkill for an IDE or overkill for a text editor. I find it Mac friendly and easy to use. I don't expect too much from it, so I tend to be pleased with what I get."[9]

Scott Beatty reviewing jEdit on SitePoint in 2005 particularly noted the application's folding feature along with its search and replace and PHP syntax highlighting capabilities.

He recommended the use of the PHPParser plug-in. PHPParser is a sidebar that checks for PHP syntax errors whenever a PHP code file is loaded or saved.

He noted that downloading jEdit is simple, but that getting and installing the plug-ins to customize it for individual use can be a complex process:

"Beware that a full setup requires a series of downloads, and that this process can take time."[10]

Writing in December 2011, reviewer Rares Aioanei praised jEdit's versatility, stating "jEdit's design allows you to use it as a simple editor, but also use it as an IDE and expand its functionality via plugins

so that it becomes exactly what you want it to be for the task or language at hand." but also adding that "jEdit is not, however, an IDE with everything but the Christmas tree, like Eclipse or Microsoft Visual Studio.

Rather, it's a compact application for editing code, providing practical tools along with basic IDE features."[11]

DESCRIPTION

Artificial intelligence (AI) has been fluidly defined, but, broadly, definitions have focused on either replicating human-like thought/behavior or creating rational thought/behavior. Historically, the goal of AI was to create a general intelligence (GI). when it became clear GI was out of reach, AI split into numerous subfields like

computer vision and natural language processing, to name a few. Within the last decade, each of these subfields has achieved breakthrough results, facilitating AI applications like autonomous vehicles and home assistants. These successes have caused some to speculate the replication of GI in a computer is just around the corner. However, the reality of the situation is quite the opposite: these applications, while groundbreaking, are not indicative progress toward the original goals of intelligent thought or action. In this course, we will discuss the limits and potentials of various approaches to modern AI applications, as well as opportunities to overcome these limitations. In this vein, this course will be a tour of both classic ideas and state-of-the-art methods in AI.

LEARNING OUTCOMES

Upon successful completion of the course, students will be able to:

Understand the philosophical underpinnings of the field.

Discuss which problems that are solvable with today's AI algorithms and others that require novel solutions.

Deploy general search algorithms that can be applied to a wide variety of tasks.

Formulate decision making processes that can be used for planning and classification purposes.

Build intelligent agents that perform simple tasks in an autonomous fashion.

Learn task-specific models from large collections of labeled training data samples using algorithms that are optimized using numeric solvers.

Analyze the successes and limitations of an AI system, with an emphasis on real-world applications and ethics.