Eric Wahlquist
Puja Ramanathan
CSE 403
Winter 2019

Teachure

Our Motivation

The goal of our proposed project, Teachure, is to teach users about the features of the Eclipse IDE that they may not be aware of. Many people prefer the simplicity and low memory usage of lightweight text and source code editors over IDEs. During our project brainstorming sessions, a complaint that was raised in the IDE group was that Eclipse is “slow”. However, it was also noted that when programming in Java, many of those same people prefer to use Eclipse for its features (like its static evaluation tools). Our project hopes to address some of the negativity toward Eclipse. Through real-time feedback to the user, our plugin will detect when users are not taking advantage of the many features it includes and inform them of how they could.

The closest existing plugin to what we would like to accomplish is called MouseFeed. MouseFeed works by generating an annoying popup notification any time the user clicks a button in the toolbar or a menu item, reminding them of the hotkey shortcut for that feature. This works great if a user already knows that a feature exists, but falls short as a full solution as it requires the user to be aware that a feature exists in the first place. The key difference between existing approaches (i.e. MouseFeed) and ours is that rather than teaching users keyboard shortcuts for features they already know about, our design will teach them about features they may not even know about.

Our solution would inform the user in a similar way, by presenting them with a small notification when the plugin detects that a feature is being skipped over. Imagine a user trying to rename a variable, manually changing it everywhere it is used in the code rather than simply using the Refactor->Rename feature. Another example might be a user typing in the “import” statement, not knowing that existing import issues can often be resolved automatically by pressing shift-cmd-o. Perhaps a user is manually commenting out a large block of lines, rather than simply selecting them and pressing cmd-/. In each of these examples, the user would be informed of how they can take advantage of the feature, teaching them how to save time and avoid hassle in the future.

Those who would benefit most from such a tool are of course the users who do not take advantage of many of the available features of Eclipse. In particular, this affects those who are new to Eclipse, but is not limited to that subset, as even experienced programmers and Eclipse users may not be aware of its features. This tool will help users to leverage those features to increase the speed at which they can write their Java code and hopefully counteract some of the perceptions of “slowness” of Eclipse.

Our Approach

In order to detect when the user has not taken advantage of a given feature, the first step will be to track the changes that are made in the document as the user types. This is shown to be done already in Eclipse, as the application provides real-time static analysis feedback. Our plugin would track changes in a similar fashion, except rather than tracking just the changes to the document, it would track the changes as well as the actual key input of the user, so that we can determine whether the changes were induced with a particular set of keystrokes. In doing so, a possible limitation might be the computation required to constantly track these changes, which could lead to performance issues.

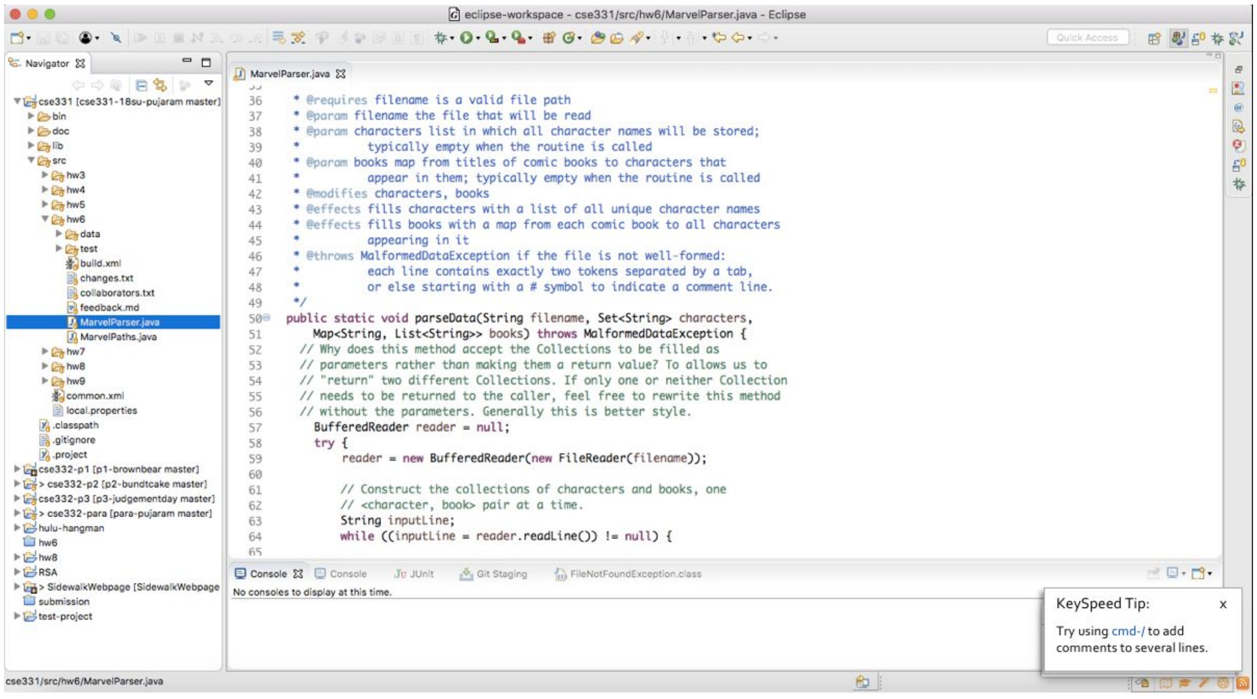
After tracking the document changes and user input, the next step would be to analyze the document changes and user input to determine whether a feature is being missed. As an example, we don't want to warn the user every time they type an inline comment ("//"), since this is normal behavior as code is typed. However, if we detect that the user has manually typed "//" at the beginning of several lines to comment them out, we do want to warn them about the ability to simply use "cmd-/" to do so instead. A potential limitation of this step may be that making these determinations will prove to be difficult or require trade-offs between user freedom and informing the user (e.g. how many inline comments do we allow users to type before we give them a warning?). It may also require a lot of computation to make these determinations, which could lead to performance issues.

Once the plugin has detected that the user is missing out on a feature, the final step is to display a notification to the user about the feature and how to use it. Potential limitations of this step include making tradeoffs between the benefits of providing less or more information to the user (how much do information do we want to give the user? Is a simple hotkey reminder enough, or should we direct them to a webpage with more information on a feature? Do we want to annoy the user to the point where they are fed up with the notifications and decide to use the features, or make our notifications less invasive?)

Challenges

Throughout the whole of the project, the biggest challenge we will face is that we are limited by the abilities of the Eclipse API as well as our lack of familiarity with it. Though our initial research into the API seems to indicate that functionality exists for key steps of our approach, we may run into issues with the intricacies of the API that could make certain functionality of our plugin difficult to implement. Further research of the API before designing a particular module of our project, as well as making use of the Eclipse Plugin Development forum for advice will help to mitigate this risk.

Example Notification


eclipse-workspace - cse331/src/hw6/MarvelParser.java - Eclipse
Navigator
MarvelParser.java
Console
Git Staging
FileNotFoundException.class
cse331/src/hw6/MarvelParser.java

```
 * @requires filename is a valid file path
 * @param filename the file that will be read
 * @param characters list in which all character names will be stored;
 *          typically empty when the routine is called
 * @param books map from titles of comic books to characters that
 *          appear in them; typically empty when the routine is called
 * @modifies characters, books
 * @effects fills characters with a list of all unique character names
 * @effects fills books with a map from each comic book to all characters
 *          appearing in it
 * @throws MalformedDataException if the file is not well-formed:
 *          each line contains exactly two tokens separated by a tab,
 *          or else starting with a # symbol to indicate a comment line.
 */
public static void parseData(String filename, Set<String> characters,
    Map<String, List<String>> books) throws MalformedDataException {
  // Why does this method accept the Collections to be filled as
  // parameters rather than making them a return value? To allows us to
  // "return" two different Collections. If only one or neither Collection
  // needs to be returned to the caller, feel free to rewrite this method
  // without the parameters. Generally this is better style.
    BufferedReader reader = null;
    try {
        reader = new BufferedReader(new FileReader(filename));

        // Construct the collections of characters and books, one
        // <character, book> pair at a time.
        String inputLine;
        while ((inputLine = reader.readLine()) != null) {
```

No consoles to display at this time.

KeySpeed Tip:

Try using cmd-/ to add comments to several lines.

Time Spent

The combined amount of hours that we spent on this proposal is around 18-20. This includes coming up with an idea, researching existing solutions, considering aspects of our proposal, creating the document and slides, and preparing our presentation.