Annotating Java Programs to Provide Feedback to CS1 Student

Author: Krish Godiawala
Advisor: Dr. Carlos Rivero

Department Computer Science

May 11, 2017
Abstract

When students start with CS-101 classes there are certain mistakes they are likely to make as they begin to program for example, forgetting to increment for loops, forgetting semicolons, missing out on closing braces to name a few. With most graded assignments all students have is textual feedback as to what is wrong with their code, what a lot of students miss is actual working code because often they struggle to incorporate comments from the grader or professor. It can be helpful for graders and professors if they have at hand a set of know error’s to “annotate” the program with and also provide actual code which can replace the students incorrect code. We provide a tool where graders can not only easily provide feedback to assignments but also provide substitute code, using which the application could generate a new program with the substituted code that students can use as reference.
Acknowledgements

I would like to thank Dr. Carlos Rivero for guiding me through my capstone project and always being available to help. I would also like to thank my colloquium instructor Dr Joe Geigel for guiding and preparing me for my presentations and submissions for my capstone. Finally I would like to thank the faculty members at RIT, family and friends for helping me through not only this project but my Masters Degree.
Contents

Abstract i
Acknowledgements ii
1 Introduction 1
2 Background 2
  2.1 Abstract Syntax Tree’s 2
3 System Overview 4
  3.1 System Components 4
  3.2 System Interaction 4
4 System Implementation 6
  4.1 Text Editor 6
  4.2 Load Programs 7
  4.3 Abstract Syntax Tree 7
  4.4 Feedback Provider 7
  4.5 Annotations 8
  4.6 Save Annotations 9
  4.7 Graded Feedback 10
5 Conclusions 11
6 Future Work 12
Bibliography 14
Chapter 1

Introduction

When a student submits code to be graded, the grader or the professor who is correcting it prints the code and underlines parts of the code which are incorrect and provides written feedback. However for most CS101 students the mistakes are common and repeating. It can be a fairly time consuming process to provide the same feedback to students. Also at times a new grader might even over look some mistakes if he/she had a set of errors to look up it would be beneficial to them.

Additionally when a grader provides feedback he only provides textual feedback stating the mistake made. Often students can be left wondering how to correct the mistake he/she has made because the feedback by the grader might only state the problem and not provide a solution to the problem. It is very helpful to students if they have the corrected code available for reference so they can analyze the mistakes they have made. Also at times it can be difficult to read hand written grader comments as they can be squashed, having typed comments is generally a lot more preferable.

Very often when assignments are printed and given back, they are often misplaced and when you want to go through them they cannot be found. So having a system which lets you store your graded assignment electronically is of the essence.
Chapter 2

Background

2.1 Abstract Syntax Tree’s

An Abstract Syntax Tree (AST) is a tree representation of source code with each AST node corresponding to a name, type, expression, statement or declaration\[9\]. The figure below is some sample source code, and its corresponding AST. Each node in the AST either represents a variable, operator or constant.

![Sample AST](image)

While there are many online tools available in the market like web cat\[14\] and stepik\[13\] that allow you to host assignments and automatically grade programming assignments, most of them primarily rely on having a bunch of test cases and running user submissions against those test cases. The feedback provided from them are most commonly the number of test cases that passed and if any failed they return either the incorrect value or the error generated by the programming language compiler, very so often they also give out errors such as timeouts. Looking at the errors it is very difficult for even for good programmers to identify their mistakes let alone CS1 students. These tools are furthest thing away from getting personalized feedback.

Another important tool that we analyzed was Codepourri\[7\]. The main idea behind this tool was that most programming language tutorials had static text and did
not provide a lot of information as and when each code statement executes. They de-
developed a tool that allowed anyone to create a tutorial with dynamic text that would
change as the state of the program changed. They too used the concept of AST’s
to highlight lines of code to provide feedback. They annotated each line of code
with comments and as the program executed the annotated text changed depending
upon the state of the program and AST. The idea for this project in many ways was
derived from the same tool with the main difference being that they targeted python
code.

For this project too I initially targeted creating the system for python, however
I just wanted to write basic AST creation and manipulation code in python and do
most of the other work in java. To do so I decided to use Jython which is a java imple-
mentation of python and provides simple integration of python code to java using
its libraries. However the problem with Jython was that its current release version
is 2.7 and we wanted to allow code written in python 3.0 and above. Also writing
JNI code to bridge java and python was beyond the scope of this project. Another
big problem was that the python AST library did not provide a lot of functions that
were needed to accomplish this project and writing all of them from scratch would
have ended up me not being able complete the project. Thus after carefully study-
ing the eclipse AST and plugin libraries we decided to go ahead to support java
programs. A big advantage of using eclipse’s AST library over python’s was that if
programs have syntactical mistake’s python’s AST library would throw exceptions,
however eclipse still parses the program with just not showing the AST subtree for
the method in which the error exists. And since my annotation approach is loosely
coupled from the AST’s I can still provide feedback to the students regarding their
errors.
Chapter 3

System Overview

To programmatically provide feedback for a submission it is useful to look at its AST. Instead of providing feedback for a program at a particular line we could select that AST’s node instead. There a few advantages of using AST’s, primarily if we look at the AST we can identify patterns in code which can be used to make our approach scalable and generic. Also another advantage to using AST’s is that we can check if the construct where the grader is providing feedback is a complete statement or not. A more detailed explanation of these advantages are given in following chapters.

So the idea of the project is to load and represent a students program as an AST and provide a user interface that allows a grader or professor to provide personalized feedback and substitute code to students. Also the user interface is responsible for showing feedback to students in a visually appealing manner and store the feedback on disk for future reference.

3.1 System Components

The system has 4 main components User Interface, Backend Handler, AST Handler and Data Store. The user interface is where the students program is loaded for grading, it is basically a text editor. Once the program is loaded it is passed to the backend handler to parse and store the code as well as associate changes and feedback with code. The AST handler is responsible for creating, manipulation and querying of AST, the backend handler communicates with the AST handler. The data store is responsible for storing the graded program and feedback.

3.2 System Interaction

The way a grader interacts with the system is shown in figure [3.2]. He starts off by loading the program which he wants to grade. He can then select the construct that he wants to provide feedback for and the system will check whether or not it is a valid construct. If it is not valid, the system will return a valid construct and the grader can then provide feedback. After he/she has finished providing feedback, they can save the result of the graded assignment with its feedback on disk as well as a new copy of the program with the suggested code changes.
Chapter 3. System Overview

**Figure 3.1: System Components**

**Figure 3.2: System Interaction**
Chapter 4

System Implementation

The code has been implemented completely in java with the help of jFace, SWT and Eclipse plugin libraries. SWT and jFace have been used for the graphical user interface and eclipse plugins have been used as a base for the editor as well as for AST generation and manipulation.

4.1 Text Editor

The text editor is the first and main user interface component which gets loaded when the system is run. The text editor is where the programs are loaded and shown to the user. The text editor internally uses class Document to hold the code. The document class is helpful as it provides information about the exact location of selected text by providing offset and length. I have created a class called MyDocument which extends the Document class and have added a few features to it such as save to file, attaching a name to the document and even provides the corrected selected construct. This allows easy modification to the document in an object oriented manner. The following image shows the design of the editor.

Figure 4.1: Editor
4.2 Load Programs

To load the file I have created a jFace Action component called Load File which opens a dialog box to navigate through a user’s file system to load a file. There are two ways to invoke this action, one way is to press ctrl+o to open the dialog box or by clicking on the “Files” menu on the top menu bar and navigating to “Open File”. The dialog box filters out all but java files.

4.3 Abstract Syntax Tree

Once the program has been loaded, code is parsed to generate an AST. The AST classes are responsible for generating, modifying and querying inside the Abstract Syntax Tree. I have written a simple interface for AST parse to allow extension of this system to other programming languages such as python. The IAST interface takes in the source code as an input to the parse method and returns an object which represents an AST.

```java
public interface IAST<
    E T> {
  public E parse(T args);
}
```

In the case of the eclipse AST library the parse method takes as input the source code as a char[] array and returns a compilation unit. When the grader selects the line of code he/she wants to annotate, the first thing that needs to be checked is whether or not the selected code is valid. Consider the code in figure 2.1 if the grader selects "a := a +" instead of "a := a + 1;" that is an error. So the system’s AST parser checks the selected code and returns "a := a + 1;". It does so by performing a Least Common Ancestor (LCA) search on all the AST nodes selected and returns the entire subtree selected from the parent of all the AST nodes selected. So in this case the + operator is the parent and since b too is its child it returns the entire statement "a := a + b." The way eclipse allows you to traverse around the AST is by using ASTVisitor interface which follow the visitor pattern. We can implement the preOrder2(ASTNode n) method in ASTVisitor to correctly traverse the AST.

Once all the feedback had been provided by the grader and when it is time to save the graded assignment, not only do I have to save the feedback but also create a new file (the working program) which contains the changes suggested by the grader. In my initial implementation I was modifying the Abstract Syntax Tree by using the ASTRewrite class while doing so I had written code to take care of a few cases for rewrite however the problem with this approach was that there are way too many types of ASTNode’s and far too many permutations to take care. That is why instead of modifying the ASTNode I decided to directly modify the document. The only disadvantage to this method is that we cannot directly know that if the feedback provided by the grader is syntactically correct or not (an assumption is made that the grader is right) however we can re-parse the newly created document into an AST to check for errors.

4.4 Feedback Provider

Once the program has been selected by the grader an AST is created by the back-end system the code is loaded into the text editor to grade. To provide feedback to a student at particular line of code, the grader needs to select that line of code and
press F7 or from the menu bar under Annotations select annotate. This will invoke a jFace action called Annotate and it will first verify whether the selected construct (using the LCA method described above) is valid or not and if not load a valid construct. In addition to this the system will also load all the previous feedback into a dialog box. Shown below is a sample screen shot of that process. The code snippet in the image is to reverse an array. However within the for loop the student has set the variable "j" to "a.length" instead of "a.length - 1" which is a very common mistake causing an ArrayIndexOutOfBoundsException. So when the grader wants to provide feedback, he/she instead of selecting "j = a.length" only selects "j =", the system ends up selecting the correct construct into the dialog box which is j = a.length. In addition it also loads all the feedback that has been previously stored. The dialog box is a jFace Composite which has a gridLayout of 3 rows and columns. The fist row have 3 labels, the second row has an empty label (this is done so that the combo boxes align with their labels above) and two combo boxes one for feedback and the other for code, and the last row contains 3 text boxes which store the code and feedback. I initially thought of having 3 combo boxes in the second row where in the first combo box the grader would get to choose between the original construct he/she had selected or the one that the system recommended, however I decided to do away with that because it looked confusing. To add the feedback the grader needs to click the add button at the end.

4.5 Annotations

Once all the feedback is provided one of the main goals of the project was to provide the student an appealing way to see the feedback. To do so we take advantage of Eclipses’ annotation framework. Eclipse provides a class called SourceViewer\[12\] that can store a document (defined in the previous sections) and can support highlighting to the document, annotating the document, and listen for changes in the document. Eclipse also defines an interface IAnnotationHover\[2\] which on mouse hover the line of code loads the annotation. However the eclipse javadocs do not clearly explain how to to get annotations to work. However I have found this discussion\[1\] on the eclipse forum and have built my solution using the same approach with Dr. Rivero’s permission. To allow annotations to work on my editor I first need to extend and override a few methods in the Annotation class for my custom annotations. My custom annotations store a line number (which line in the SourceViewer this annotation belongs to), a position (which holds the offset and length for the annotation within the SourceViewer) and the actual feedback which is both textual and
code (this is represented as a class which holds both together). The Annotation Interface does not provide some information about Annotations such as how to paint the annotations, or if the annotation spans multiple lines etc. So to get that information we need to implement the IAnnotationAccess interface. The IInformationControlCreator interface is responsible for holding the shell inside which the annotation should be painted. The CompositeRuler, OverviewRuler are the components within the SourceViewer where the Annotations are presented and the AnnotationModel is responsible for storing all the annotations within the SourceViewer. The following image shows how the annotations displays themselves.

To get persistent storage for the feedback provided by the grader the annotations need to be stored on disk. To do so we take advantage of java’s serialization interface. Although the Annotation class does not implement the serialization interface internally my subclass which defines custom Annotations takes care of this by implementing the serialization interface. This works in the case of the Annotations class because the annotation class’s field’s remain the same for all the annotations and I have overridden its methods to load the values at runtime and serialize my custom field’s. However as explained above each annotation also requires a position object which too does not define the serialization interface. To solve that problem I have created a subclass of the position object and implemented the writeObject and readObject methods. To save the feedback all that needs to be done is press ctrl+s or navigate to the save option on the menu bar this will open a new dialog box that will let you navigate through the file system to store the graded result in a folder. This will result in all the annotations being stored as well as creating a new file called created.java which contains the program with all of the changes provided by the grader. Also although currently all this information is currently being serialized and stored directly on disks, one can easily extend this to store the results in a NoSQL database like MongoDB.
4.7 Graded Feedback

The above sections discuss how the grader provides feedback and stores the feedback using the system, this section discusses how the student can see the feedback. To see the feedback provided by the grader a student can run the application which would open the text editor, next he must press ctrl+a or navigate to Annotated Assignments inside Files folder on the menu bar which would open a dialog box to load the folder that has all the feedback. What would be visible in the text editor would be the same as what the grader see’s when he/she have annotated the assignment as seen in Figure 4.3.
Chapter 5

Conclusions

The two main goals of this project was to create a system to help graders have a set of repeating errors and feedback available to annotate an assignment with and also provide students a user friendly and easy way to go through their feedback. With this new system developed we have been able to achieve the above objectives. Generally when a student gets feedback it is on a sheet of paper with their program on it with a bunch of handwritten grader comments which can be tough to read. With this system not only do we solve the problem of not understanding the graders comments but also obviate the need to print hundreds of assignments.

However this new system does have a few limitations. If a student has made multiple mistakes in the code and if one mistakes fall under one of the other mistakes while looking at their AST’s then we cannot give them separate feedback however we could give the feedback together in one annotation. Another limitation of this project is that annotations are stored on a folder and not on a database however to get over that we can use MongoDB to store the feedback and create a centralized repository.
Chapter 6

Future Work

The current system has not been tested in the classroom setting and if it undergoes trial we can see what feedback graders, professors and students have and try to incorporate them.

In the current system graders need to manually go through each submitted assignment and look for errors, however it would be extremely useful if the system could auto detect errors in an assignment based on previous annotations. One way to do this is explained in the paper A framework for source code search using program patterns[11] which describes a way to search for a pattern in code. What the paper describes is changing the original code into a generic format which could be searched by providing a pattern.

Consider that you want to search for the above piece of code in a project. What the paper describes is replacing the source code you want to search, with symbols as defined below.

```java
while ( # ) { @ }
```

The code and pattern above have been taken from [11]. Above the # sign represents a expression and @ sign represents statements. However when I tried to do the above by modifying the original code and perform subgraph matching the problem that I encountered was when I modified the source code into the above pattern and tried reading it into an AST it was throwing error’s because the new code does not follow Java Language Specifications (JLS). Thus the only way to do the above is using a non-deterministic finite automata as described in the paper.

Today a very popular programming language among CS1 classes is python. This is mainly due to its ease of use and also you could teach students all the important programming concepts while shielding them from the concepts of pointers and type safety. So an important extension to this project would be porting the code to annotate python programs. To do this two things primarily need to be handled. First you need to implement the IAST interface to create an AST. Secondly one needs to create a node traverser to be able to find the Lowest Common Ancestor (LCA) of a bunch of nodes and return the subtree from its ancestor.

Since most CS1 students submit their programs as a single file the current system only supports assignments submitted as a single file. However the system can support entire projects having multiple packages and dependencies. Eclipse defines an interface called IJavaProject[10] which represents the root of a java project and
lets you handle packages, multiple classes etc. Also the current user interface would need to be tweaked to support multiple files.

Finally an integration for this project can be provided into mycourses so students can have simple web based module to view their feedback. Mycourses currently does let you provide textual feedback to assignments however adding such support would make it a comprehensive way for submitting, grading, and viewing feedback.
Bibliography


