CLIP: An Introduction to Programming

A Senior Research Paper

Presented to the

Math & Science Department

Russell Math & Science Center

Alabama School of Fine Arts

In Partial Fulfillment

Of the Graduation Requirement

Of the Math & Science Program

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

By

Casey Barnette

May 1, 2013

# Acknowledgements

I would like to give thanks to my mentor, Dr. Gray, at the University of Alabama Computer Science Department for his ideas and guidance throughout the research process and for his help with my poster for science fair. I would also like to thank my faculty advisor, Mrs. Yarbrough for reminding me about deadlines. I would like to thank the Math and Science faculty that has taught me over my five years at ASFA. Also I would like to thank my parents for support both during the project and during my time at ASFA. Lastly I would like to thank the Math Science class of 2013 for being awesome and making through these years with me.

**Biographical Outline**

**Personal Data**

Name: Casey Barnette

Date of Birth: April 15, 1995

Place of Birth: Fort Myers, Florida, U.S.A.

City of Residence: Trussville, Alabama, U.S.A.

**Accomplishments**

12th Grade (2012-2013)

* North American Computational Linguistics Olympiad
* CARSEF 3rd in Math and Computer Science
  + Intel Excellence in Computer Science
* ASEF 3rd in Computer Science
  + Intel Excellence in Computer Science
  + Armed Forces Communications and Intelligence Award
* American Computer Science League Participant
* National Honors Society inductee
* Mu Alpha Theta Inductee

11th Grade (2011-2012)

* North American Computational Linguistics Olympiad
* USA Computing Olympiad
* 3rd at Alabama High School Programming Competition
* BEST Robotics competition

10th Grade (2010-2011)

* 7th at Alabama High School Programming Competition
* 10th at Hoover Math competition
* Competed in Chemistry Olympiad
* North American Computational Linguistics Olympiad

9th Grade (2009-2010)

* BEST Robotics Competition

# Abstract

The purpose of this project is to bridge the gap between learning tools and, to provide an easy to learn, English-like language that allows a sense of accomplishment one would find in making a program in a more conventional language such as Java or C++. In order to achieve this goal 3 classes were written in order to create the language with it's syntax. The language was then taught to a group of students who were then given a survey about the language and about Computer Science in general.

Table of Contents

[**Chapter 1** 6](#_Toc354742214)

[Introduction to the problem 6](#_Toc354742215)

[Statement of the Problem 7](#_Toc354742216)

[List of Terms and Definitions 8](#_Toc354742217)

[**Chapter 2** 9](#_Toc354742218)

[Introduction 9](#_Toc354742219)

[Lowering the Barriers to Programming 9](#_Toc354742220)

[Programming and Kids 10](#_Toc354742221)

[**Chapter 3** 13](#_Toc354742222)

[Introduction 13](#_Toc354742223)

[Materials 13](#_Toc354742224)

[The IDE Class 13](#_Toc354742225)

[The Compiler Class 14](#_Toc354742226)

[The Main Class 15](#_Toc354742227)

[Limitations 15](#_Toc354742228)

[Data Collection 16](#_Toc354742229)

[**Chapter 4** 17](#_Toc354742230)

[Data 17](#_Toc354742231)

[Data Analysis 18](#_Toc354742232)

[**Chapter 5** 19](#_Toc354742233)

[Summary 19](#_Toc354742234)

[Recommendations 20](#_Toc354742235)

[Future Implications 20](#_Toc354742236)

[**Literature Cited** 21](#_Toc354742237)

[**Appendix** 22](#_Toc354742238)

[EBNF Grammar 22](#_Toc354742239)

[Code 23](#_Toc354742240)

# Chapter 1

## Introduction to the problem

There are many jobs in computer science today. These jobs are often high paying but require a specific skill set such as programming to attain. This means there is a higher demand for computer scientists while not as many people know computer science. There are many efforts by the government to try fund education in science, engineering, technology, and math in order to increase interest in these areas. Many schools and universities recognize that computer science jobs are in high supply and have created tools to try to get younger students interested in computer science.

Alice is one of these tools, written in Java, which started development in 1997 by University of Virginia. Carnegie-Melon then continued the development of it. EA has partnered with Carnegie-Melon in order to use characters from The Sims in Alice 3.0. Before Java was sold to Oracle, Sun microsystems was working with Carnegie-Melon to try to globalize Alice. Jeroo is another tool that tries to teach object oriented concepts such as polymorphism, and objects. The user controls a creature called a jeroo on an island with different commands such as hop, pick, plant, turn, and toss. This limits what the user can do significantly. A user can only create programs to solve specific problems created by a teacher and not any problems in the real world. Unlike Alice Jeroo does not have a drag and drop interface.

Scratch is a more recent tool created by MIT in order to teach programming concepts to a younger audience. Scratch is aimed more towards elementary school to early middle school children whereas Alice is aimed towards late middle school to early high school. Scratch is similar to Alice in that it can only create simple games and movie; however, Scratch programs are in two dimensions as opposed to three dimensions like in Alice. Scratch has an online sharing system where people can put the games and movies they have created on the website and can even download what others have made in order to play or modify them. Python is a language created as a teaching language. Python is heavily influenced by Java. It features automatic memory management and is object oriented. Unlike java Python stresses readability and in doing so simplifies the code. This is an attempt to make it easier for someone who is learning a new language to start with Python.

There is a gap in the available tools. To someone who is computer savvy, but young and inexperienced in programming, they are either too easy and provide no sense of accomplishment or they are too hard and make the user turn away from programming. The purpose of my project is to bridge this gap, to provide an easy to learn, English-like language that allows a sense of accomplishment one would find in making a program in a standard programming language such as Java or C++.

## Statement of the Problem

My solution to the gap in tools and language is a language of my own. The language will be a whitespace dependent language such as Python but with simpler commands. It will also be interpreted, not compiled, and written in java The intention is that most statements will end up reading like a sentence where the subject is the variable, the verb is the assignment operator, and the predicate is what the variable is being assigned too. Another goal of the language is to try to eliminate operators as much as possible because they can become confusing and intimidating to someone new to programming. The idea is to promote a sense of pride in someone new to programming when they use this language to do something that can solve a problem as opposed to making a movie or game.

## List of Terms and Definitions

1. Buffer = A region of physical memory used to store data before transfer
2. Hash Table = A data structure that saves memory by using hash functions
3. Interpreter = A type of compiler that analyzes a program line by line
4. Java = An object oriented programming language developed by Oracle
5. Swing = A graphics framework provided by Oracle

# Chapter 2

## Introduction

The purpose of this project is to teach younger children about programming concepts while providing a simple, yet fulfilling language. The language is intended to be easy to learn but still allow the programmer to create useful programs that could be created in a language such as Java or Python. It will be English-like in its syntax. Each command, one line in the program, will be formatted similarly to a sentence. The language will be only a prototype with many more commands that could be added in the future.

## Lowering the Barriers to Programming

Kelleher and Pausch in their article, *Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers*, discuss various types of programming languages and how the ideas of trying to make programming languages more accessible. They categorize the various languages throughout the years. They also look at the mechanics of the language and what the language is trying to stress. They talk about different ways that languages try to simplify programming for beginners.

There are various mechanical barriers for a programming language, such as syntax. CLIP is, like those similar to many introduced in the article, “…focus[ed] on making the mechanics of programming more manageable.”(Kelleher –Pausch 49) This was done because syntax is often the most difficult thing to get used to when starting to program. Simplifying syntax allows the tool to get a larger audience and will end up influencing more people to get into programming and to eventually take on more complicated languages in the future. With syntax out of the equation the learner can focus on the problem solving aspect of programming rather than the tedium of perfecting its syntax. Teaching languages often focus on creating something entertaining rather than solving a problem. While those do teach some problem solving skills, CLIP attempts to focus on solving problems one might face while using a more fully functional language such as Java. This is not what a tool like Alice or Jeroo attempts to do, which is to teach ideas behind programming, not how it can be used to solve problems. CLIP is an intermediate step between tools like that and a language such as Python or Java.

Though a portion of the reasons why people don’t program are mechanical, there are many people who don’t program because of sociological barriers. These are sometimes harder to overcome than the mechanical barriers. According to their research there are “two different sociological barriers, the lack of social context for programming, and the lack of compelling contexts in which to learn programming.”(Kelleher-Pausch 50) This project does not delve into the ideas of the social barrier, nor does CLIP attempt to give a context in which to learn programming. CLIP is intended to be used in a classroom-like setting and as such the sociological barriers are not as pertinent to this project.

## Programming and Kids

With computers continuing to rise in importance it is becoming more and more important for people to understand computer science concepts as well as have some programming knowledge. It is hard to get children interested in learning concepts related to something that has traditionally been difficult to get started in. Many school teachers and administrators have seen the value of teaching the concepts to their students and as such many different tools have been made in order to interest kids while teaching them concepts.

Elementary schools very rarely teach computer science though many have been using computers for a while, “Yet the students typically learn very little about what computers are made of, how they work, and what makes them work.”(Dr. Viera Krnanová Proulx 2) Dr. Viera discusses how the curriculum for teaching needs to be based more on algorithms, how to discover them, and how to optimize them as opposed to programming. A language like CLIP would allow for something like this because of its extremely simplistic syntax; however, it is limited by its inability to handle any data structure or methods. Unlike other subjects such as the social science or mathematics, computer science requires the student to be precise with their language in order to achieve the intended result; however, it also allows almost infinite creative freedom with the way a problem can be solved. This promotes a different type of problem solving which can be helpful in almost any other subject the student takes. The required precision of phrasing helps train the student to be very concise with their words and actions when solving other types of problems.

Middle schools, like elementary schools, do not usually teach computer science, though many teach their students computer applications. Zachary Dodds, Mike Erlinger, and Elizabeth Sweedyk say in their article *Middle CS? Yes!* “A generation ago, computation was a tool with relatively little impact on day-to-day life… As a result, computational enthusiasm and computational skills are unbalanced.” What this means is that the older generation that only uses the computer for programs, like Excel and Word, don’t know how to teach skills that are needed in order to create innovations in the field of computer science. This leads to stagnation in the technology and a loss of interest by many. Currently there are more jobs in computer science than there are people to fill them. This can be attributed to the way computer science is taught—or not taught in many cases—and can easily be changed if vital concepts are introduced to children at a very young age. Doing so would promote it in a similar way math or science is promoted. Imagine if math or science was not taught in elementary and middle school, there would be a significant shortage of scientists, statisticians, and doctors. A shortage in these fields would mean many things including a shorter life span and an overall lower quality of life.

# Chapter 3

## Introduction

The CLIP IDE and Compiler were programmed in Java using Oracle’s SDK. The IDE used to create and run test cases was Eclipse, and open source IDE created by a foundation of the same name. In order to keep the design of the language extensible three classes were created to share the workload.

## Materials

Oracle’s Java Development Kit

Eclipse

TextAreaOutputStream class

## The IDE Class

The IDE class handles all elements of the graphical user interface (GUI). The class itself extends the JPanel class provided by the Java Development Kit (JDK). The reason this was done was to allow other JComponents-- such as JTextAreas, JButtons, etc…-- to be added to other JPanels in order to organize the GUI more clearly. The IDE uses a BorderLayout which allows for JComponents to be added to five zones in a JPanel or JFrame. Four JButtons, two JLabels, and two JTextAreas were arranged in a visually appealing way through the use of several JPanels and Layouts.

The building of the GUI occurs when an IDE object is instantiated. The IDE includes a JFileChooser. This is an organized GUI file opener as the name suggests, it makes opening a file user friendly without requiring the extensive time required to create an equivalent. When the open button is clicked by the user, the JFileChooser will open and prompt the user to select the intended file. If the file that the user chooses is not a \*.txt file then it will not be opened and the user will be told that the file failed to open. If the save button is clicked then the JFileChooser opens and asks the user where to save and what to name the file, similar to other programs such as Microsoft’s Word. When the run button is clicked the program takes all the text from the editing window and calls the Compiler’s run method. The Clear button removes all text from the output console.

## The Compiler Class

The Compiler class takes the program created by the user and steps through each line to produce output. This class extends the Thread class. The Compiler is contained in the IDE, meaning an IDE has a Compiler in it, though they are still separate classes. The Compiler class uses the TextAreaOutputStream class in order to re-route System.out to the console JTextArea. This was done in an attempt to solve a problem where if looping the JTextArea would only update after the loop was done; however, this method did not work, nor did any other method, but it remains in because of the ease of use for redirecting System.out.

One of the longest methods in this class is the parseLine() method. This method is the backbone of the entire compiler. When the run button is clicked the run() method is called. The run method takes all the lines written in the program and feeds them into parseLine() one at a time. The parseLine() method then looks at the contents of the line and determines what to do. If there is math the method sends that line to the calculate() method where the math statement is solved. If the line starts with a while or if statement then the conditional is sent to the checkCondition(String line) method where the result will be a number, Zero is false and non-zero is true. The “say” keyword will output whatever the user tells the program to say to the console. When run is first clicked the program will check for mismatched parenthesis, uneven number of if/whiles and ends, and mismatched quotation marks. While the program is running through each line it will also check for syntax errors. If an error exists the program will stop running and inform the user.

## The Main Class

This is the driver class of the language. This creates a JFrame using the System’s look and feel. The look and feel is determined by the operating system that the user has installed. This creates a more familiar environment for people no matter what system they run the program on. An instance of IDE is created and then added to the JFrame. The JFrame is then displayed and then the program runs from there.

## Limitations

There were a few limitations when creating this project. One of the biggest was that the JTextArea does not update dynamically. This means that the text being displayed would not update until the loop that was writing to it stopped trying to write. Several fixes were attempted. One fix was to put a forced delay of a few seconds into the loop. This only made the loops longer and didn’t help. Another solution tried was to have the updating of the JTextArea running on a different thread but this solution also did not work. The last thing I tried was redirecting System.out to the JTextArea in order to force the JTextArea to flush its buffer but this did not help either. The only other limitation I had was time. With more time, the language would be more fleshed out and have more commands and functionality. This language will not be as featured as other languages such as Java or Python but will provide what is necessary in order to teach concepts to users new to programming. The language is interpreted and therefore can’t list explicit syntactical errors.

## Data Collection

1. A group of 8th graders was obtained as a test group
2. They were then taught to use the language, were shown a few examples of some programs created in the language, and then asked to create a program of their own

The students were then given a post-lesson survey

# Chapter 4

## Data

The data has values ranging from 1-5, 5 is the strongest positive answer and 1 is the strongest negative answer. Below the data is a histogram that shows the frequency of answers given by the students.

|  |  |  |  |
| --- | --- | --- | --- |
| Experience with CS | Interest in CS | Easy to Learn | Structured like a Sentence |
| 4 | 2 | 4 | 5 |
| 5 | 3 | 5 | 5 |
| 5 | 2 | 5 | 5 |
| 3 | 1 | 2 | 2 |
| 1 | 3 | 4 | 4 |
| 2 | 1 | 3 | 4 |
| 1 | 2 | 1 | 5 |
| 2 | 3 | 3 | 4 |
| 2 | 1 | 3 | 4 |
| 1 | 1 | 2 | 2 |
| 3 | 2 | 3 | 4 |
| 4 | 3 | 4 | 5 |
| 4 | 3 | 3 | 4 |
| 3 | 2 | 5 | 5 |
| 3 | 1 | 2 | 3 |
| 3 | 1 | 2 | 3 |
| 4 | 2 | 5 | 5 |
| 2 | 1 | 2 | 2 |
| 4 | 3 | 4 | 5 |
| 5 | 3 | 1 | 5 |
| 3 | 3 | 3 | 2 |

## Data Analysis

The mean values shown in the tables and figures below support that CLIP was an easy to learn programming language for the groups with high interest and high experience. The high experience group should be taken with a grain of salt because asking 8th graders to rate their experience in something they more than likely haven’t taken a class on it is likely that they will rate themselves as high experience when in reality they are not.

# Chapter 5

## Summary

There are many jobs in computer science today. These jobs are often high paying but require a specific skill set such as programming to attain. This means there is a higher demand for computer scientists than there are people who know computer science. There are many efforts by the government to try fund education in science, engineering, technology, and math in order to increase interest in these areas. Many schools and universities recognize that computer science jobs are in high supply and have created tools to try to get younger students interested in computer science.

There is a gap in the current teaching tools and larger more complex languages. The purpose of CLIP is the fill the gap in order to smooth the transition between learning tools and complex languages. It attempts to do this by creating a simple, English-like syntax. It is intended to have few, essential commands and to be easy to read. CLIP is a parsed language developed in Java using Oracle’s JDK. There are three classes implemented to accomplish the goal. The Compiler class contains the parser and is responsible for interpreting the user’s commands and outputting the result. The IDE class provides the user interface for CLIP and contains all elements of it. The Main class is just a driver class and declares the IDE and the compiler and allows people to run the CLIP executable.

For data, a group of 8th grade fine arts students were taught CLIP during one class period, approximately 45 minutes. An anonymous survey was then given to the children asking various questions about ease of use of CLIP, experience in Computer Science, and interest in Computer Science. The mean of the data was then found.

## Recommendations

In the future I plan to add on to CLIP. One of the first things I would add to the program is the ability for the programmer to prompt the user for input. This would open up more possibilities for the types of programs that could be created. I would also like to try to implement some simple file input-output. There are near infinite things that can be added to CLIP. Adding an import system similar to that of Java, or the header files for C, would allow even more extensibility of the language. While implementing such a system would require a slight rework of the parser, the system itself would not require a huge overall. A simple if statement could be added to check if some of the used commands are in the imported packages. While at its current stage CLIP is fairly simplistic, it has the capacity to grow into a very large, easy to use, and useful language.

## Future Implications

In its current state CLIP can easily be used in a classroom environment to teach beginning programmers. While learning tools like Alice are recommended before learning something like CLIP, with a teacher trained in the material students could easily learn it. As can be seen from the results stated earlier, CLIP was shown to be easy to learn and easy to read. It currently provides all the commands necessary to make normal classroom programs, but with more time can be expanded into something all encompassing, like Java, while remaining simple and easy to read. In a future state CLIP could easily be something taught around the US as a way to introduce people to programming and its concepts.

# Literature Cited

A Visual Guide to Layout Managers retrieved September 7, 2012, from

<http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

Caitlin Kelleher and Randy Pausch. (2005).

Lowering the Barriers to Programming: A Taxonomy of Programming

Environments and Languages for Novice Programmers. *ACM Computing Surveys, Vol. 37, (2).* Retrieved August 30, 2012, from <http://dl.acm.org/citation.cfm?id=1089734>

How to Set Look and Feel retrieved September 14, 2012, from

<http://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html>

How to Use File Choosers retrieved October 5, 2012, from

<http://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html>

Viera Proulx (1993) Computer Science in Elementary and Secondary Schools retrieved

November 29, 2012, from [www.ccs.neu.edu/home/vkp/Papers/Gmunden93.doc](http://www.ccs.neu.edu/home/vkp/Papers/Gmunden93.doc)

Zachary Dodds, Mike Erlinger, and Elizabeth Sweedyk (2012) Middle School CS? Yes!

Retrieved November 29 2012, from CS K-8

# Appendix

## EBNF Grammar

1. statement: iteration\_statement | if\_statement | output\_statement | assignment ;
2. statement\_block: statement statement\_block | statement ;
3. assignment: math\_expression | string\_expression ;
4. math\_expression: number operator number | number operator math\_expression | math\_expression operator number | math\_expression operator math\_expression | number;
5. number: {digit} ;
6. digit: “0”| … |“9” ;
7. string: “”” {character} “”” ;
8. character: “A”|…|”z” ;
9. operator: “add” | “subtract” | “divide” | “multiply” ;
10. iteration\_statement: “while” condition statement\_block “end” ;
11. if\_statement: “if” condition statement\_block “end” ;
12. condition: math\_expression conditional\_operator math\_expression | condition and\_or condition | condition conditional\_operator condition ;
13. conditional\_operator: “less” | “greater” | ”le” | “ge” ;
14. and\_or: “and” | “or” ;
15. output\_statement: “say” math\_expression | “say” number | “say” string ;

## Code

**package** com;

**import** java.io.PrintStream;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.Map;

**import** javax.swing.JTextArea;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* This is the compiler in completion for partial graduation at the Alabama School of Fine Arts \*

\* Casey Michael Barnette \*

\* Senior Research Project \*

\* \*

\* This is the compiler/interpretor portion of the program \*

\* For the purposes of bookmarking a **TODO** is used so that I may jump between portions of the code\*

\* in eclipse \*

\* \*

\* Periods can't be used in variable names because I said so \*

\* \*

\* Started: August 29 2012 \*

\* Prototype 1: September 17, 2012 \*

\* Prototype 2: October 7, 2012 \*

\* Prototype 3: October 10, 2012 \*

\* Prototype 4: October 23, 2012 \*

\* Finished: December 22, 2012 \*

\* Last Modified: November 13, 2012 \*

\* Next Prototype Milestone: N/A \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//The compiler class extends thread so that swing isn't overloaded with tasks

**public** **class** Compiler **extends** Thread{

//The first string refers to the name while the second string refers to the value

//Variables can't have spaces, true, false, or any other keyword

**private** Map<String, String> variables;

// Put back into use after prototype 3 to allow for looping

**private** ArrayList<String> loopCommands = **new** ArrayList<String>();

**private** **boolean** error;

**private** **boolean** elif;

**private** **boolean** loop;

**private** String errMessage;

**private** String[] keywords;

//this is the console

**public** **static** JTextArea *console*;

**public** **static** JTextArea *program*;

**public** Compiler(JTextArea c, JTextArea p)

{

*console* = c;

*program* = p;

variables = **new** HashMap<String,String>();

error = **false**;

loop = **false**;

elif = **false**;

errMessage = "";

createKeywords();

//This is setting System.out to print at the text area

TextAreaOutputStream textOut = **new** TextAreaOutputStream(*console*);

PrintStream outStream = **new** PrintStream(textOut, **true**);

System.*setOut*(outStream);

}

//These are keywords that can't be used as variables

**private** **void** createKeywords()

{

keywords = **new** String[4];

keywords[0] = "is";

keywords[1] = "if";

keywords[2] = "say";

keywords[3] = "while";

}

**public** **boolean** isError()

{

**return** error;

}

**public** **boolean** canRun()

{

**return** *program*.getText() != "" || *program*.getText()!=**null**;

}

**public** **void** run()

{

String consoleProgram = *program*.getText();

String[] consoleCommands = consoleProgram.split("\n");

//Check Error returns true if there is an error

**if**(checkError(consoleProgram))

{

System.*out*.println(errMessage);

**return**;

}

**for**(**int** i=0;i<consoleCommands.length;i++)

{

String curCommand = consoleCommands[i];

//System.out.println(curCommand);

parseLine(curCommand);

**if**(error)

{

System.*out*.println("Syntax Error at line: " + i + "\n" + "Line: " + curCommand);

error = **false**;

**return**;

}

}

}

//This checks if there is an even number of a specific token, used in the check error methods

**private** **boolean** checkEven(String program, String token)

{

**if**(program.contains(token))

{

**int** counter = 0;

**for**(**int** i=0;i<program.length();i++)

{

**if**(program.substring(i).contains(token))

{

i = program.indexOf(token,i);

counter++;

}

**else**

{

i=program.length();

}

}

**if**(counter%2 == 0)

{

**return** **true**;

}

**else**

{

**return** **false**;

}

}

**else**

{

**return** **true**;

}

}

//This checks if parenthesis match up

**private** **boolean** checkParenth(String program)

{

**if**(!program.contains("(") && !program.contains(")"))

{

**return** **true**;

}

**int** lIndex = 0;

**for**(**int** i=0;i<program.length();i++)

{

**if**(program.substring(i).contains("(") && program.substring(lIndex).contains(")"))

{

lIndex = program.indexOf(")",lIndex);

i = program.indexOf("(",i);

}

**else**

{

**return** **false**;

}

}

**return** **true**;

}

//this checks if while, end, and if keywords are all in the appropriate amount

**private** **boolean** checkIfWhile(String program)

{

**int** ifwhile =0;

**int** ends = 0;

**for**(**int** i=0;i<program.length();i++)

{

**if**(program.substring(i).contains("if"))

{

ifwhile++;

i = program.indexOf("if",i) +1;

}

}

**for**(**int** i=0;i<program.length();i++)

{

**if**(program.substring(i).contains("else"))

{

ifwhile++;

i = program.indexOf("else",i) +1;

}

}

**for**(**int** i=0;i<program.length();i++)

{

**if**(program.substring(i).contains("while"))

{

ifwhile++;

i = program.indexOf("while",i)+1;

}

}

**for**(**int** i=0;i<program.length();i++)

{

**if**(program.substring(i).contains("end"))

{

ends++;

i = program.indexOf("end",i)+1;

}

}

**return** ends == ifwhile;

}

//this is the overall check error method that does some pre-runtime checking

**private** **boolean** checkError(String program) {

**if**(program.contains(keywords[0]) || program.contains(keywords[1]) || program.contains(keywords[2]) ||

program.contains(keywords[3]))

{

**if**(checkEven(program,"\""))

{

**if**(checkParenth(program))

{

**if**(checkIfWhile(program))

{

**return** **false**;

}

**else**

{

errMessage = "Number of End statements does not match number of If and

Whiles";

**return** **true**;

}

}

**else**

{

errMessage = "Parenthesis mis-match";

**return** **true**;

}

}

errMessage = "Incorrect number of \" either add one or remove one";

**return** **true**;

}

errMessage = "No required keywords(" + keywords[0] + ", " + keywords[1] + ", " + keywords[2] + ", " + keywords[3] + ") were found";

**return** **true**;

}

//This runs the loop based on conditions

**private** **void** runLoop()

{

String condition = loopCommands.get(0);

condition = condition.replace("while ","");

loopCommands.remove(0);

//System.out.println(loopCommands);

**while**(checkCondition(condition))

{

**for**(**int** i=0;i<loopCommands.size();i++)

{

parseLine(loopCommands.get(i));

}

}

loopCommands.clear();

}

**private** **void** runIf()

{

String condition = loopCommands.get(0);

condition = condition.replace("if ","");

loopCommands.remove(0);

**if**(checkCondition(condition))

{

**for**(**int** i=0;i<loopCommands.size();i++)

{

parseLine(loopCommands.get(i));

}

}

loopCommands.clear();

}

//**TODO** checkConditon

**private** **boolean** checkCondition(String line)

{

//For my language false is a 0 and true is any other number

// Valid operators include or, and, less, equals, greater, le, ge

//System.out.println("debug");

//this is for parenthesis support

**if**(line.charAt(0) == ' ')

line = removeLeading(line);

//System.out.println(line);

**if**(line.contains("(") && line.contains(")"))

{

**int** p1 = line.indexOf("(");

**int** p2 = line.lastIndexOf(")");

String temp3 = line.substring(p1+1,p2);

line = line.replace(line.substring(p1,p2+1), checkCondition(temp3)? "0":"1");

}

//

ArrayList<String> lines = **new** ArrayList<String>();

String[] t = line.split(" ");

**for**(**int** i=0;i<t.length;i++)

{

lines.add(t[i]);

}

**while**(lines.size()>1)

{

**for**(**int** i=0;i<lines.size()-1;i++)

{

**if**(lines.get(i).equals("greater"))

{

**if**(isNumeric(lines.get(i-1)))

{

**float** a = Float.*parseFloat*(lines.get(i-1));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a>b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

**else** **if**(isNumeric(variables.get(lines.get(i-1))))

{

**float** a = Float.*parseFloat*(variables.get(lines.get(i-1)));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a>b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

}

**else** **if**(lines.get(i).equals("less"))

{

**if**(isNumeric(lines.get(i-1)))

{

**float** a = Float.*parseFloat*(lines.get(i-1));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a<b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

**else** **if**(isNumeric(variables.get(lines.get(i-1))))

{

**float** a = Float.*parseFloat*(variables.get(lines.get(i-1)));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a<b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

}

**else** **if**(lines.get(i).equals("ge"))

{

**if**(isNumeric(lines.get(i-1)))

{

**float** a = Float.*parseFloat*(lines.get(i-1));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a>=b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

**else** **if**(isNumeric(variables.get(lines.get(i-1))))

{

**float** a = Float.*parseFloat*(variables.get(lines.get(i-1)));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a>=b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

}

**else** **if**(lines.get(i).equals("le"))

{

**if**(isNumeric(lines.get(i-1)))

{

**float** a = Float.*parseFloat*(lines.get(i-1));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a<=b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

**else** **if**(isNumeric(variables.get(lines.get(i-1))))

{

**float** a = Float.*parseFloat*(variables.get(lines.get(i-1)));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a<=b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

}

**else** **if**(lines.get(i).equals("equals"))

{

**if**(!isNumeric(variables.get(lines.get(i-1))) && !isNumeric(variables.get(lines.get(i+1))))

{

**return** variables.get(lines.get(i-1)).equals(variables.get(lines.get(i+1)));

}

**else** **if**(isNumeric(lines.get(i-1)))

{

**float** a = Float.*parseFloat*(lines.get(i-1));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a==b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

**else** **if**(isNumeric(variables.get(lines.get(i-1))))

{

**float** a = Float.*parseFloat*(variables.get(lines.get(i-1)));

**float** b;

**if**(isNumeric(lines.get(i+1)))

{

b = Float.*parseFloat*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Float.*parseFloat*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **false**;

}

}

**if**(a==b)

{

lines.remove(i+1);

lines.set(i, "1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i,"0");

lines.remove(i-1);

}

}

}

**else** **if**(lines.get(i).equals("and"))

{

//System.out.println(lines.get(i+1));

**if**(!(lines.get(i-1).equals("0") || lines.get(i+1).equals("0")))

{

lines.remove(i+1);

lines.set(i,"1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i, "0");

lines.remove(i-1);

}

}

**else** **if**(lines.get(i).equals("or"))

{

**if**(!(lines.get(i-1).equals("0") && lines.get(i+1).equals("0")))

{

lines.remove(i+1);

lines.set(i,"1");

lines.remove(i-1);

}

**else**

{

lines.remove(i+1);

lines.set(i, "0");

lines.remove(i-1);

}

}

}

}

**return** lines.get(0).equals("1");

}

//**TODO** start of parseLine

**private** **double** parseLine(String line)

{

**if**(loop && !line.equals("end"))

{

//System.out.println("I have been added");

loopCommands.add(line);

}

**else** **if**(line.equals("end") && loop)

{

loop = **false**;

**if**(!elif)

runLoop();

**else**

runIf();

elif = **false**;

}

**else** **if**(line.contains("while") || line.contains("if"))

{

loop = **true**;

loopCommands.clear();

line = line.trim();

loopCommands.add(line);

elif = line.contains("if") ? **true** : **false**;

}

//**TODO** this is for printing stuff out until the IDE is up it only prints to the console

**else** **if**(line.contains("say") && !line.contains("\"") && !(line.contains("add") || line.contains("subtract") || line.contains("multiply")

|| line.contains("divide") || line.contains("exp")))

{

line.trim();

**final** String[] lines = line.split(" ");

**if**(variables.get(lines[1])!=**null** || variables.get(lines[1])!="null" && lines.length<3)

{

System.*out*.println(variables.get(lines[1]));

}

**else**

{

error = **true**;

**return** -1;

}

}

**else** **if**(line.contains("say") &&(line.contains("add") || line.contains("exp") || line.contains("subtract") ||

line.contains("multiply") || line.contains("divide")))

{

String[] lines = line.split("say");

lines[1] = removeLeading(lines[1]);

String temp = "";

**for**(**int** i=1;i<lines.length;i++)

{

temp+=lines[i];

}

System.*out*.println(temp);

}

**else** **if**(line.contains("say") && line.indexOf("\"")!=-1 && line.lastIndexOf("\"")!=line.indexOf("\""))

{

line.trim();

**int** x1 = line.indexOf("\"");

**int** x2 = line.indexOf("\"", x1+1);

String temp = line.substring(x1+1,x2);

System.*out*.println(temp);

}

//**TODO** this is for setting variables to other variables or numbers

**else** **if** (!(line.contains("add") || line.contains("exp") || line.contains("subtract") || line.contains("multiply") ||

line.contains("divide") || line.contains("\"") || line.equals("")))

{

line.trim();

String[] lines = line.split(" ");

**if**(!isNumeric(lines[2]))

{

**if**(variables.get(lines[2]) != **null** || variables.get(lines[2])!="null")

{

variables.put(lines[0], variables.get(lines[2]));

}

**else**

{

error = **true**;

**return** -1;

}

}

**else**

{

variables.put(lines[0], lines[2]);

}

}

//**TODO** this is for setting string variables

**else** **if**(line.indexOf("\"")!=-1 && line.lastIndexOf("\"")!=line.indexOf("\"") && !(line.contains("add")

||line.contains("exp")||line.contains("subtract")||line.contains("multiply")||line.contains("divide")))

{

line.trim();

**int** x1 = line.indexOf("\"");

**int** x2 = line.lastIndexOf("\"");

variables.put(line.substring(0,line.indexOf(" ")),line.substring(x1+1,x2));

}

//**TODO** this is for numbers

**else** **if**((line.contains("add")||line.contains("subtract")||line.contains("multiply")||line.contains("divide") ||

line.contains("exp")) && line.contains("is"))

{

line.trim();

String[] lines = line.split("is");

//System.out.println(lines[1]);

lines[1] = calculate(lines[1]);

lines[0] = trim(lines[0]);

//System.out.println(lines[0]);

**if**(lines[1]!=**null**)

variables.put(lines[0], lines[1]);

**else**

**return** -1;

}

**else** **if** (line.contains("add")||line.contains("subtract")||line.contains("multiply")||line.contains("divide") &&

!line.contains("is"))

{

**return** Double.*parseDouble*(calculate(line));

}

**else** **if**(line.equals(""))

{

}

**else**

{

error = **true**;

**return** -1;

}

**return** 0;

}

//**TODO** this is used to calculate math expressions

**private** String calculate(String line)

{

**if**(line.charAt(0) == ' ')

line = removeLeading(line);

//System.out.println(line);

**if**(line.contains("(") && line.contains(")"))

{

**int** p1 = line.indexOf("(");

**int** p2 = line.lastIndexOf(")");

String temp3 = line.substring(p1+1,p2);

line = line.replace(line.substring(p1,p2+1), calculate(temp3));

}

ArrayList<String> lines = **new** ArrayList<String>();

String[] t = line.split(" ");

**for**(**int** i=0;i<t.length;i++)

{

lines.add(t[i]);

}

**boolean** exp =**false**;

**boolean** md = **false**;

**while**(lines.size()>1)

{

//System.out.println(lines);

**for**(**int** i=0;i<lines.size()-1;i++)

{

**if**(!exp && lines.get(i).equals("exp"))

{

**double** a;

**double** b;

**if**(isNumeric(lines.get(i-1)))

{

a = Double.*parseDouble*(lines.get(i-1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i-1))))

{

a = Double.*parseDouble*(variables.get(lines.get(i-1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

**if**(isNumeric(lines.get(i+1)))

{

b = Double.*parseDouble*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Double.*parseDouble*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

String t1 = "";

t1 += Math.*pow*(a, b);

//System.out.println(t1+ " "+ a + " " + b );

lines.remove(i+1);

lines.set(i, t1);

lines.remove(i-1);

**break**;

}

**else** **if**(exp && lines.get(i).equals("multiply") && !md)

{

**double** a;

**double** b;

**if**(isNumeric(lines.get(i-1)))

{

a = Double.*parseDouble*(lines.get(i-1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i-1))))

{

a = Double.*parseDouble*(variables.get(lines.get(i-1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

**if**(isNumeric(lines.get(i+1)))

{

b = Double.*parseDouble*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Double.*parseDouble*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

String t1 = "";

t1 += a\*b;

lines.remove(i+1);

lines.set(i, t1);

lines.remove(i-1);

**break**;

}

**else** **if**(lines.get(i).equals("divide") && !md)

{

**double** a;

**double** b;

**if**(isNumeric(lines.get(i-1)))

{

a = Double.*parseDouble*(lines.get(i-1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i-1))))

{

a = Double.*parseDouble*(variables.get(lines.get(i-1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

**if**(isNumeric(lines.get(i+1)))

{

b = Double.*parseDouble*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Double.*parseDouble*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

**if**(b==0)

{

error = **true**;

**return** "-1";

}

String t1 = "";

t1 += a/b;

lines.remove(i+1);

lines.set(i, t1);

lines.remove(i-1);

**break**;

}

**else** **if**(md && lines.get(i).equals("add"))

{

**double** a;

**double** b;

**if**(isNumeric(lines.get(i-1)))

{

a = Double.*parseDouble*(lines.get(i-1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i-1))))

{

a = Double.*parseDouble*(variables.get(lines.get(i-1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

**if**(isNumeric(lines.get(i+1)))

{

b = Double.*parseDouble*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Double.*parseDouble*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

String t1 = "";

t1 += a+b;

lines.remove(i+1);

lines.set(i, t1);

lines.remove(i-1);

**break**;

}

**else** **if**(md && lines.get(i).equals("subtract"))

{

**double** a;

**double** b;

**if**(isNumeric(lines.get(i-1)))

{

a = Double.*parseDouble*(lines.get(i-1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i-1))))

{

a = Double.*parseDouble*(variables.get(lines.get(i-1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

**if**(isNumeric(lines.get(i+1)))

{

b = Double.*parseDouble*(lines.get(i+1));

}

**else**

{

**if**(isNumeric(variables.get(lines.get(i+1))))

{

b = Double.*parseDouble*(variables.get(lines.get(i+1)));

}

**else**

{

error = **true**;

**return** **null**;

}

}

String t1 = "";

t1 += a-b;

lines.remove(i+1);

lines.set(i, t1);

lines.remove(i-1);

**break**;

}

**if**(exp && i==lines.size()-2 && !md)

{

md = **true**;

}

**if**(i==lines.size()-2 && !exp)

{

exp = **true**;

}

}

}

**return** lines.get(0);

}

**private** String removeLeading(String line)

{

String temp = "";

**for**(**int** i=1;i<line.length();i++)

{

temp+=line.charAt(i);

}

**return** temp;

}

//**TODO** this is the trimming stuff for the strings

**private** String trim(String string) {

String ret = "";

**for**(**int** i=0;i<string.length();i++)

{

**if**(string.charAt(i) != ' ' )

{

ret+=string.charAt(i);

}

}

//System.out.println(ret);

**return** ret;

}

//**TODO** this is to check if a string is a number

**public** **boolean** isNumeric(String s)

{

**for**(**int** i=0;i<s.length();i++)

{

**if**(!Character.*isDigit*(s.charAt(i))&& s.charAt(i)!='.' && s.charAt(i)!='-')

{

**return** **false**;

}

}

**return** **true**;

}

}

**package** com;

**import** java.awt.BorderLayout;

**import** java.awt.Insets;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.PrintStream;

**import** java.io.PrintWriter;

**import** java.util.Scanner;

**import** javax.swing.JButton;

**import** javax.swing.JFileChooser;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.JTextArea;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* **@author** Casey Barnette \*

\* This class contains an instance of the Compiler class and Creates all the necessary Graphical User Interface elements\*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** **class** IDE **extends** JPanel **implements** ActionListener{

//This is the instantiation step

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** JButton open;

**private** JButton run;

**private** JButton save;

**private** JButton clear;

**private** JButton help;

**private** JTextArea console;

**private** JTextArea program;

**private** JLabel consoleLabel;

**private** String message;

**private** **final** JFileChooser fileChooser;

**private** Compiler compiler;

//**TODO** this creates the layout and components for the IDE

**public** IDE()

{

**super**(**new** BorderLayout());

//this loads the string for the help box

message = "To use the say command:\n say <variable or quoted string>\n\nAssignment:\n <variable> is <number expression, variable, or string>\n\nWhile Loop:\n while <condition>\n <statements>\n end\n\nIf Statements:\n if <condition> \n <statements> \n end";

//This is the file chooser instantiation

fileChooser = **new** JFileChooser();

//Open Button

open = **new** JButton("Open");

open.addActionListener(**this**);

//open.setBorder(border);

//Run Button

run = **new** JButton("Run");

run.addActionListener(**this**);

//run.setBorder(border);

//Save Button

save = **new** JButton("Save");

save.addActionListener(**this**);

//Clear Button

clear = **new** JButton("Clear");

clear.addActionListener(**this**);

//Help Button

help = **new** JButton("Help");

help.addActionListener(**this**);

//JTextArea and JScrollBar are created

console = **new** JTextArea(10, 50);

//console.setBorder(border);

console.setMargin(**new** Insets(5,5,5,5));

//This is setting System.out to print at the text area

TextAreaOutputStream textOut = **new** TextAreaOutputStream(console);

PrintStream outStream = **new** PrintStream(textOut, **true**);

System.*setOut*(outStream);

JScrollPane scroll = **new** JScrollPane(console);

consoleLabel = **new** JLabel("Console:");

//panel for the scroll pane and label

JPanel consolePanel = **new** JPanel(**new** BorderLayout());

consolePanel.add(consoleLabel,BorderLayout.*WEST*);

consolePanel.add(scroll, BorderLayout.*SOUTH*);

//The Program Editor

program = **new** JTextArea(20,100);

JScrollPane scroll2 = **new** JScrollPane(program);

JLabel programLabel = **new** JLabel("Program Editor:");

JPanel editorPanel = **new** JPanel(**new** BorderLayout());

editorPanel.add(programLabel,BorderLayout.*WEST*);

editorPanel.add(scroll2,BorderLayout.*SOUTH*);

//Separate Panels for the buttons then putting

JPanel buttonPanel = **new** JPanel();

buttonPanel.add(open);

buttonPanel.add(save);

buttonPanel.add(run);

buttonPanel.add(clear);

buttonPanel.add(help);

//Panel that combines the button Panel and the editor panel

JPanel userPanel = **new** JPanel(**new** BorderLayout());

userPanel.add(buttonPanel, BorderLayout.*WEST*);

userPanel.add(editorPanel,BorderLayout.*SOUTH*);

//Adds everything to the main panel a.k.a this

**this**.add(userPanel, BorderLayout.*CENTER*);

**this**.add(consolePanel, BorderLayout.*SOUTH*);

//This is creating the compiler

compiler = **new** Compiler(console,program);

}

**public** JTextArea getConsole()

{

**return** console;

}

//**TODO** This handles what happens when buttons are pressed

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getSource() == open)

{

**int** returnVal = fileChooser.showOpenDialog(IDE.**this**);

**if** (returnVal == JFileChooser.*APPROVE\_OPTION*) {

File file = fileChooser.getSelectedFile();

**if**(file.getName().contains(".txt"))

{

program.setText("");

//This loads the file into the program editor

**try**

{

Scanner in = **new** Scanner(file);

**while**(in.hasNext())

{

program.append(in.nextLine()+"\n");

}

}

**catch**(Exception exception)

{

System.*out*.print("This shouldn't happen ever \n");

}

}

**else**

{

System.*out*.print("Error: Can't open that file type \n");

}

}

}

**else** **if**(e.getSource() == save)

{

//This is the code to save your program

**int** returnVal = fileChooser.showSaveDialog(IDE.**this**);

**if**(returnVal == JFileChooser.*APPROVE\_OPTION*)

{

PrintWriter out;

**try** {

out = **new** PrintWriter(**new** BufferedWriter(**new** FileWriter(fileChooser.getSelectedFile())));

program.write(out);

out.close();

} **catch** (IOException e1) {

e1.printStackTrace();

}

}

}

**else** **if**(e.getSource() == run)

{

//This takes what is in the program editor and runs the user created program

compiler = **new** Compiler(console,program);

**if**(compiler.canRun())

{

compiler.run();

}

**else**

{

System.*out*.print("No program loaded \n");

}

}

**else** **if**(e.getSource() == clear)

{

console.setText("");

}

**else** **if**(e.getSource() == help)

{

JOptionPane.*showMessageDialog*(**this**,message,"Help",JOptionPane.*QUESTION\_MESSAGE*);

}

}

}

**package** com;

**import** javax.swing.JFrame;

**import** javax.swing.UIManager;

**public** **class** Main {

**public** **static** **void** main(String[] args)

{

//The following code sets the look and feel to the user's operating system

**try**

{

UIManager.*setLookAndFeel*(UIManager.*getSystemLookAndFeelClassName*());

}

**catch**(Exception e)

{

}

UIManager.*put*("swing.boldMetal", Boolean.*FALSE*);

IDE ide = **new** IDE();

ide.setFocusable(**true**);

ide.requestFocus();

//JFrame creation

JFrame frame = **new** JFrame();

JFrame.*setDefaultLookAndFeelDecorated*(**true**);

frame.setTitle("CLIP");

frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

frame.setResizable(**false**);

frame.add(ide);

frame.pack();

frame.setVisible(**true**);

}

}