

UNIVERSITY OF THE PHILIPPINES MANILA
COLLEGE OF ARTS AND SCIENCES
DEPARTMENT OF PHYSICAL SCIENCES AND MATHEMATICS

Novice Assistance in Java Introduction

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Salcedo, Najinar Raysal Marie G.

May 2016

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled “Novice Assistance in Java Introduction” prepared and submitted by Najinar Raysal Marie G. Salcedo in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Ma. Sheila A. Magboo, M.S.

Adviser

EXAMINERS:

	Approved	Disapproved
1. Gregorio B. Baes, Ph.D. (candidate)	_____	_____
2. Avegail D. Carpio, M.S.	_____	_____
3. Richard Bryann L. Chua, Ph.D. (candidate)	_____	_____
4. Perlita E. Gasmen, M.S. (candidate)	_____	_____
5. Marvin John C. Ignacio, M.S. (candidate)	_____	_____
6. Vincent Peter C. Magboo, M.D., M.S.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.S.

Unit Head

Mathematical and Computing Sciences Unit
Department of Physical Sciences and
Mathematics

Marcelina B. Lirazan, Ph.D.

Chair

Department of Physical Sciences
and Mathematics

Alex C. Gonzaga Ph.D., Dr. Eng'g

Dean

College of Arts and Sciences

ABSTRACT

Novice Assistance in Java Introduction is an extension developed for BlueJ that helps new programmers being introduced to Java in their debugging. Its primary objective is to give a clearer explanation and point out the root cause of a compile error. The compile errors currently thrown by the compiler error don't necessarily point the novice programmers to the right direction. With the help of NAJI, these compile errors are processed to have a more detailed output like background, root cause, and example.

Keywords: BlueJ, Java, Extension, Novice Programmer

LIST OF FIGURES

Figure 1: Sample Output of Espresso.....	12
Figure 2: Derivation tree.....	14
Figure 3: Context Diagram	18
Figure 4: Use-Case Diagram.....	19
Figure 5: Data Flow Diagram	21
Figure 6: Flowchart with swimlanes.....	22
Figure 7: Class Diagram	23

CONTENTS

Acceptance Sheet	i
Abstract	ii
List of Figures	iii
I. INTRODUCTION	1
A. Background of the Study.....	1
B. Statement of the Problem	1
C. Objectives of the Study	3
D. Significance of the Study	3
E. Scope and Limitations	4
F. Assumptions	5
II. REVIEW OF RELATED LITERATURE	6
III. THEORETICAL FRAMEWORK	11
IV. DESIGN AND IMPLEMENTATION	18
A. Context Diagram	18
B. Use-Case Diagram.....	19
C. Data Flow Diagram	21
D. Swimlane Flowchart	22
E. Class Diagram	23
F. Technical Requirements	24
V. RESULTS	25
VI. DISCUSSION.....	52
VII. CONCLUSION	53
VIII. RECOMMENDATION	54
IX. BIBLIOGRAPHY.....	55
X. APPENDIX.....	59
XI. ACKNOWLEDGEMENT	80

I. INTRODUCTION

A. Background of the Study

Debugging is a process of removing bugs from coded programs. If the program is not working according to design, developers must debug the source code and fix the issues. [1] It has been known to account for more than half of the effort and time spent in software development. Structured programming lowers the risk of faulty programs but it can't guarantee bug-free programs.

Level of debugging skill is one of the major differences between novice and expert programmers. Experts make fewer errors and locate and correct bugs faster than novices. [2] Debugging training is even more needed by novice programmers. Unlike experienced programmers who can easily locate errors or root causes of a problem, novice programmers often turn to trial and errors for debugging. Programming classes usually concentrate on teaching programming languages, syntax, problem analysis, and design. Little time is saved for practicing debugging. Because of that, novice programmers resort to develop their own skills as they struggle to find the root causes of errors found in their programs. [3]

There are three types of errors that can occur in a computer program: syntax, semantic, and logic. Syntax errors are usually caught upon compiling the code and are errors in grammar, punctuation, and the likes. Semantic and logic errors are usually caught after runtime. [4]

B. Statement of the Problem

Compiler is not helpful all of the time. It may give misleading messages regarding the error. This can cause confusion to novice programmers. [5] Too

often, compile error messages are cryptic, long, or hard to understand even for experienced programmers. Shneiderman pointed out that “phrasing of error messages or diagnostic warning is critical“ and “can significantly affect user performance and satisfaction“. [6]

Though certain compilers may flag correctly some of the most common mistakes, more often than not, these Java error messages are so cryptic for new students of the language that they have a hard time identifying the errors, let alone applying fixes. Syntax errors make a program incomprehensible to compilers and are then easily pointed out. While compilers detect the obvious syntax errors, their error messages do not necessarily point the students in the right direction needed to fix the code.[7]

Compiler design and constructions research seems to have neglected the area of compiler error messages. The side effect of poorly designed error messages is that they lead programmers to take actions in the form of source code editing trying to fix it without actual effort to understand the case. On the other hand, well-designed error messages help programmers take actions to fix the issue while also helping them understand the cause. [6]

Using the five errors identified in “Methods and Tools for Exploring Novice Compilation Behaviour“, we need to develop an extension of the IDE to help novice programmers understand the compile errors. This way, the errors are described clearly, real causes are pointed out, and better error messages are generated. This is

useful during the early phase of learning Java programming as they become more proficient and knowledgeable with the language.

Compile error messages will help students to clarify concepts, misconceptions, or improve their mental models. [6]

C. Objectives of the Study

The goal of this project is to create a debugging tutor that addresses the needs of novice programmers. In particular, help them become better programmers, in terms of debugging and writing error-free programs, by improving their program comprehension skills and giving them debugging experience. The specific objectives are the following:

1. Determine the compile error
2. Provide information on the error
 - a. Background
 - b. Description
 - c. Suggestion

D. Significance of the Study

Debugging is an integral and time-consuming phase of software development. [9] Computing curricula rarely provide formal debugging training. Novice programmers are then left to develop their own skills. [3] When they do, they develop debugging skills with limited abilities in formulating ideas about the possible bugs in their code. Therefore, it seems logical to start them early. In that

case, those who are trained early in debugging would become better debuggers more quickly.

This project is will be useful for students in introductory Java classes. This improves their program comprehension ability and give them debugging experience. The error messages and analysis could be used to improve debugging in other similar languages.

E. Scope and Limitations

1. The project focuses on the following errors:

1.1 Brace expected

1.1.1 parameter list in class

1.1.2 invalid class name

1.1.3 placed a throws Exception in class

1.1.4 bracket used in opening the class code block

1.1.5 extra semicolon after class name declaration

1.2 semicolon expected

1.2.1 extra) in method call

1.2.2 colon instead of a semicolon

1.2.3 comma instead of a period to call method

1.2.4 used ++ instead of + for addition

1.2.5 no * for multiplication

1.3 Can't find symbol

1.3.1 extra equal sign in method call

1.3.2 undeclared variable

- 1.3.3 misspelled method name in the method call
- 1.3.4 package not imported
- 1.3.5 misspelled data type
- 1.4 Parenthesis expected
 - 1.4.1 return type in a void method
 - 1.4.2 used void as a type for a variable
 - 1.4.3 invalid method name
 - 1.4.4 conditions declared without parentheses
 - 1.4.5 if statement without conditions
- 1.5 . Incompatible types
 - 1.5.1 did not use boolean in if statement
 - 1.5.2 assigned void to a variable
 - 1.5.3 unfinished assignment statement
 - 1.5.4 wrong return type
 - 1.5.5 assigning null to a primitive
- 2. Detection is limited to compile errors and does not tackle runtime and logic errors.
- 3. Other types of errors such as illegal start of expression, class or interface expected, missing return, method application error, private access violation are not be covered.

F. Assumptions

- 1. The user uses BlueJ as the IDE for Java.
- 2. NAJI is installed as an extension in BlueJ.

II. REVIEW OF RELATED LITERATURE

A study made by Lee and Wu has findings on improving programming skills of novice programmers by the way they debug. They developed a debugging training which will uncover and correct any misconceptions of the programmers and improve their debugging skills. The model they developed called DebugIt covers frequently committed errors in Pascal language. One-way Analysis of Covariance was done to test the achievements. The group exposed to DebugIt scored higher. The results showed that the model of supervised debugging was effective in improving novice programmers' debugging skills. [3]

User requests and research continue to push IDEs to improve their recommendation engines. Bruch et al. Extended Eclipse's auto-complete by taking historical data and previous users' habits into account. This showed that it's possible to improve auto-complete by reordering and assigning a confidence value to the suggestions. Hou and Pletcher used historical data by performing type-hierarchy filter and grouping suggestions by functional roles. [10]

Smith and Webb also did a study called "Transparency Debugging with Explanations for Novice Programmers". They made a debugging assistant that provides the users with explicit models of their programs and hence encourage them to find errors for themselves. The transparency debugger called Bradman was created to help novice programmers debug their C programs. This way it also provides an active support during the debugging process. They have demonstrated novices appreciate having such information made explicit and that a facility that explains individual statements supports them in their debugging endeavours. [11]

Espresso was done in Bryn Mawr College. It is an educational tool for Java programming. The tool specifically does not eliminate the need for understandable compiler error messages; rather, the tool enhances the functions of a compiler. The intention was to create a helpful interactive tool that would do a better job generating error messages than existing compilers and also provide suggestions on how to fix the code. [7]

Csallner and Smaragdakis developed Check 'n Crash for Java. It is used in generation of test cases since testing is the predominant way of discovering bugs. The programmer can apply the tool to newly written code, inspect reports of conditions indicating possible crashes, and possibly update the code if the error condition is possible. It is not a bug pattern matcher. It only has basic preconceived notion of what the program text of a bug would look like. [12]

PROUST is a debugging system for Pascal that tries to understand the programmer's intention. It attempts to compare the programmer's intention with the program's design. PROUST generates a hypothesis about the user's intention and matches these against the code; it then explains this to the user. [13] The problem with this approach of error detection is that it relies on being provided with the program's design, thus it might have a higher accuracy and be able to detect semantic as well as logic errors, but it would not be able to handle any source code it is given.

Adil (Automated Debugger in Learning system) is a knowledge-based automated debugger in C language. Stereotyped code and bugs are stored as library of plans in the knowledge-base. Adil is able to understand an error-free program and locate, pinpoint, and explain logical errors. It also acts as an IDE by having necessary supporting tools to

facilitate the recognition and debugging. Given a syntax error-free program and its specification, this debugger is able to locate, pinpoint and explain logical errors of programs. [14]

HelpMeOut is a social recommender system that aids the debugging of errors by suggesting fixes that peers have applied in the past. It collects examples of code changes that fix errors in a central database. The user feeds the error to a suggestion interface then it queries the database for relevant fixes. The system is able to suggest useful fixes for 47% of the errors. [15]

Jade is a model for Java programs that uses model-based diagnosis as framework. It allows to localize certain bug occurrences in Java programs by using knowledge about incorrect outputs (or incorrect parts of the output), specified in terms of observations of incorrect values. This approach provides flexibility of errors involving multiple variables. [16]

Eclipse Guard implements relative debugging where in assertions are the key construct used by a relative debugger. An assertion defines the names of two data structures that are to be compared in each code and the locations in the two programs at which comparisons are to occur. [17]

Java Intelligent Tutoring System was a prototype developed to aid in tutoring the language. It focuses on variables, operators, and looping structures. It is a web-based application where you will upload and run your program and returns the output. [18]

A research was done on compilation behaviour by Jadud. They observed novice programmers learning Java and collected their compilations. The work involved the

development of three tools for the study : code browser to read compilations of a program, visualization that captures the type and frequency of syntax errors in a programming session, and algorithm to which they can score sessions and quantitatively compare one session against another. [8]

This paper presents a program animation system, PlanAni, that is based on the concept of the roles of variables. Roles represent schematic uses of variables that occur in programs over and over again, and a set of nine roles covers practically all variables in novice-level programs. PlanAni has been tested in a teaching experiment comparing traditional teaching with role-based teaching and animation. The results of a semi-structured interview with the teacher indicate that students like to work with the animator and that the system clarifies many concepts in programming. [19]

The algorithm is divided into blocks, and a description is given for each block. The system contains a separate window for code, flowchart, animation, explanations, and control. Assignments are based on the flowchart and on the coverage conditions of path testing. Path testing is expected to lead into more accurate evaluation of learning outcomes because it supports systematic instruction in addition to more free trial-and-error heuristics. A qualitative analysis of preliminary experiences with the prototype indicates that the approach helps a student to reflect on her own reasoning about the algorithm. However, a prerequisite for an successful learning process with the environment is a motivating introduction, describing both the system and the main idea of the algorithm to be learned. [20]

The key feature of Jeliot is the fully or semi-automatic visualization of the data and control flows. The development process of Jeliot has been research-oriented, meaning that all the different versions have had their own research agenda rising from the design of the previous version and their empirical evaluations. In this process, the user interface and visualization has evolved to better suit the targeted audience, which in the case of Jeliot 3, is novice programmers. In this paper we explain the model for the system and introduce the features of the user interface and visualization engine. Moreover, we have developed an intermediate language that is used to decouple the interpretation of the program from its visualization. This has led to a modular design that permits both internal and external extensibility. [21]

III. THEORETICAL FRAMEWORK

Taxonomy of novice programmer difficulties

Ebrahimi et al. developed a taxonomy of novice programmer difficulties in order to better understand the obstacles novice programmers encounter and so that we can develop solutions to these problems. [22] The difficulties generally fall under language construct misconceptions, plan composition errors, programming environments, and inability to get help.

Language construct misconceptions refer to misunderstandings of how the language works. For example, naming a variable “case“ which is a keyword in Java and C. Plan composition errors refer to problems in how the student plans to solve the problem. An example forgetting to increment a counter in a loop. Programming environment refers to difficulties using the IDE, the computer, and the surroundings in general. Inability to get help is simply that nobody to help or guide the student with something he or she does not understand. [22]

Error Detection

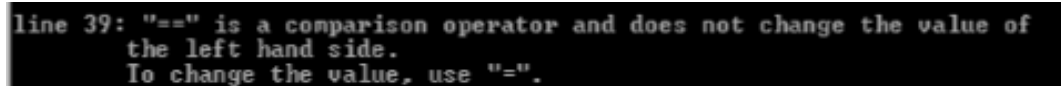
One strategy for helping novice programmers is through the detection of errors and help regarding those errors. This will ease language construct misconceptions on the part of the students.

Methods to detect errors There are generally two methods to analyze source code for errors: static and dynamic analysis. Static analysis involves parsing code without actually running it, while dynamic analysis involves testing a program with input data sets.

The QUT framework developed in Queensland University of Technology was designed to be a configurable and extensible framework that can automatically assess a student's work through static analysis and software metrics. [23] Static analysis is done by parsing the student's code into an abstract syntax tree and its structure is compared with the model solution for the given problem. Various software metrics can also be applied such as measuring the complexity of the code.

Their implementation does have its limitations. It can only evaluate small fill-in-the-blanks style programming assignments. An example would be, “Write a simple program that obtains two integer values lowerLimit and upperLimit from the user. Display all integers between lowerLimit and upperLimit in ascending order.” The student is given nearly complete program, which already handles the input. Only the loop and printing logic needs to be written. Also, their implementation can handle only compilable programs, which is not that helpful given the nature of the study.

Espresso was developed in Bryn Mawr College to overcome the problem of cryptic compiler messages. [24] The approach Espresso did is to do a better job of generating error messages and suggesting possible solutions to those errors. It is implemented as a multi-pass preprocessor. Comments are first stripped then the program is stored into memory and finally mistakes are detected.

A screenshot of a terminal window showing an error message from Espresso. The text is white on a black background. It reads: "line 39: "==" is a comparison operator and does not change the value of the left hand side. To change the value, use "=".

```
line 39: "==" is a comparison operator and does not change the value of
the left hand side.
To change the value, use "=".
```

Figure 1: Sample Output of Espresso

A sample output message of their program is shown above. The code that caused the error is “appleSauce == apple sauce;” This error is a semantic error in which the student

mistook the the equality comparison “==“ as the assignment operator “=“. the code will compile correctly but the result will not be what the student is expecting.

Top-Down Parsing

It is a strategy in which the start in the parse tree is at the highest level then down by using the rewriting rules of a formal grammar. It analyzes unknown relationships by hypothesizing general parse tree structures and then considering whether the known structures are compatible with the hypothesis. It attempts to find the left-most derivations of an input. Consumption of tokens is from left to right. It attempts to figure out the derivation for the input string, starting from the start symbol. [25]

An example of top-down parsing is recursive descent parsing. Consider the simple expression

$$x+(x+x)$$

The grammar for the expression above would be

$$E \rightarrow x+T$$

$$T \rightarrow (E)$$

$$T \rightarrow x$$

The derivation tree for the expression would be

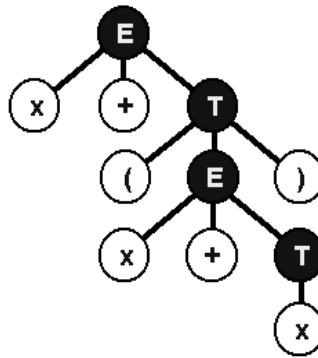


Figure 2: Derivation tree

A recursive descent parser traverses the tree by first recognizing the E. It then reads an ‘x’ and ‘+’ and then recognizes a T. It then determines whether the T had the form (E) or x then the appropriate terminals and nonterminals are recognized. [26]

Errors and subcases

The five errors considered is from “Methods and Tools for Exploring Novice Compilation Behaviour”.

“{ expected“

Java is specific about use of characters such as semicolons, brackets, or parentheses. Brackets can be more complicated since you may have to read through several blocks of codes to make sure all brackets match up with each other. Possible causes[27, 28, 29, 30]:

- parameter list in class
- invalid class name
- placed a throws Exception in class

- bracket in opening the class code block
- extra semicolon after class name declaration

“; expected“

This error is most likely thrown when a semicolon is missing at the end of line.

Possible causes[27, 28, 29, 30]:

- extra) in method call
- colon instead of a semicolon
- comma instead of a period to call method
- used ++ instead of +
- no * for multiplication

“Can't find symbol“

It is an error when the compiler doesn't have enough information to piece together what the Java code wants to run. Possible causes [27, 28, 29, 30, 31]:

- extra equal sign in method call
- undeclared variable
- misspelled method name in the method call
- package not imported
- misspelled data type

“(expected“

This error most likely shows up when there is a missing parenthesis in the code.

Possible causes[27, 28, 29, 30]:

- extra return type in a void method
- used void as a type for a variable
- invalid method name
- conditions declared without parentheses
- if statement without conditions

“Incompatible types“

It happens when the the assigned value to a variable is not of the same type. Possible causes[27, 28, 29, 30]:

- did not use boolean in if statement
- assigned void to a variable
- unfinished assignment statement
- wrong return type
- assigning null to a primitive

BlueJ

BlueJ is an IDE developed to help in learning and teaching Java. It's a top choice among introductory programming classes because of ease of use. The environment is

designed in such a way that the users would not need to spend significant time struggling with the IDE and instead focus on the programming task. It has a “code pad“ that evaluates Java code without having to write “public static void main“ every time just to evaluate expressions. Most of the functionality in other environments like Eclipse and Jbuilder is not in BlueJ. This makes BlueJ an educational environment but suitable for small-scale software development. [32] It offers an extension API that gives third parties to develop extensions to the IDE. Extensions add functionalities not in the core system. [33]

In NetBeans and Eclipse, programmers spend time looking at lines of code. If all students see are lines of code, they will think about lines of code. But the developers of BlueJ aim to make the programmers see classes and objects first; method calls, interactions, and so on. [34]

IV. DESIGN AND IMPLEMENTATION

A. Context Diagram

The application is implemented as defined in the succeeding diagrams. Generally, the tool's input and output requirements are defined in the context diagram below.

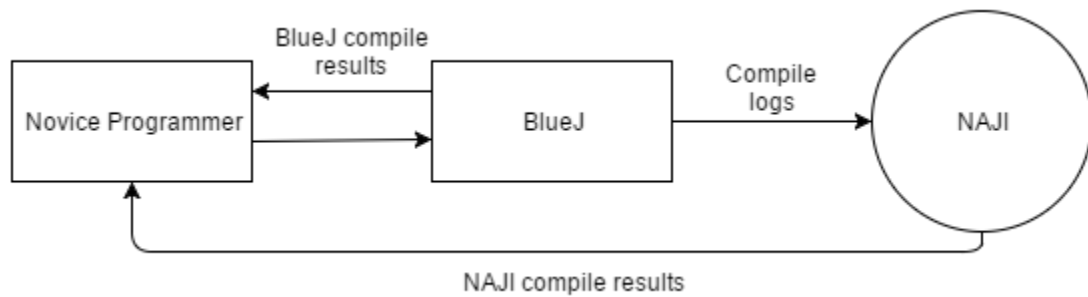


Figure 3: Context Diagram, NAJI

The extension gathers compile logs from the user's compile of the Java program in BlueJ. It then analyzes the logs and returns the error analysis. If there are errors, it returns error messages with the proper identification of the root cause and some suggestions and examples on how to fix it.

B. Use-Case Diagram

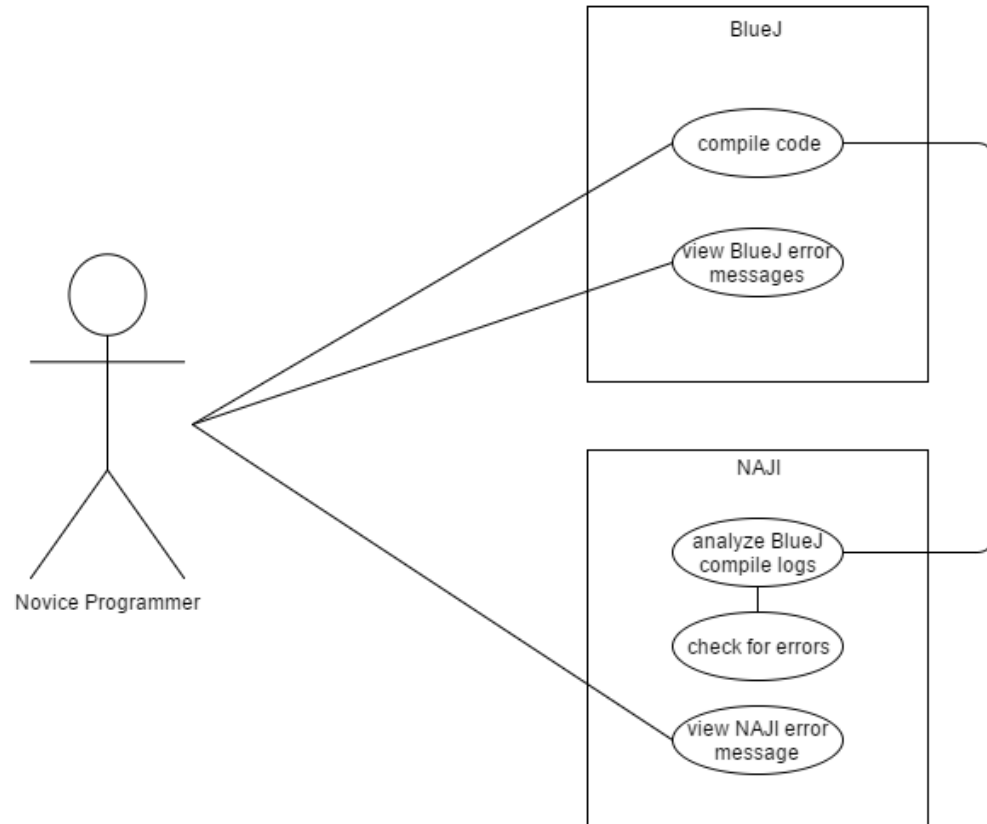


Figure 4: Use-Case Diagram, NAJI

The Novice Assistance in Java Introduction aids the novice programmers by providing a clearer explanation of the compile errors they encounter. The programmer compiles the code and views the error messages.

Use Case Name	Analyze compile logs
Description	The plugin analyzes the compile logs from BlueJ.

Actors	
Preconditions	Compile error
Flow of Activities	<ol style="list-style-type: none"> 1. NAJI receives the compile logs from BlueJ. 2. NAJI verifies if the compile error message falls within the five implemented errors. 3. If it falls within the five, the compile logs will be processed. 4. If it doesn't fall within the five, the unhandled error in NAJI will be displayed instead.
Postconditions	Compile logs are processed for errors

Use Case Name	Check for errors
Description	The plugin checks the root cause of the error determined when the code was compiled.
Actors	
Preconditions	Compile error falls within the five defined errors
Flow of Activities	<ol style="list-style-type: none"> 1. Check for the syntax of the line that caused the compile error. 2. If it falls within the five subcases we have resolved, show the NAJI error message. 3. If it is unhandled by NAJI, show the BlueJ unhandled error message.
Postconditions	Clearer compile error is returned – root cause, samples

Use Case Name	View error message
----------------------	--------------------

Description	The extension will show a more specific error message based on the compile logs.
Actors	Novice programmer
Preconditions	Compile logs are within the five implemented errors
Flow of Activities	<ol style="list-style-type: none"> 1. A more specific compile error message is returned by NAJI. 2. The message is shown in the other window.
Postconditions	Novice programmer views the error message in another window

C. Data Flow Diagram

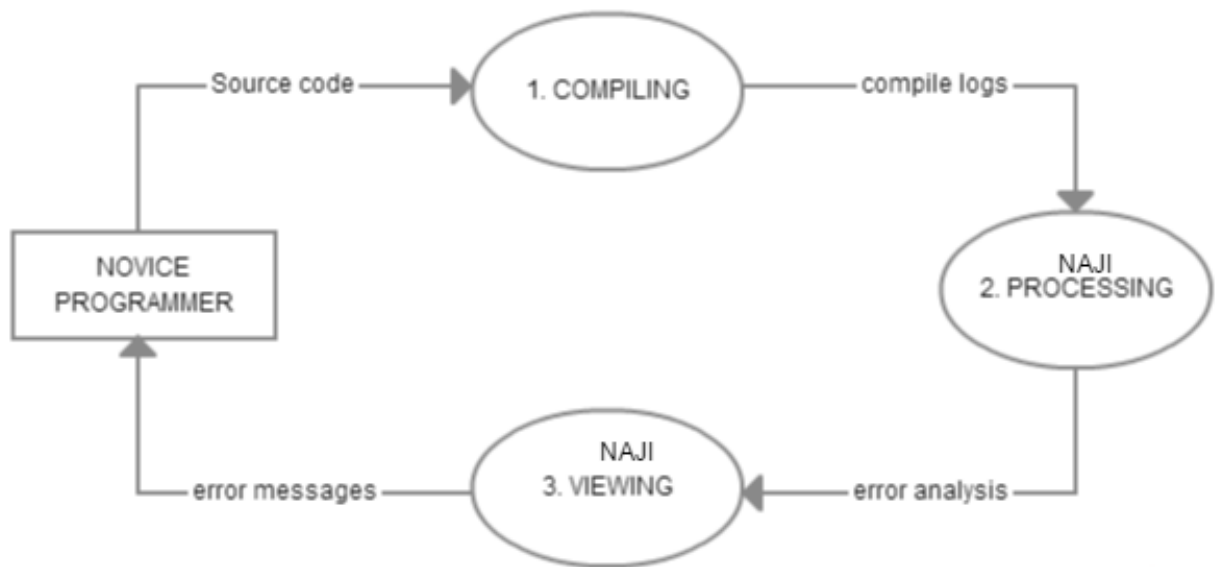


Figure 5: Data Flow Diagram, NAJI

The functionalities of the system are defined in the above data flow diagram.

The novice programmer compiles the code in BlueJ. The compile logs are sent to

the processor. It then outputs the error analysis and sends it to the user interface. The novice programmer views the error messages from the processing.

D. Swimlane Flowchart

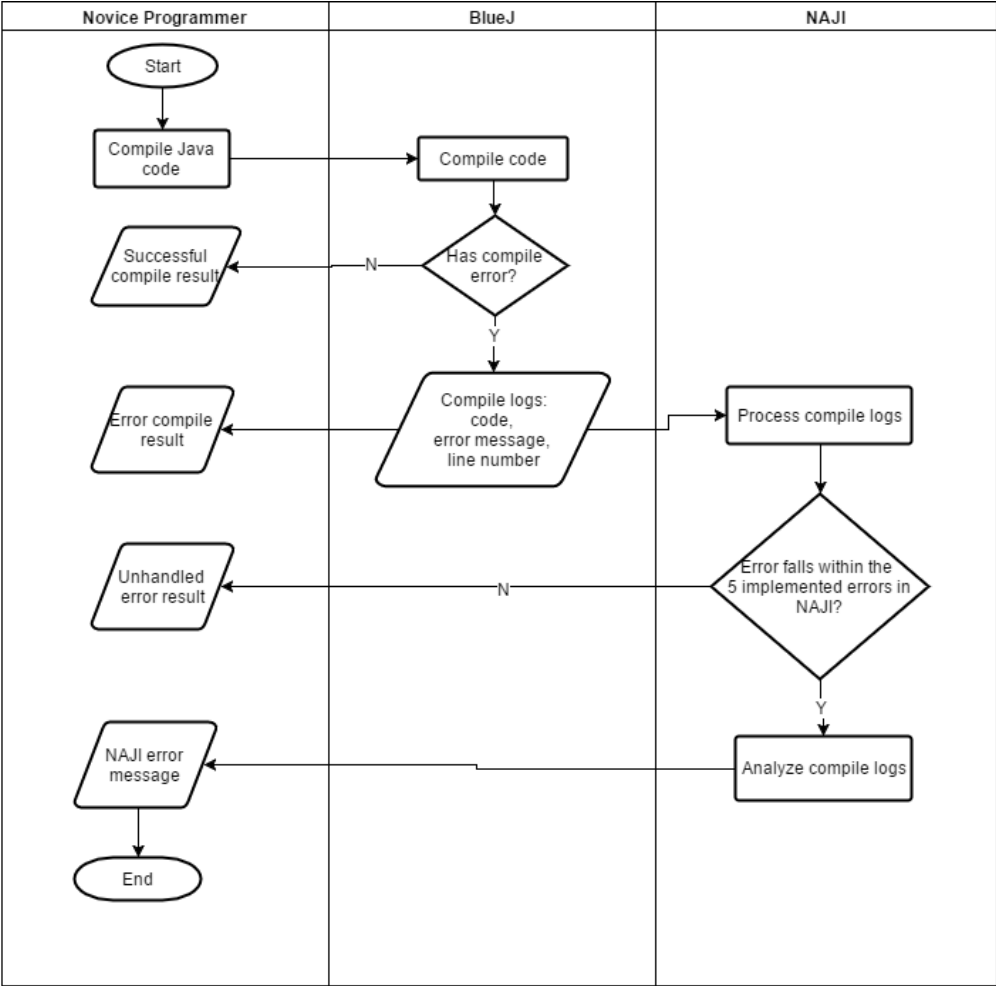


Figure 6: Flowchart with swimlanes, NAJI

The diagram above shows how the plugin classifies the compile errors from BlueJ. The user is responsible for compiling the Java code. BlueJ compiles the code submitted by the user and then shows the output. NAJI, on the other hand, processes the compile logs from BlueJ and then categorizes if it falls within the five

implemented errors. If it doesn't, it shows the NAJI unhandled error. If it is one of the five errors, NAJI analyzes the error and outputs a clearer cause of the error and some examples on how to fix it.

E. Class Diagram

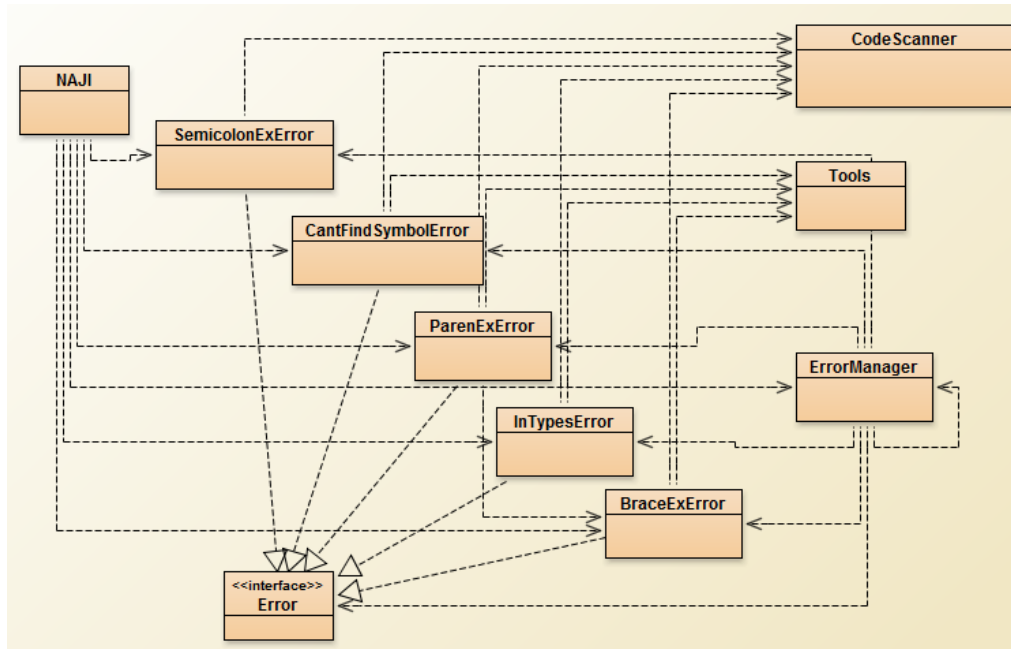


Figure 7: Class diagram, NAJI

The diagram above shows the classes in NAJI. The class NAJI passes the five error classes – SemicolonExError, CantFindSymbolError, ParenExError, InTypesError, BraceExError – to ErrorManager. Each error class checks if it accepts the compile error from BlueJ and processes it after. The five error classes use the CodeScanner class to scan through the code that caused the compile error and Tools class to do some functions.

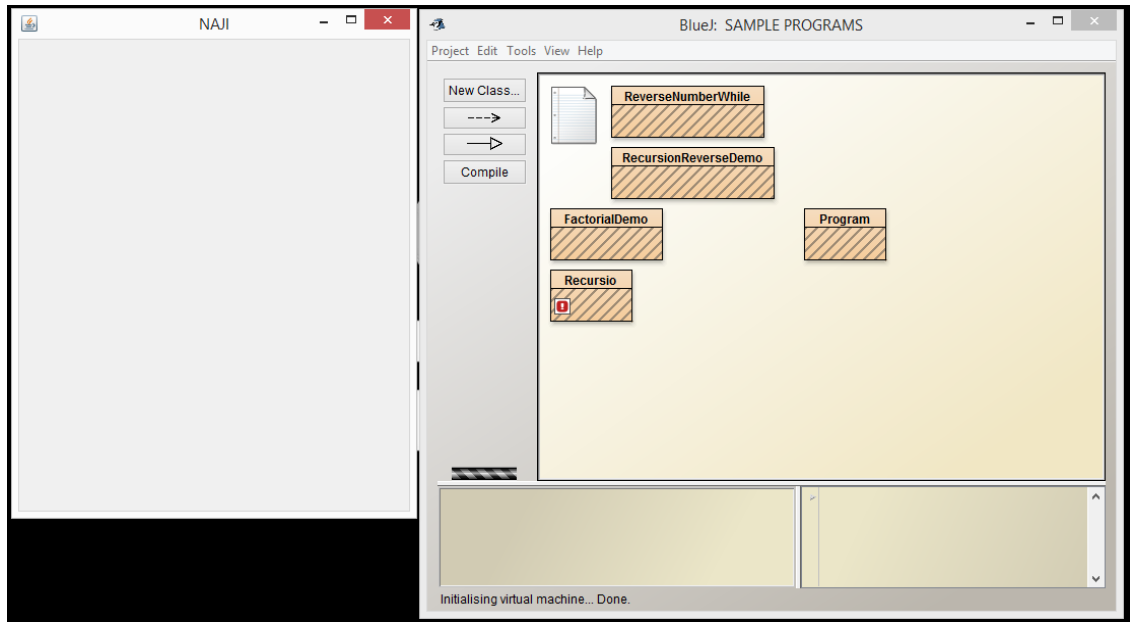
F. Technical Requirements

The extension is implemented in Java programming language. Thus a machine installed with Java Runtime Environment is required to run the it. It also needs the BlueJ IDE to run the extension.

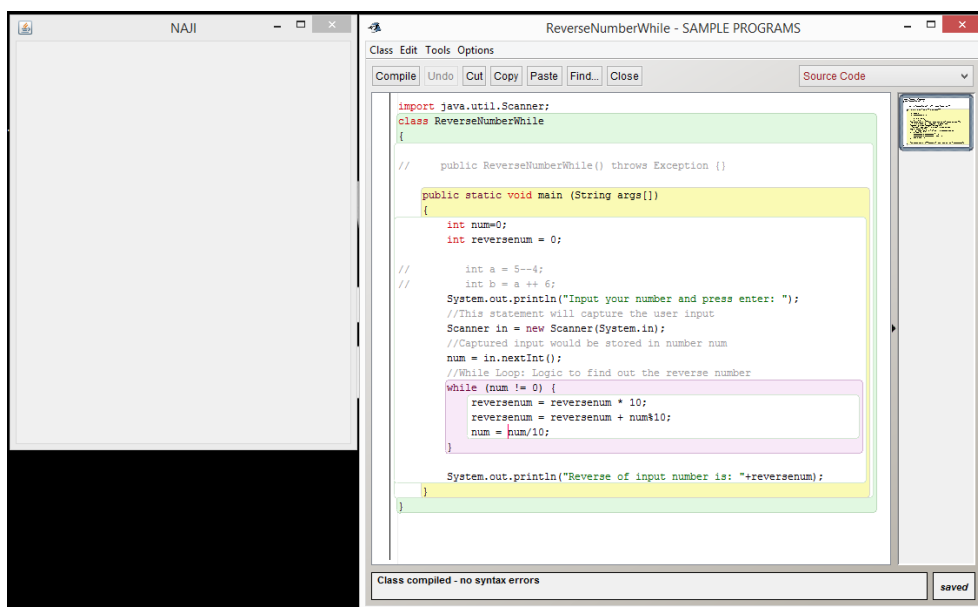
V. RESULTS

Once NAJI is installed as an extension in BlueJ, opening the IDE also loads NAJI.

The shaded classes mean these are not compiled yet.



The novice programmer opens a class and then compiles it. If there's no compile error, BlueJ shows the following output. No message is shown in the NAJI window.



The novice programmer is able to view the more specific error message in the NAJI window which contains the root cause of the issue, suggestion, and examples on how to fix it.

A. Brace expected errors

1. Parameter list in class

A parameter list '()' is in the class declaration. This throws the brace expected error.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class Details () {

    public void countDupChars(String str){

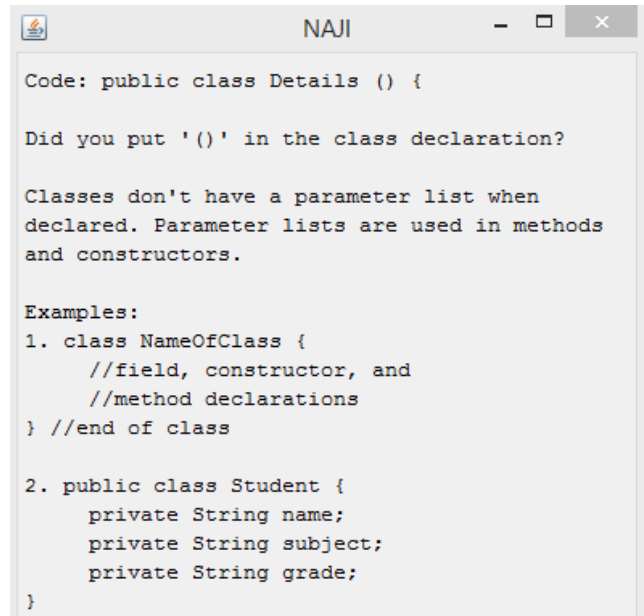
        //Create a HashMap
        Map<Character, Integer> map = new HashMap<Character, Integer>();

        //Convert the String to char array
        char[] chars = str.toCharArray();

        /* logic: char are inserted as keys and their count
        * as values. If map contains the char already then
        * increase the value by 1
        */
        for(Character ch:chars){
            if(map.containsKey(ch)){
```

'{' expected

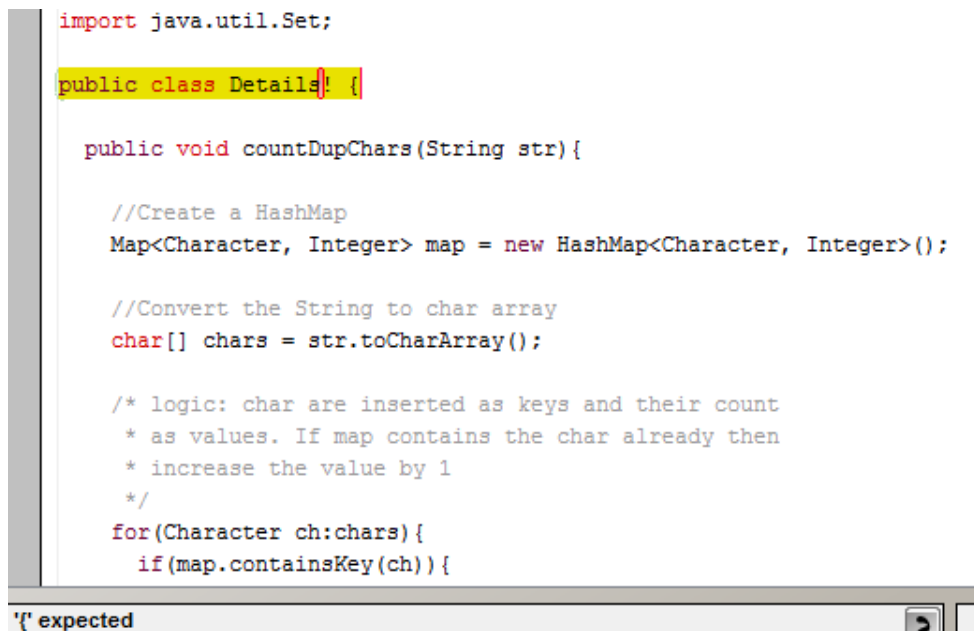
The output message by NAJI regarding the error is shown on the window.



```
Code: public class Details () {
Did you put '()' in the class declaration?
Classes don't have a parameter list when
declared. Parameter lists are used in methods
and constructors.
Examples:
1. class NameOfClass {
    //field, constructor, and
    //method declarations
} //end of class
2. public class Student {
    private String name;
    private String subject;
    private String grade;
}
```

2. Invalid class name

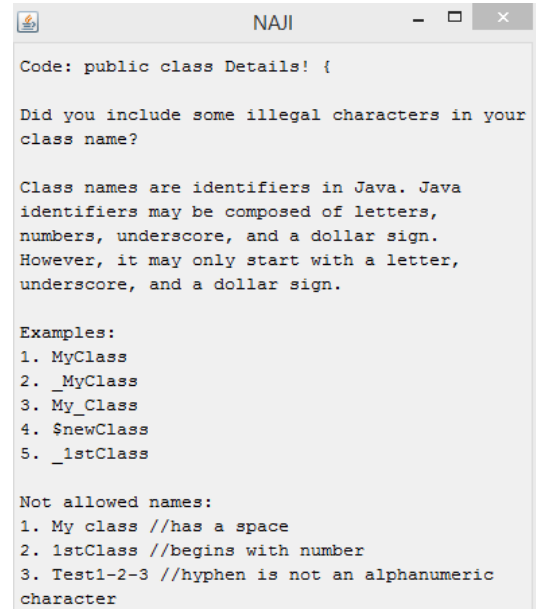
If the novice programmer adds some illegal characters in naming the class, brace expected error is thrown by BlueJ.



```
import java.util.Set;
public class Details! {
    public void countDupChars(String str){
        //Create a HashMap
        Map<Character, Integer> map = new HashMap<Character, Integer>();
        //Convert the String to char array
        char[] chars = str.toCharArray();
        /* logic: char are inserted as keys and their count
        * as values. If map contains the char already then
        * increase the value by 1
        */
        for(Character ch:chars){
            if(map.containsKey(ch)){
```

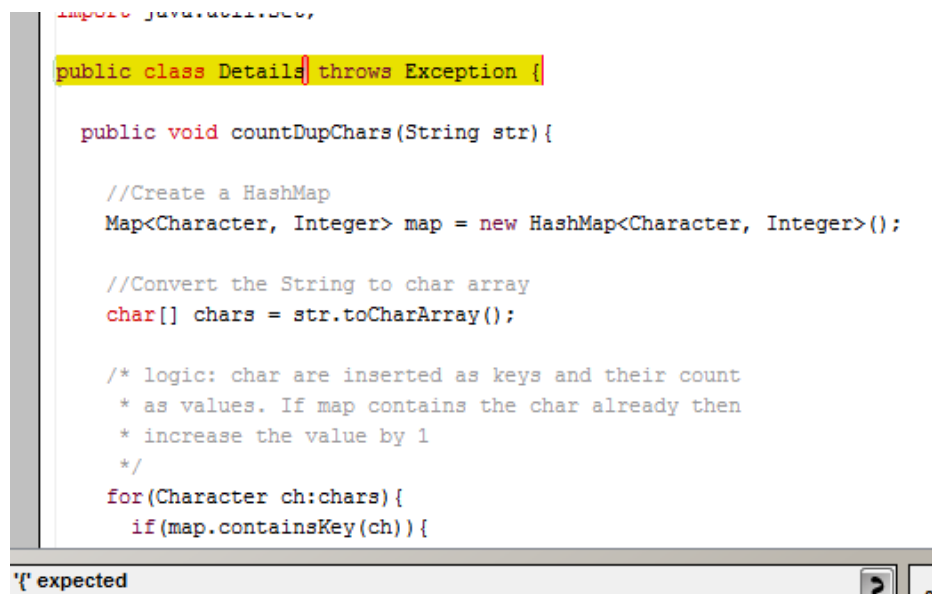
'{' expected

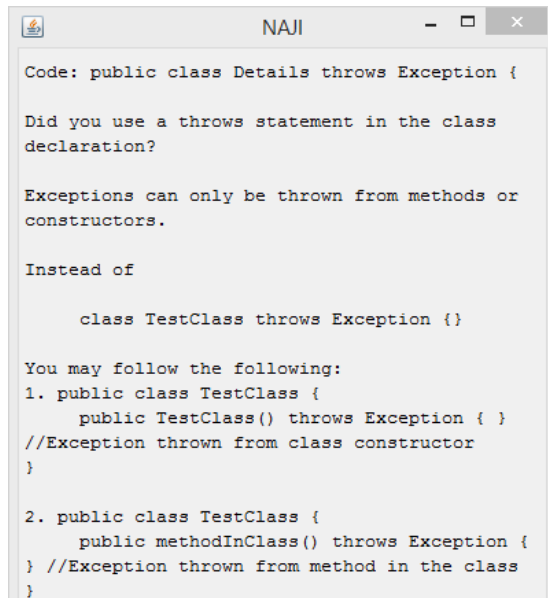
NAJI shows the more specific cause of the error.



3. Placed a throws Exception in class

Brace expected error is also thrown when a throws statement is added in the class declaration.





Code: public class Details throws Exception {

Did you use a throws statement in the class declaration?

Exceptions can only be thrown from methods or constructors.

Instead of

```
class TestClass throws Exception {
```

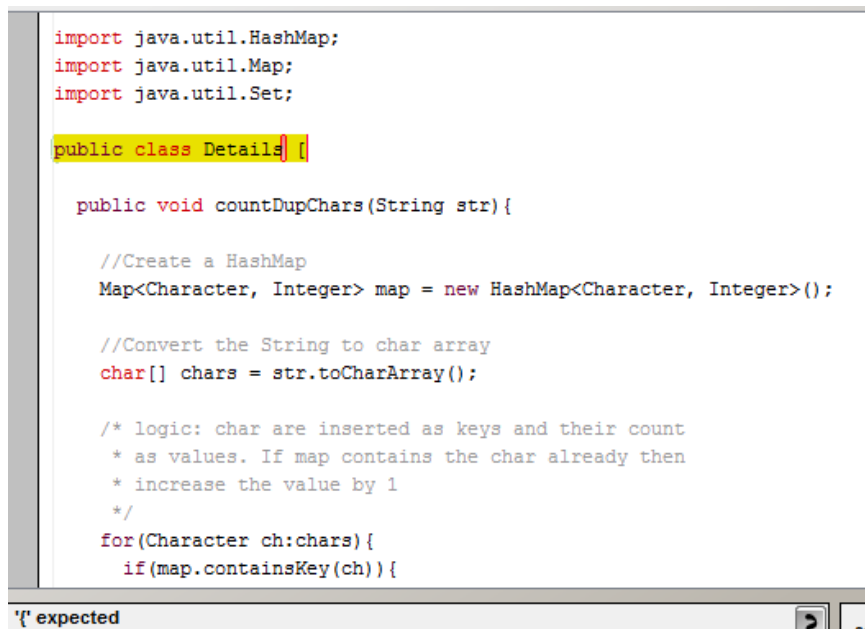
You may follow the following:

1. public class TestClass {
 public TestClass() throws Exception { }
 //Exception thrown from class constructor
}
2. public class TestClass {
 public methodInClass() throws Exception {
} //Exception thrown from method in the class
}

NAJI shows the following output.

4. Bracket in opening the class code block

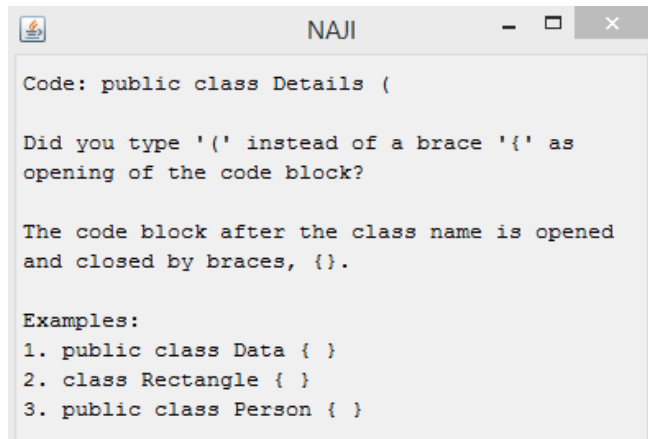
When an opening bracket is used to start a code block, bracket expected error is returned.



```
import java.util.HashMap;  
import java.util.Map;  
import java.util.Set;  
  
public class Details {  
  
    public void countDupChars(String str){  
  
        //Create a HashMap  
        Map<Character, Integer> map = new HashMap<Character, Integer>();  
  
        //Convert the String to char array  
        char[] chars = str.toCharArray();  
  
        /* logic: char are inserted as keys and their count  
        * as values. If map contains the char already then  
        * increase the value by 1  
        */  
        for(Character ch:chars){  
            if(map.containsKey(ch)){
```

'{' expected

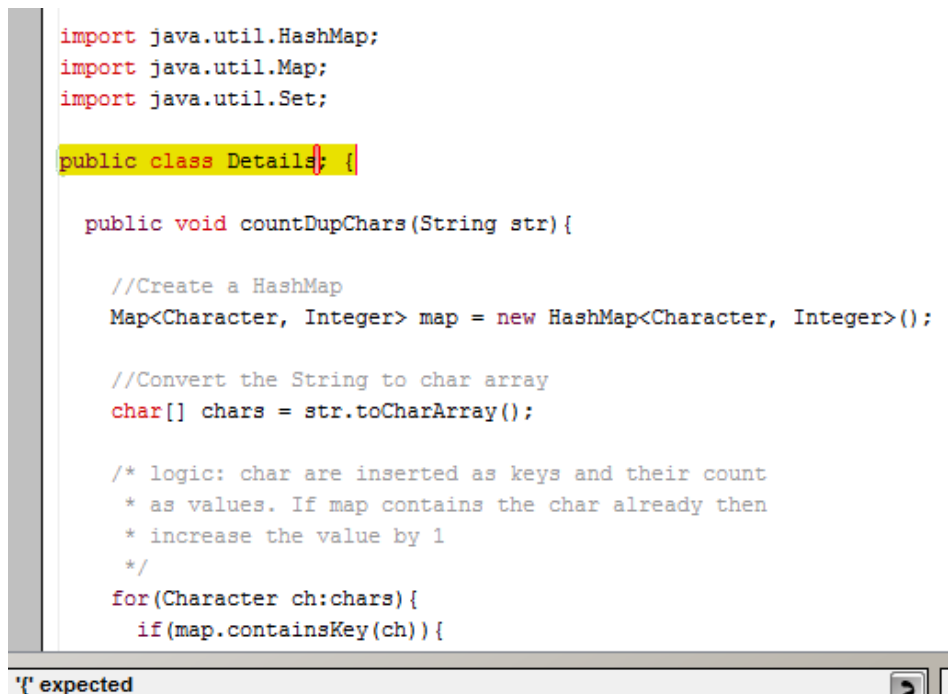
NAJI shows the following output for the error.



```
Code: public class Details (  
  
Did you type '(' instead of a brace '{' as  
opening of the code block?  
  
The code block after the class name is opened  
and closed by braces, {}.  
  
Examples:  
1. public class Data { }  
2. class Rectangle { }  
3. public class Person { }
```

5. Extra semicolon after class name declaration

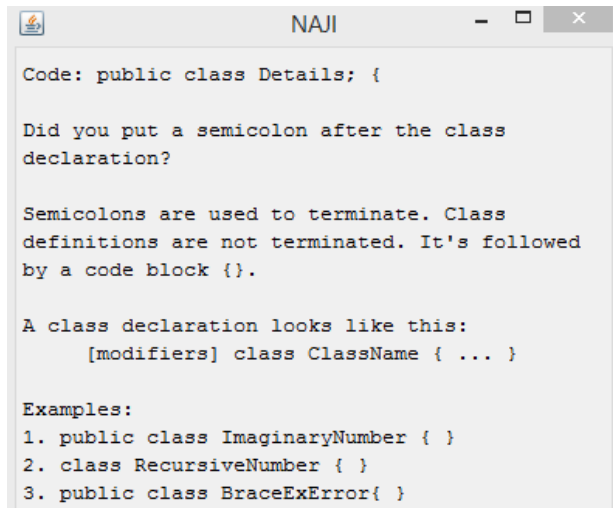
Lastly, when a semicolon is added in the class declaration, it throws a brace expected error.



```
import java.util.HashMap;  
import java.util.Map;  
import java.util.Set;  
  
public class Details; {  
  
    public void countDupChars(String str) {  
  
        //Create a HashMap  
        Map<Character, Integer> map = new HashMap<Character, Integer>();  
  
        //Convert the String to char array  
        char[] chars = str.toCharArray();  
  
        /* logic: char are inserted as keys and their count  
        * as values. If map contains the char already then  
        * increase the value by 1  
        */  
        for(Character ch:chars){  
            if(map.containsKey(ch)){
```

'}' expected

However, NAJI shows the more specific message below.



```
Code: public class Details; {

Did you put a semicolon after the class
declaration?

Semicolons are used to terminate. Class
definitions are not terminated. It's followed
by a code block {}.

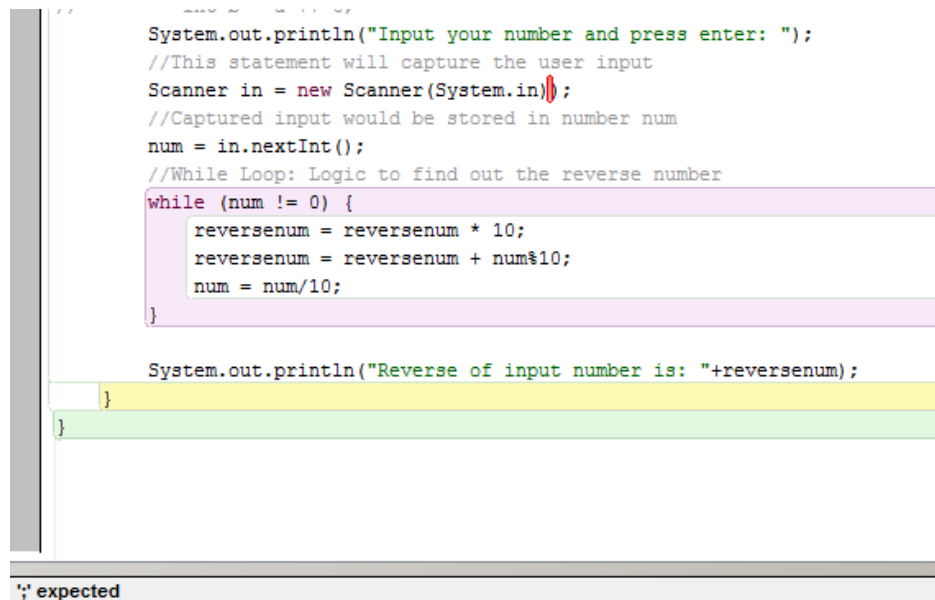
A class declaration looks like this:
    [modifiers] class ClassName { ... }

Examples:
1. public class ImaginaryNumber { }
2. class RecursiveNumber { }
3. public class BraceExError{ }
```

B. Semicolon expected errors

1. Extra ‘)’ in the method call

Semicolon expected is the compile error thrown by BlueJ when there's an extra parenthesis in the calling of a method of a class.

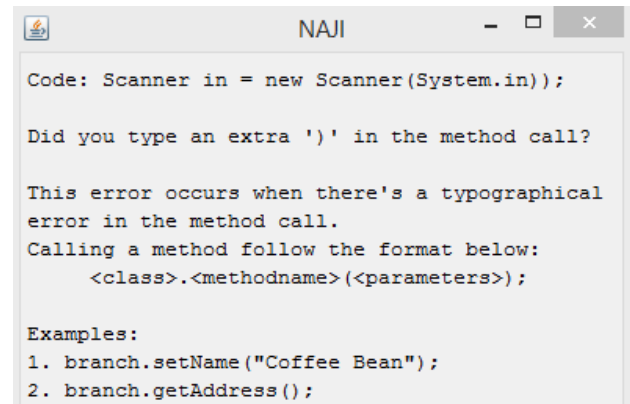


```
//
System.out.println("Input your number and press enter: ");
//This statement will capture the user input
Scanner in = new Scanner(System.in));
//Captured input would be stored in number num
num = in.nextInt();
//While Loop: Logic to find out the reverse number
while (num != 0) {
    reversenum = reversenum * 10;
    reversenum = reversenum + num%10;
    num = num/10;
}

System.out.println("Reverse of input number is: "+reversenum);
}
}
```

' expected

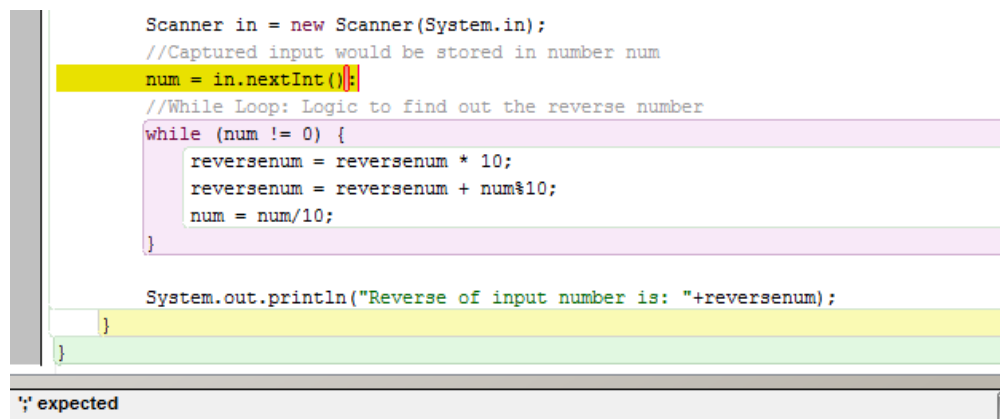
The output message shown by NAJI is on the left.



```
Code: Scanner in = new Scanner(System.in);
Did you type an extra ')' in the method call?
This error occurs when there's a typographical error in the method call.
Calling a method follow the format below:
    <class>.<methodname>(<parameters>);
Examples:
1. branch.setName("Coffee Bean");
2. branch.getAddress();
```

2. Colon instead of a semicolon

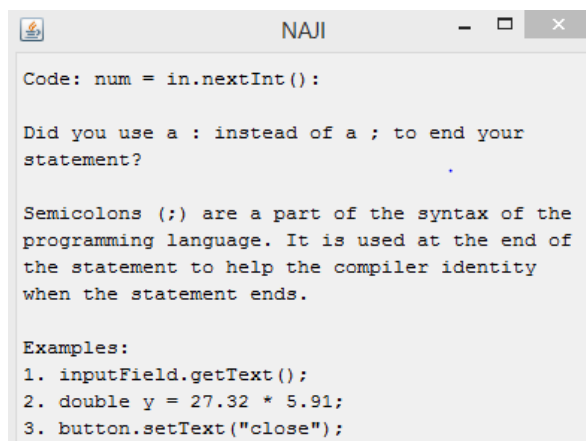
If a colon was used instead of a semicolon to terminate a statement, semicolon expected error is thrown.



```
Scanner in = new Scanner(System.in);
//Captured input would be stored in number num
num = in.nextInt():
//While Loop: Logic to find out the reverse number
while (num != 0) {
    reversenum = reversenum * 10;
    reversenum = reversenum + num%10;
    num = num/10;
}
System.out.println("Reverse of input number is: "+reversenum);
}
```

';' expected

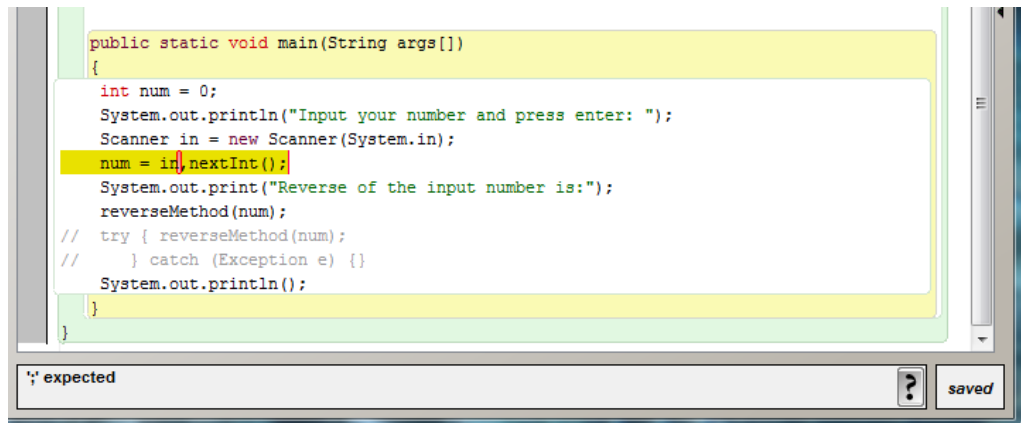
NAJI shows the specific message below.



```
Code: num = in.nextInt():
Did you use a : instead of a ; to end your statement?
Semicolons (;) are a part of the syntax of the programming language. It is used at the end of the statement to help the compiler identify when the statement ends.
Examples:
1. inputField.getText();
2. double y = 27.32 * 5.91;
3. button.setText("close");
```

3. Comma instead of a period to call method

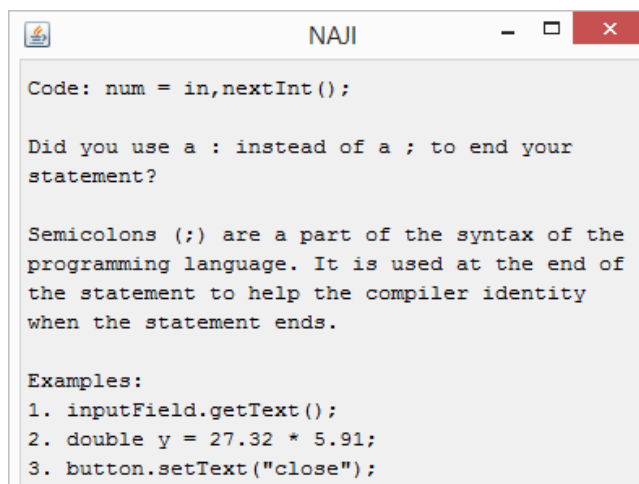
Another cause of semicolon expected error is when a comma is used instead of a period in calling a method of a class.



```
public static void main(String args[])
{
    int num = 0;
    System.out.println("Input your number and press enter: ");
    Scanner in = new Scanner(System.in);
    num = in,nextInt();
    System.out.print("Reverse of the input number is:");
    reverseMethod(num);
    // try { reverseMethod(num);
    //     } catch (Exception e) {}
    System.out.println();
}
}
```

": expected

NAJI explains the error in the following output message.



```
Code: num = in,nextInt();

Did you use a : instead of a ; to end your
statement?

Semicolons (;) are a part of the syntax of the
programming language. It is used at the end of
the statement to help the compiler identify
when the statement ends.

Examples:
1. inputField.getText();
2. double y = 27.32 * 5.91;
3. button.setText("close");
```

4. Used ++ instead of +

The user may also use ++ instead of + in adding operands. This causes a semicolon expected error.

```
{
    int num = 0;
    num = 9 ++ 4;
    System.out.println("Input your number and press enter: ");
    Scanner in = new Scanner(System.in);
    num = in.nextInt();
    System.out.print("Reverse of the input number is:");
    reverseMethod(num);
    // try { reverseMethod(num);
    // } catch (Exception e) {}
    System.out.println();
}
```

';' expected

NAJI gives the following explanation for this.

```
Code: num = 9 ++ 4;

Did you use a unary instead of a binary operator?

Unary operators require only one operand. They are used to increment or decrement a value by 1.

Examples:
1. int result = 0;
2. result++; //result is now 1
3. result--; //result is now 0

Binary operators require two operands. It follows the format : <Operand1> <operator> <Operand2>

Examples:
1. 5 + 1
2. x = y - z
3. num = a + b
```

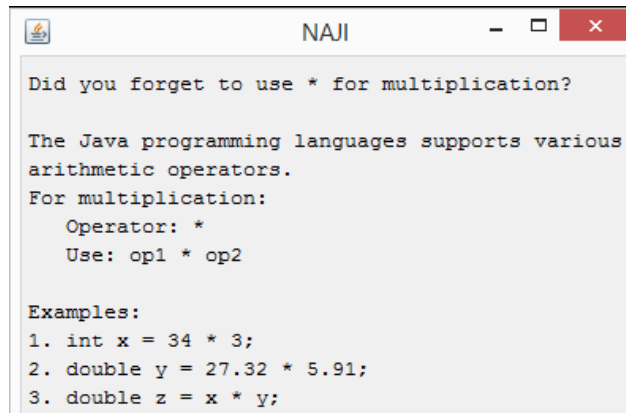
5. No * for multiplication

New programmers also commit the mistake of using x to multiply which throws a semicolon expected error.

```
int num = 0;
num = 9 x 4;
System.out.println("Input your number and press enter: ");
Scanner in = new Scanner(System.in);
num = in.nextInt();
System.out.print("Reverse of the input number is:");
reverseMethod(num);
// try { reverseMethod(num);
// } catch (Exception e) {}
System.out.println();
}
```

';' expected

NAJI, on the other hand, shows the following output.



```
Did you forget to use * for multiplication?

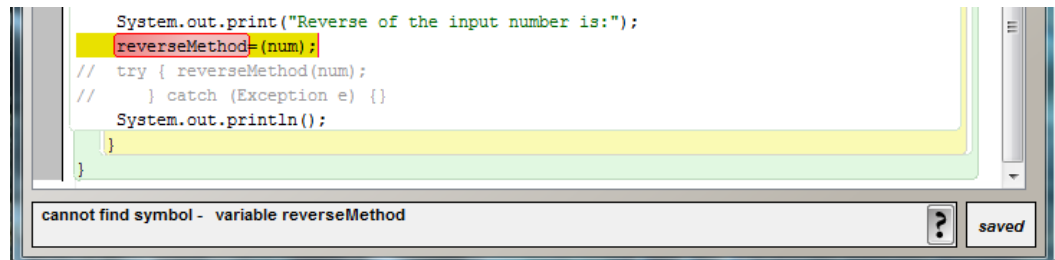
The Java programming languages supports various
arithmetic operators.
For multiplication:
  Operator: *
  Use: op1 * op2

Examples:
1. int x = 34 * 3;
2. double y = 27.32 * 5.91;
3. double z = x * y;
```

C. Can't find symbol errors

1. Extra equal sign in the method call

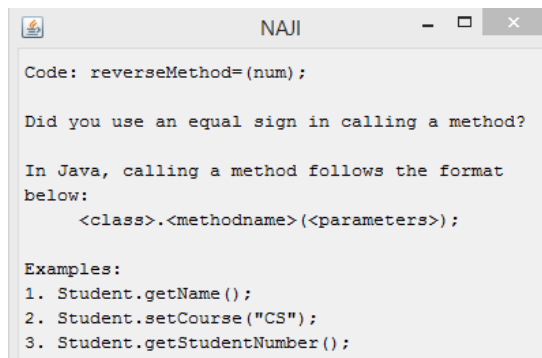
When an extra equal sign is typed in the method call, this causes the can't find symbol error.



```
System.out.print("Reverse of the input number is:");
reverseMethod=(num);
// try { reverseMethod(num);
//   } catch (Exception e) {}
System.out.println();
}
```

cannot find symbol - variable reverseMethod

On the other hand, NAJI shows the more descriptive message.



```
Code: reverseMethod=(num);

Did you use an equal sign in calling a method?

In Java, calling a method follows the format
below:
  <class>.<methodname>(<parameters>);

Examples:
1. Student.getName();
2. Student.setCourse("CS");
3. Student.getStudentNumber();
```

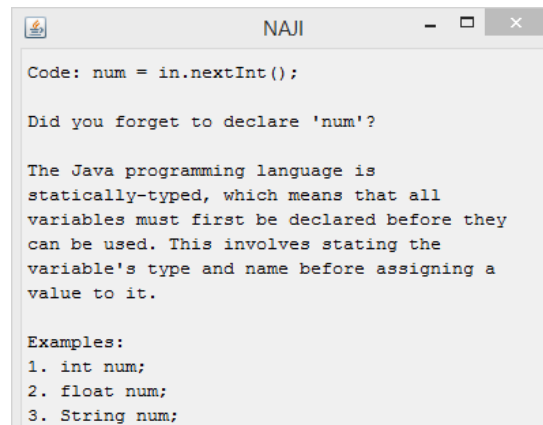

2. Undeclared variable

Can't find symbol error is thrown when there's a variable used but is not declared in the code.

```
Scanner in = new Scanner(System.in);
num = in.nextInt();
System.out.print("Reverse of the input number is:");
reverseMethod(num);
// try { reverseMethod(num);
// } catch (Exception e) {}
System.out.println();
}
```

cannot find symbol - variable num

NAJI describes the error committed in the window below.



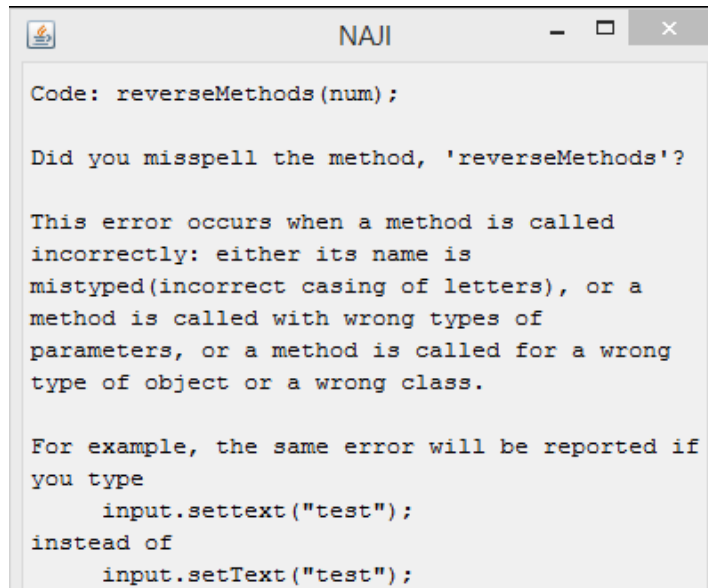
3. Misspelled method name in the method call

A method may be misspelled when called in the program. This causes the compiler to throw the can't find symbol error.

```
System.out.print("Reverse of the input number is:");
reverseMethods(num);
// try { reverseMethod(num);
// } catch (Exception e) {}
System.out.println();
}
```

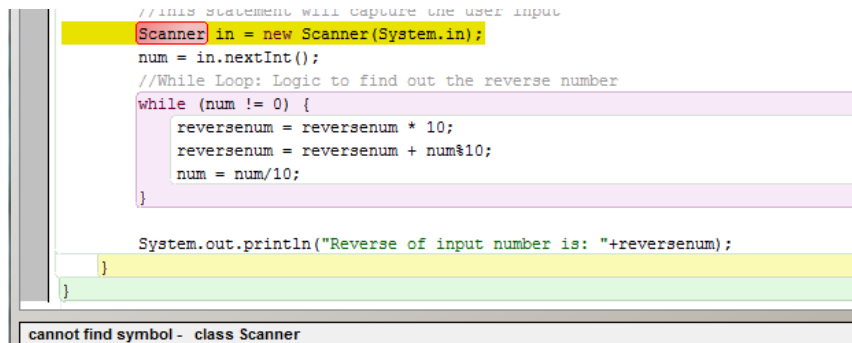
cannot find symbol - method reverseMethods(int)

NAJI describes the error committed in the window below.

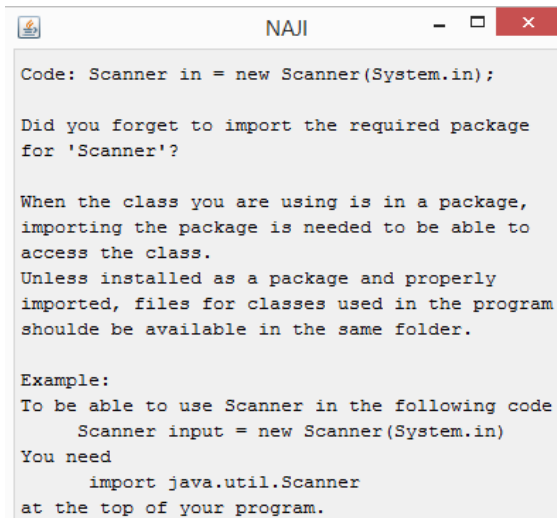


4. Package not imported

Sometimes a class may be used without importing the package where it is included. This causes the error below.

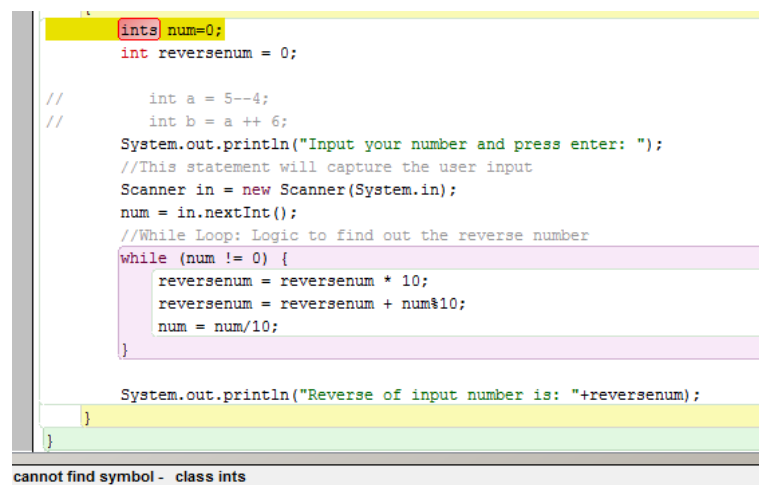


NAJI describes the error committed in the window below.

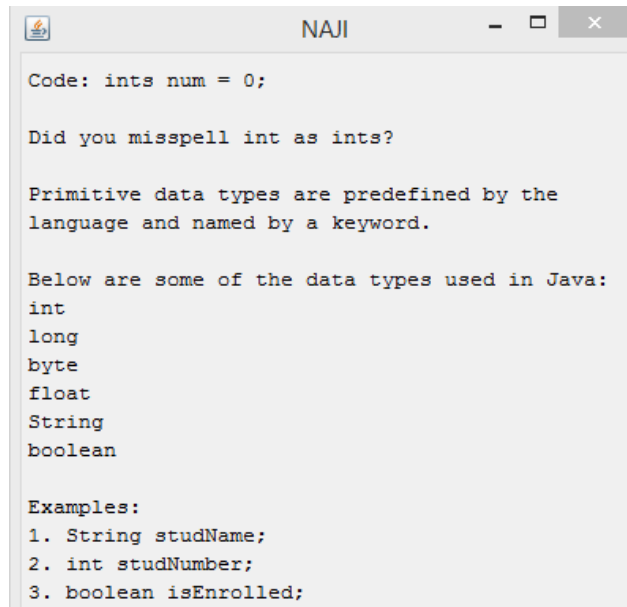


5. Misspelled data type

Another error that causes the can't find symbol compile error is when a primitive data type is misspelled in the code.



NAJI clarifies this error with the following output.



```
Code: ints num = 0;

Did you misspell int as ints?

Primitive data types are predefined by the
language and named by a keyword.

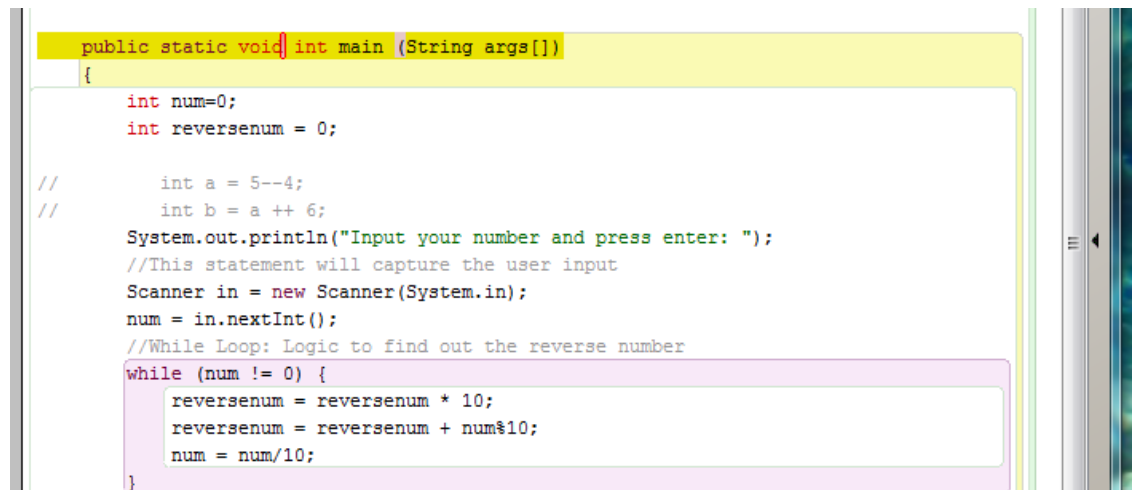
Below are some of the data types used in Java:
int
long
byte
float
String
boolean

Examples:
1. String studName;
2. int studNumber;
3. boolean isEnrolled;
```

D. Parenthesis expected errors

1. Extra return type in a void method

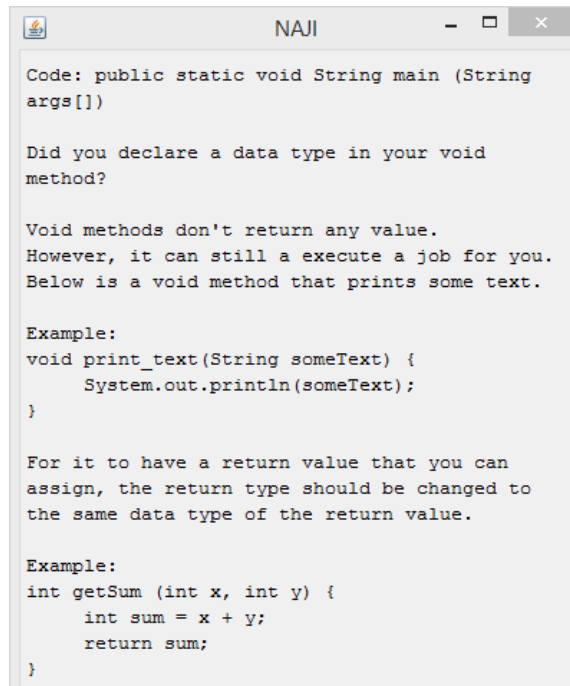
When a return type is added in declaring a void method, the compiler throws a parenthesis expected error.



```
public static void int main (String args[])
{
    int num=0;
    int reversenum = 0;

    //      int a = 5--4;
    //      int b = a ++ 6;
    System.out.println("Input your number and press enter: ");
    //This statement will capture the user input
    Scanner in = new Scanner(System.in);
    num = in.nextInt();
    //While Loop: Logic to find out the reverse number
    while (num != 0) {
        reversenum = reversenum * 10;
        reversenum = reversenum + num%10;
        num = num/10;
    }
}
```

NAJI clarifies this error with the following output.



```
Code: public static void String main (String
args[])

Did you declare a data type in your void
method?

Void methods don't return any value.
However, it can still a execute a job for you.
Below is a void method that prints some text.

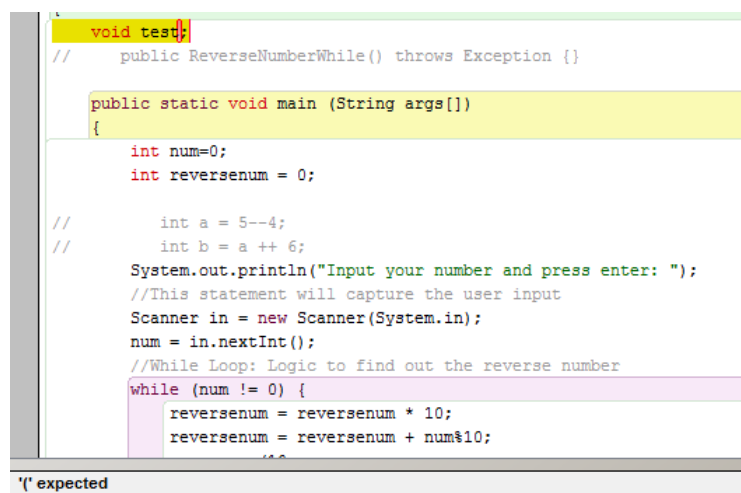
Example:
void print_text(String someText) {
    System.out.println(someText);
}

For it to have a return value that you can
assign, the return type should be changed to
the same data type of the return value.

Example:
int getSum (int x, int y) {
    int sum = x + y;
    return sum;
}
```

2. Used void as a type for a variable

When the novice programmer declares a variable as void, it throws the error below.



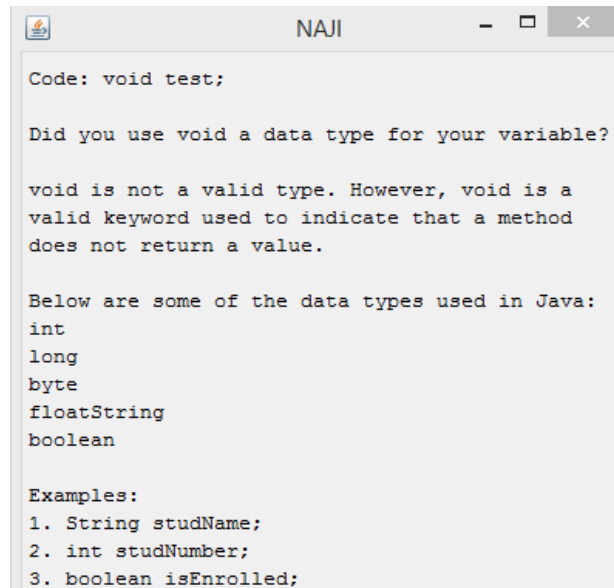
```
void test;
//    public ReverseNumberWhile() throws Exception {}

public static void main (String args[])
{
    int num=0;
    int reversenum = 0;

    //    int a = 5--4;
    //    int b = a ++ 6;
    System.out.println("Input your number and press enter: ");
    //This statement will capture the user input
    Scanner in = new Scanner(System.in);
    num = in.nextInt();
    //While Loop: Logic to find out the reverse number
    while (num != 0) {
        reversenum = reversenum * 10;
        reversenum = reversenum + num%10;
    }
}
```

' expected

NAJI clarifies this error with the following output.



```
Code: void test;

Did you use void a data type for your variable?

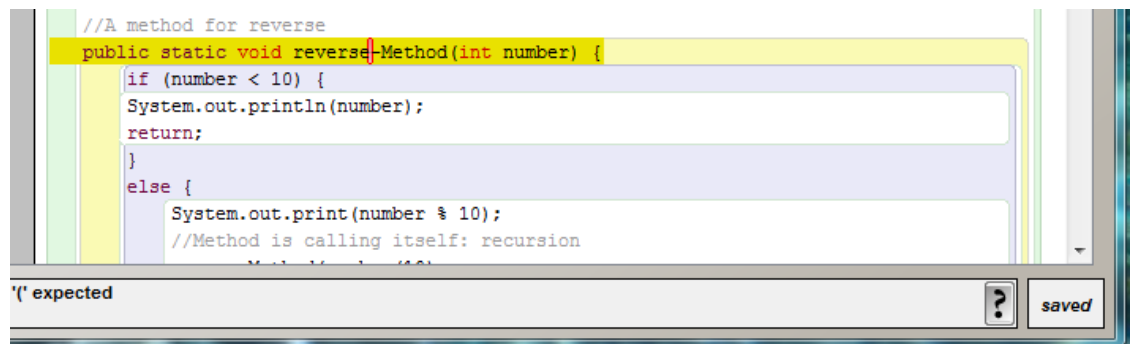
void is not a valid type. However, void is a
valid keyword used to indicate that a method
does not return a value.

Below are some of the data types used in Java:
int
long
byte
floatString
boolean

Examples:
1. String studName;
2. int studNumber;
3. boolean isEnrolled;
```

3. Invalid method name

Method names that have illegal characters throw the parenthesis expected error.



```
//A method for reverse
public static void reverse-Method(int number) {
    if (number < 10) {
        System.out.println(number);
        return;
    }
    else {
        System.out.print(number % 10);
        //Method is calling itself: recursion
        Method reverse-Method(10);
    }
}
```

'(' expected

NAJI describes the error committed in the window.

```
Code: public static void reverse Method(int number) {

Did you include some illegal characters in your method name?

Method names are identifiers in Java. Java identifiers may be composed of letters, numbers, underscore, and a dollar sign. However, it may only start with a letter, underscore, and a dollar sign.

Examples:
1. MyMethod
2. _MyMethod
3. My_Method
4. $newMethod
5. _1stMethod

Not allowed names:
1. My method //has a space
2. 1stMethod //begins with number
3. Test1-2-3 //hyphen is not an alphanumeric character
```

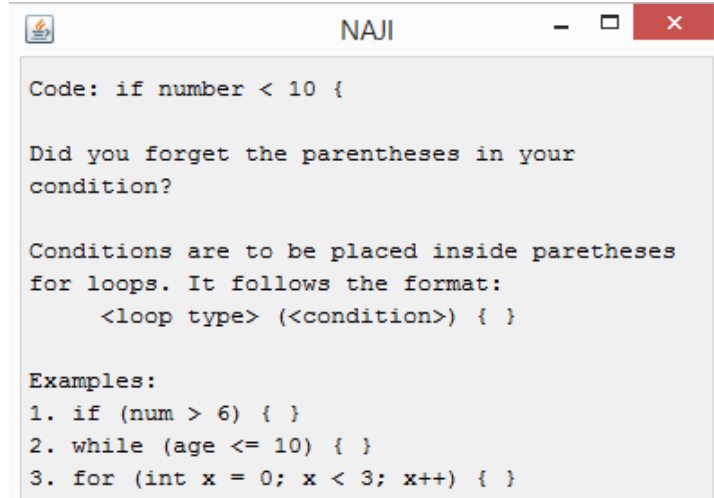
4. Conditions declared without parentheses

When a condition is declared without parentheses, the compiler detects this as parenthesis expected error.

```
public static void reverseMethod(int number) {
    if number < 10 {
        System.out.println(number);
        return;
    }
    else {
        System.out.print(number % 10);
        //Method is calling itself: recursion
        reverseMethod(number/10);
    }
}

public static void main(String args[])
{
    '(' expected
```

NAJI describes the error committed in the window.

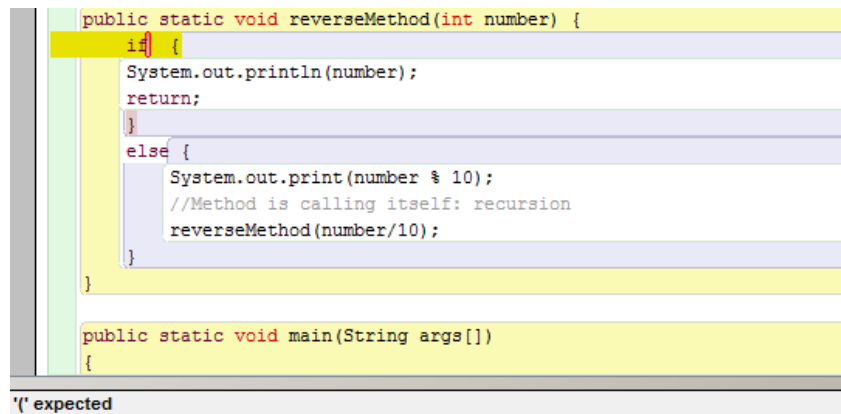


The screenshot shows a window titled "NAJI" with a standard Windows title bar (minimize, maximize, close buttons). The content of the window is as follows:

```
Code: if number < 10 {  
  
Did you forget the parentheses in your  
condition?  
  
Conditions are to be placed inside parentheses  
for loops. It follows the format:  
    <loop type> (<condition>) { }  
  
Examples:  
1. if (num > 6) { }  
2. while (age <= 10) { }  
3. for (int x = 0; x < 3; x++) { }
```

5. If statement without conditions

When an if statement is used without conditions, the compiler throws a parenthesis expected error.

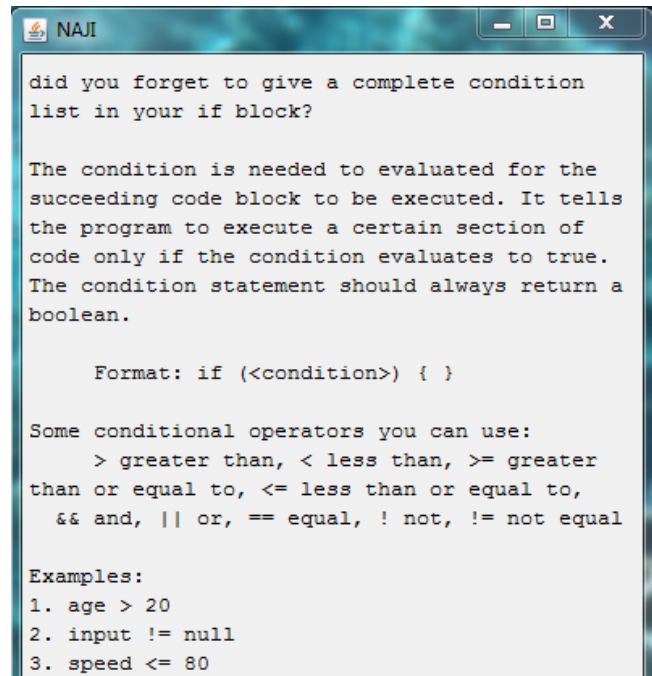


The screenshot shows a code editor with the following Java code:

```
public static void reverseMethod(int number) {  
    if {  
        System.out.println(number);  
        return;  
    }  
    else {  
        System.out.print(number % 10);  
        //Method is calling itself: recursion  
        reverseMethod(number/10);  
    }  
}  
  
public static void main(String args[])  
{
```

At the bottom of the editor, the error message "'(' expected" is visible.

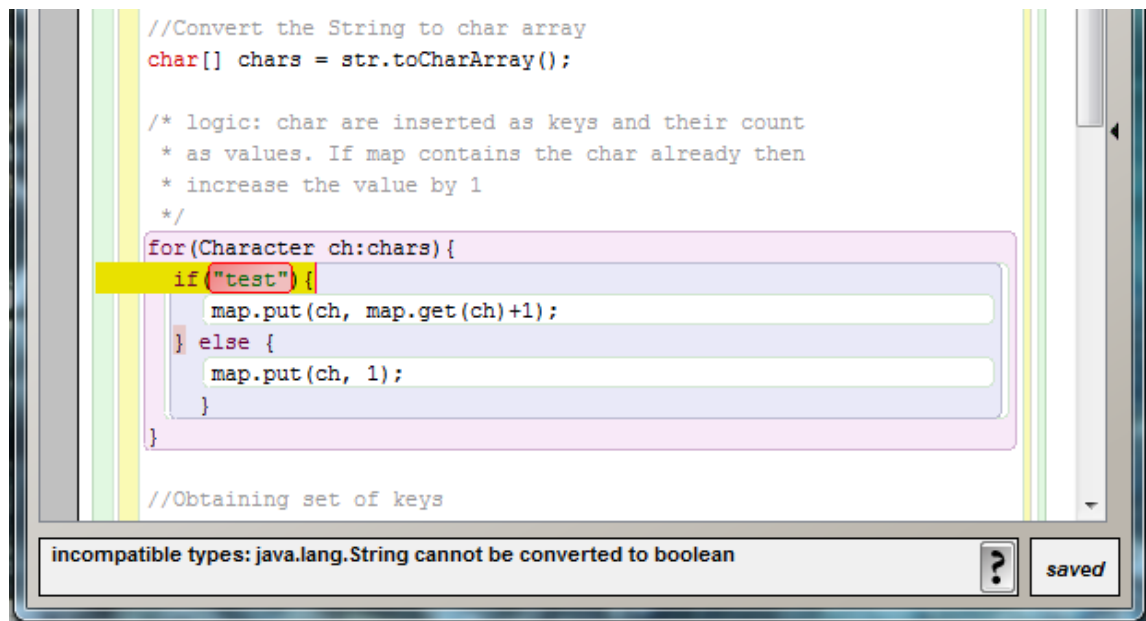
NAJI explains the error in the window.

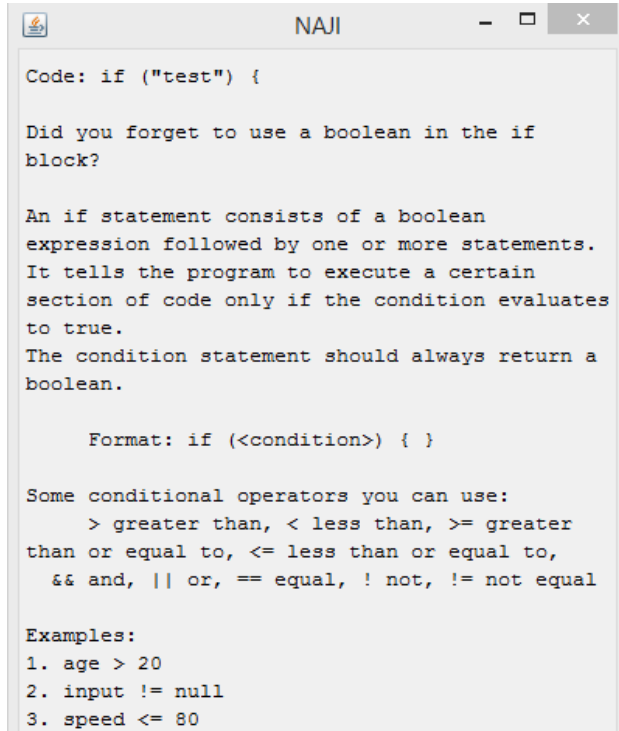


E. Incompatible types errors

1. Didn't use a boolean in if statement

Conditions in the if statement should return a boolean. If it doesn't, incompatible types error is detected by the compiler.



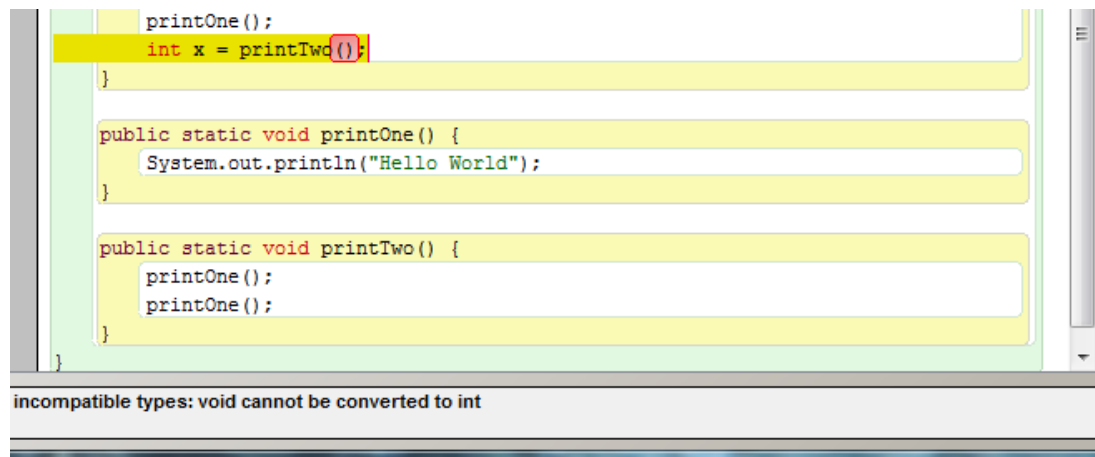


```
Code: if ("test") {  
  
Did you forget to use a boolean in the if  
block?  
  
An if statement consists of a boolean  
expression followed by one or more statements.  
It tells the program to execute a certain  
section of code only if the condition evaluates  
to true.  
The condition statement should always return a  
boolean.  
  
Format: if (<condition>) { }  
  
Some conditional operators you can use:  
  > greater than, < less than, >= greater  
than or equal to, <= less than or equal to,  
  && and, || or, == equal, ! not, != not equal  
  
Examples:  
1. age > 20  
2. input != null  
3. speed <= 80
```

NAJI describes the error committed in the window.

2. Assigned void to a variable

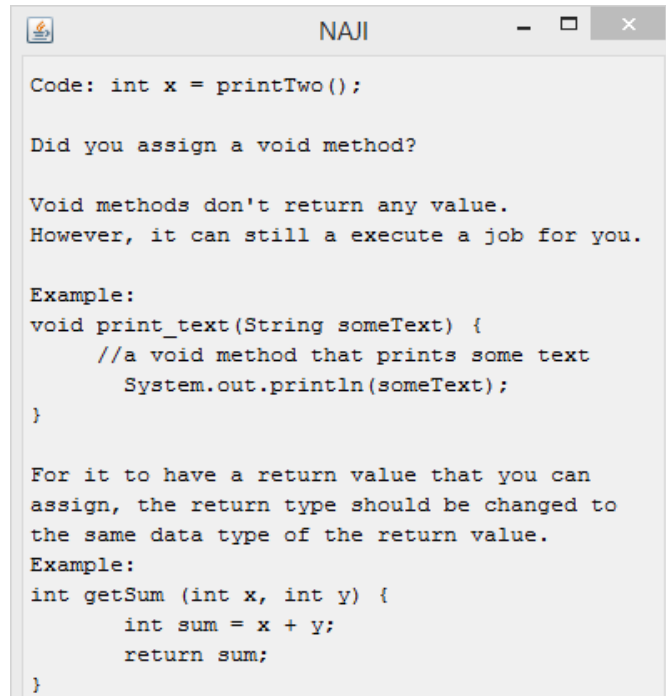
A void method can't be assigned to a variable. This will throw an incompatible types error.



```
printOne();  
int x = printTwo();  
}  
  
public static void printOne() {  
    System.out.println("Hello World");  
}  
  
public static void printTwo() {  
    printOne();  
    printOne();  
}  
}
```

incompatible types: void cannot be converted to int

The output message by NAJI regarding the error is shown in the window.



Code: `int x = printTwo();`

Did you assign a void method?

Void methods don't return any value.
However, it can still a execute a job for you.

Example:

```
void print_text(String someText) {  
    //a void method that prints some text  
    System.out.println(someText);  
}
```

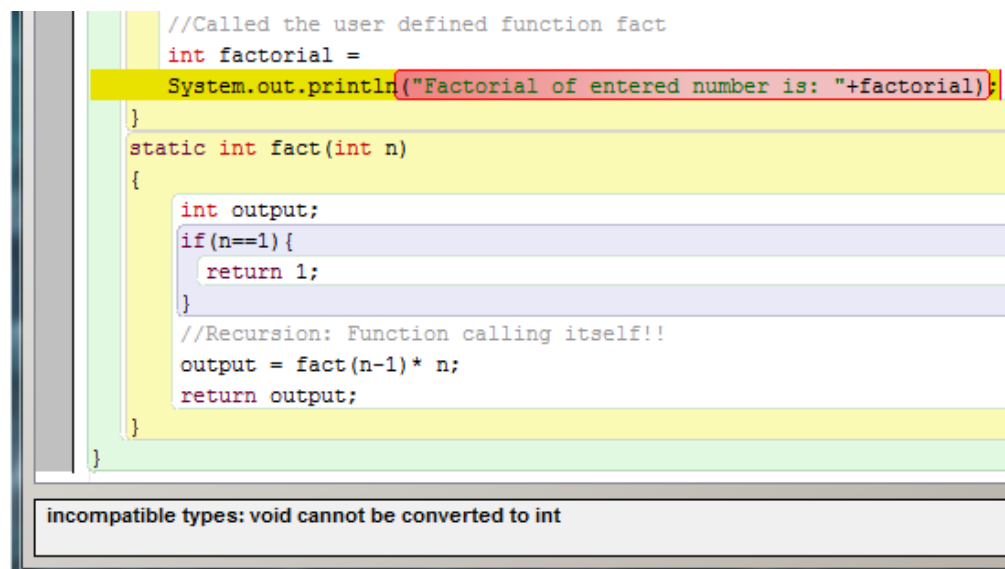
For it to have a return value that you can assign, the return type should be changed to the same data type of the return value.

Example:

```
int getSum (int x, int y) {  
    int sum = x + y;  
    return sum;  
}
```

3. Unfinished assignment statement

An unfinished assignment wherein the value to be assigned is missing causes an incompatible types error.



```
//Called the user defined function fact  
int factorial =  
System.out.println("Factorial of entered number is: "+factorial);  
}  
static int fact(int n)  
{  
    int output;  
    if(n==1){  
        return 1;  
    }  
    //Recursion: Function calling itself!!  
    output = fact(n-1)* n;  
    return output;  
}
```

incompatible types: void cannot be converted to int

The output message by NAJI regarding the error is shown below.

```
Code: System.out.println("Factorial of entered
number is: "+factorial);

Did you forget to finish the assignment
statement of the previous line?

An assignment statement in Java uses the
assignment operator (=) to assign the result of
an expression to a variable.
You code it like this:
    variable = expression;

Examples:
1/ int a = (b * c) / 4;
2. int x;
3. int y = 4;
4. x = 7 + y;
```

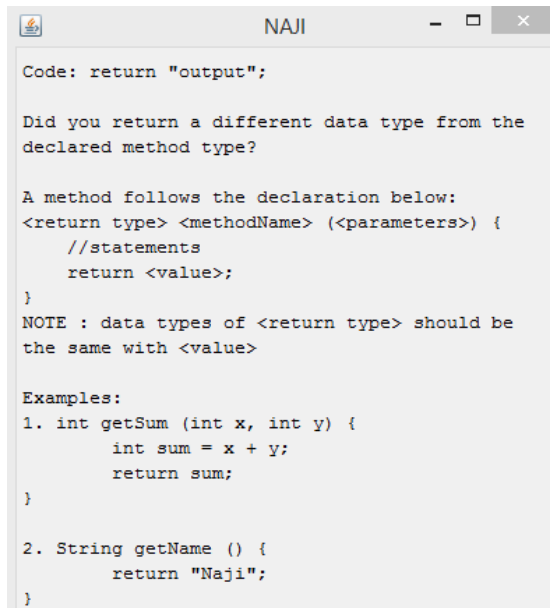
4. Wrong return type

When a return value is different from the declared data type in the method declaration, it causes an incompatible types error.

```
static int fact(int n)
{
    int output;
    if(n==1){
        return 1;
    }
    //Recursion: Function calling itself!!
    output = fact(n-1)* n;
    return "output";
}
}
```

incompatible types: java.lang.String cannot be converted to int

NAJI shows the more descriptive message for the error committed.



The screenshot shows a window titled "NAJI" with a light gray background. The text inside is as follows:

```
Code: return "output";

Did you return a different data type from the
declared method type?

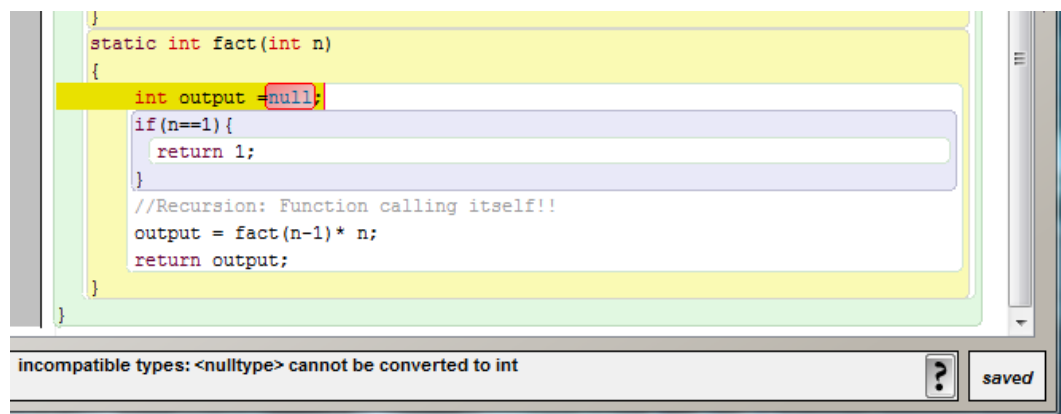
A method follows the declaration below:
<return type> <methodName> (<parameters>) {
    //statements
    return <value>;
}
NOTE : data types of <return type> should be
the same with <value>

Examples:
1. int getSum (int x, int y) {
    int sum = x + y;
    return sum;
}

2. String getName () {
    return "Naji";
}
```

5. Assigning null to a primitive type

Primitive data types expect a value. When a null is assigned to it, the compiler throws an incompatible types error.

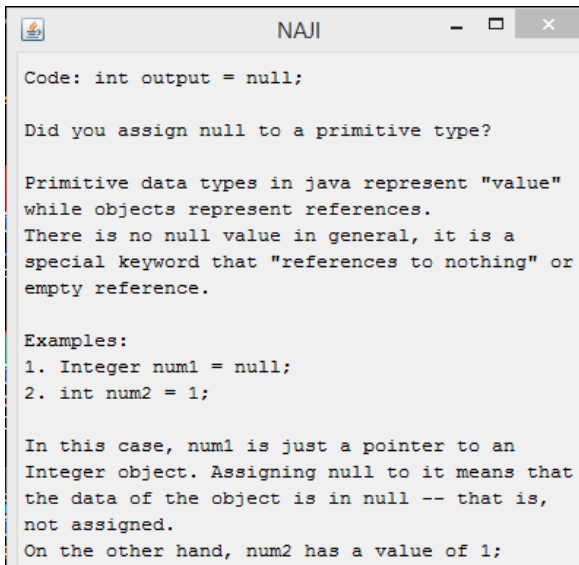


The screenshot shows a code editor window with a yellow background. The code is as follows:

```
}
static int fact(int n)
{
    int output =null;
    if (n==1){
        return 1;
    }
    //Recursion: Function calling itself!!
    output = fact(n-1)* n;
    return output;
}
}
```

The line `int output =null;` is highlighted in yellow. At the bottom of the window, there is a status bar with the text "incompatible types: <nulltype> cannot be converted to int" and a "saved" button.

NAJI explains it elaborately in its window.



```
Code: int output = null;

Did you assign null to a primitive type?

Primitive data types in java represent "value"
while objects represent references.
There is no null value in general, it is a
special keyword that "references to nothing" or
empty reference.

Examples:
1. Integer num1 = null;
2. int num2 = 1;

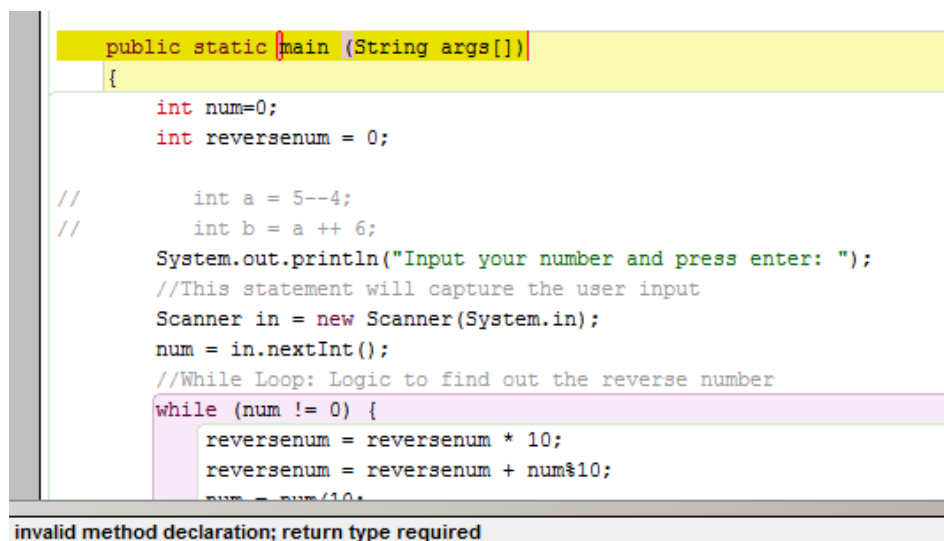
In this case, num1 is just a pointer to an
Integer object. Assigning null to it means that
the data of the object is in null -- that is,
not assigned.
On the other hand, num2 has a value of 1;
```

F. Unhandled compile errors

There errors unhandled by NAJI and those are still shown in the window.

1. No return type in the method declaration

NAJI can't process when a return type is not indicated in the method declaration.

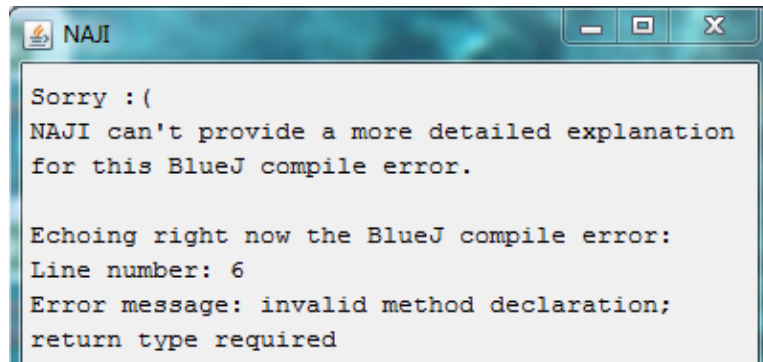


```
public static main (String args[])
{
    int num=0;
    int reversenum = 0;

    //      int a = 5--4;
    //      int b = a ++ 6;
    System.out.println("Input your number and press enter: ");
    //This statement will capture the user input
    Scanner in = new Scanner(System.in);
    num = in.nextInt();
    //While Loop: Logic to find out the reverse number
    while (num != 0) {
        reversenum = reversenum * 10;
        reversenum = reversenum + num%10;
        num = num/10;
    }
}
```

invalid method declaration; return type required

NAJI shows the line number and compile error message instead.



2. No 'class' keyword in the class declaration

Another unhandled error by NAJI is when the class keyword is missing.

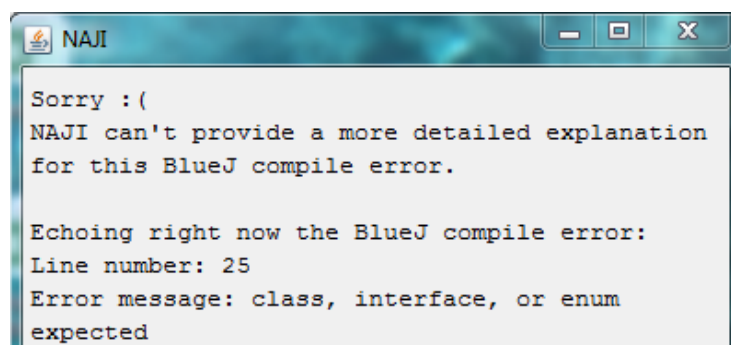
```
import java.util.Scanner;
ReverseNumberWhile
{
//    public ReverseNumberWhile() throws Exception {}

public static void main (String args[])
{
    int num=0;
    int reversenum = 0;

//    int a = 5--4;
//    int b = a ++ 6;
System.out.println("Input your number and press enter: ");
//This statement will capture the user input
Scanner in = new Scanner(System.in);
num = in.nextInt();
//While Loop: Logic to find out the reverse number
while (num != 0) {
    reversenum = reversenum * 10;
    reversenum = reversenum + num%10;
    num = num/10;
}
```

class, interface, or enum expected

NAJI shows the line number and compile error message instead.



3. Non-static method referenced from a static context

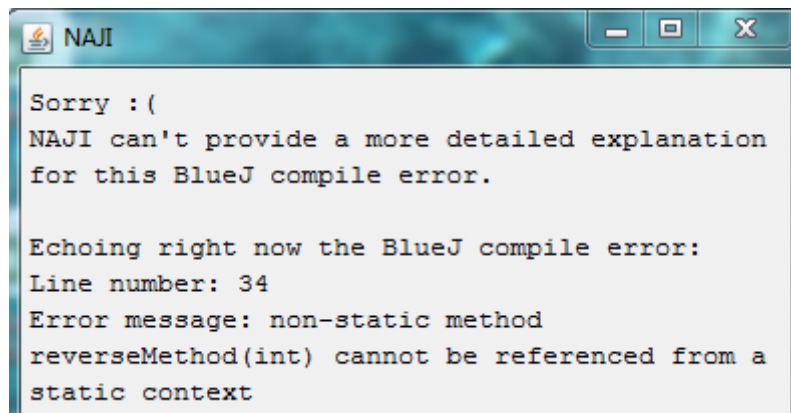
Another unhandled error is when a static method calls a non-static method.

```
public void reverseMethod(int number) {
    if (number < 10) {
        System.out.println(number);
        return;
    }
    else {
        System.out.print(number % 10);
        //Method is calling itself: recursion
        reverseMethod(number/10);
    }
}

public static void main(String args[])
{
    int num = 0;
    System.out.println("Input your number and press enter: ");
    Scanner in = new Scanner(System.in);
    num = in.nextInt();
    System.out.print("Reverse of the input number is:");
    reverseMethod(num);
    // try { reverseMethod(num);
    //     } catch (Exception e) {}
}
```

non-static method reverseMethod(int) cannot be referenced from a static context

NAJI shows the line number and compile error message instead.



```
NAJI

Sorry :(
NAJI can't provide a more detailed explanation
for this BlueJ compile error.

Echoing right now the BlueJ compile error:
Line number: 34
Error message: non-static method
reverseMethod(int) cannot be referenced from a
static context
```


VI. DISCUSSION

The extension, Novice Assistance in Java Introduction, is a tool that aids novice programmers in debugging Java programs. The user compiles their code in BlueJ and if there's a compile error, the NAJI extension first determines the compile error. The second step is it provides information on the error and suggests the fixes.

Without the tool, the error messages by the compiler are not helpful and often lead to confusion for novice programmers. Too often, the compile error messages are cryptic, long or hard to understand. These don't necessarily point the students in the right direction needed to fix the code. New students of the language have a hard time identifying the errors, let alone applying fixes.

With the tool, the errors are described clearly, real causes are pointed out, and better error messages are generated. It's useful during the early phase of learning Java programming as they become more proficient and knowledgeable with the language.

Once the user compiles the code and a compile error results, the tool then assesses the line which caused the error and processes the code. It scans the code to check the syntax, points out the actual error, and suggests the fix for it. It also gives examples for the students to have better understanding.

Users gain self-confidence and experience in debugging with the assistance of our tool. They are able to save time and improve their program comprehension skills.

VII. CONCLUSION

NAJI is a debugging assistant that addresses the needs of novice programmers. It implemented the five errors identified – brace expected, parenthesis expected, semicolon expected, incompatible types, and can't find symbol. The more specific errors messages from the extension help students to clarify concepts, misconceptions, or improve their mental models. In particular, it helps them become better programmers in terms of debugging and writing error-free programs by improving their program comprehension skills and giving them debugging experience.

Though certain compilers may flag correctly some of the most common mistakes, these errors are cryptic for new students of the language. With the help of NAJI, an extension developed, these errors are addressed efficiently since NAJI outputs more specific messages.

The extension determines the compile error from BlueJ. NAJI then processes the errors and provides more information on the error such as background, description, suggestion, and example. This pointed the novice programmer in fixing the syntax error.

VIII. RECOMMENDATION

In the extension developed, only five subcases were handled. It would be better to optimize how these are caught. Also, expanding the subcases per error would cover more error messages and in turn, teach more on debugging syntax errors for novice programmers.

Since only five from the top errors were handled by the current extension, it would be better to add more to cover to minimize the compile errors committed by the novice programmers. Another feature to be added could be some quizzes on the common compile errors committed. The output messages could be improved like a better design and more examples.

Lastly, since the extension is written in Java, it could also be extended to other Java IDE's. This could mean teaching more programmers being introduced to Java.

IX. BIBLIOGRAPHY

- [1] Chmiel, R., and Loui, M. Debugging: From Novice to Expert. Proceedings of the 35th SIGCSE technical symposium on Computer Science education (2004), 17-21.
- [2] Oman, P. W., Cook, R., & Nanja, M. (1989). Effects of programming experience in debugging semantic errors. *Journal of Systems and Software*, 9(3), 197-207.
- [3] Lee, G. C., & Wu, J. C. (1999). Debug It. *Computers & Education*, 32(2), 165-179.
- [4] Tutorial 18 - Debugging. (n.d.). Java. Retrieved December 4, 2013, from <http://home.cogeco.ca/~ve3ll/jatutori.htm>
- [5] Reese, D. Detection of Java Errors. *ACM SIGCSE Bulletin*, 23(1), 31.
- [6] Traver, V. J. (2010). On Compiler Error Messages: What They Say and What They Mean. *Advances in Human-Computer Interaction*, 2010, 1-26.
- [7] Hristova, M., Misra, A., Rutter, M., and Mercuri, R. Identifying and Correcting Java Programming Errors for Introductory CS Students. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium in Computer Science education (New York, NY, USA, 2003)*, ACM, 153-156.
- [8] Jadud, M. Methods and Tools for Exploring Novice Compilation Behaviour. *ICER '06*. (2006), 73-84.
- [9] Ahmazadeh, M., Elliman, D., and Higgins, C. An Analysis of Patterns of Debugging Among Novice Computer Science Students.

- [10] Brun, Y., Ernst, M., Holmes, R., Muslu, K., and Notkin, D. Improving IDE Recommendations by Considering Global Implications of Existing Recommendations.
- [11] transparency debugging with explanations for novice programmers
- [12] Csallner, C. And Smaragdakis, Y. Check 'n Crash: Combining static checking and testing. In ICSE '05: Proceedings of the 27th International Conference on Software Engineering (New York, NY, USA, 2005), ACM, pp. 422-431.
- [13] Johnson, W. And Soloway, E. Proust: Knowledge-based program understanding. In ICSE '84: Proceedings of the 7th International Conference on Software Engineering. (NJ, USA, 1984), IEEE Press, pp. 369-380.
- [14] Aljunid, S., Nordin, M., Shukur, Z. and Zin, A. (2000) A Knowledge-based Automated Debugger in Learning System.
- [15] Bradnt, J., Hartmann, B., Klemmer, S, and MacDougall, D. (2010) What Would Other Programmers Do? Suggesting Solutions to Error Messages.
- [16] Sumptner, M., and Wotawa, F. Jade – Java Diagnosis Experiments Status and Outlook.
- [17] Abramson, D., Chu, C., Ho, T., and Goscinski, W. Eclipse Guard: Relative Debugging in the Eclipse Framework.
- [18] Sykes, E., & Franek, F. (2004). A Prototype for an Intelligent Tutoring System for Students Learning to Program in JavaTM. *Advanced Technology for Learning*, 1(1), 1-6.

- [19] Kuittinen, M., And Sajaniemi, J. 2003. First results of an experiment on using roles of variables in teaching. In The 15th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2003).
- [20] Korhonen, A., Malmi, L., Saikkonen, R.: Design Pattern for Algorithm Animation and Simulation. In E. Sutinen (Ed.): Proceedings of the First Program Visualization Workshop, University of Joensuu, Department of Computer Science, 2001, 89–100.
- [21] Moreno, A., Myller, N., Ben-Ari, M., & Sutinen, E. (2004). Program animation in jeliot 3. *ACM SIGCSE Bulletin*, 36(3), 265.
- [22] Ebrahimi, A., Kopec, D., and Schweikert C. Taxonomy of novice programming errors with plan, web, and object solutions. Unpublished. Submitted to ACM Computing Surveys, December 2006.
- [23] Truong, N., Roe, P., and Bancroft, P. Static analysis of students' java programs. In ACE '04: Proceedings of the sixth on Australian computing education. (Darlinghurst, Australia, Australia, 2004), Australian Computer Society, Inc., pp. 317-325.
- [24] Hristova, M., Misra, A., Rutter, M., and Mercuri, R. Identifying and correcting java programming errors for introductory computer science students. In SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer Science education. (New York, NY, USA, 2003), ACM, pp. 153-156.
- [25] Top-down parsing. (n.d.) *Top-down parsing*. Retrieved April 14, 2016, from <http://www.cs.engr.uky.edu/~lewis/essays/compiler/rec-des.html>

- [26] Recursive Descent Parsing. (n.d.) *Recursive Descent Parsing*. Retrieved May 1, 2016, from <http://www.cs.engr.uky.edu/~lewis/essays/compilers/rec-des.html>
- [27] CS 111: Common Java Errors. (n.d.). *CS 111: Common Java Errors*. Retrieved December 21, 2013, from <http://cs-people.bu.edu/dgs/courses/cs111/assignments/errors.html>
- [28] List of common Java syntax errors. (n.d.). *The Open University*. Retrieved February 1, 2014, from <http://www.open.ac.uk/StudentWeb/m874/!synterr.htm>
- [29] Ben-Ari, M., (2007). Compile and Runtime Errors in Java.
- [30] Anderson, T. (n.d.). Compile-time Errors. *Java Debugging Reference ::*. Retrieved December 1, 2013, from <http://www.terryanderson.ca/debugging/compile.html>
- [31] Leahy, P. (n.d.). Error Message: Cannot Find Symbol. Java. Retrieved January 10, 2014, from <http://java.about.com/od/cerrmsg/g/Definition-Cannot-Find-Symbol.htm>
- [32] Kölling, M., & Rosenberg, J. (2000). Objects first with Java and BlueJ (seminar session). *ACM SIGCSE Bulletin*, 32(1), 429.
- [33] BlueJ Extensions. (n.d.). BlueJ Extensions. Retrieved September 4, 2013, from www.bluej.org/extensions/extensions.html
- [34] Thompson, G. (2008). A Discussion of the BlueJ IDE with Two of Its Developers: Michael Kölling and Ian Utting. *Welcome*. Retrieved March 2, 2014, from <https://today.java.net/pub/a/today/2008/06/26/bluej-interview.html>

X. APPENDIX

A. Source code

NAJI.java

```
package src.SP.NAJI.processor;

import src.SP.NAJI.processor.*;

import bluej.extensions.*;
import bluej.extensions.event.*;
import bluej.extensions.editor.*;

import java.util.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class NAJI extends
Extension implements
CompileListener {
    ErrorManager Mgr;
    BlueJ bluej = null;
    JFrame frame;
    JLabel messageField;
    JTextArea messageArea;

    public NAJI() {
        Mgr = new ErrorManager();
        Mgr.addHandler(new
BraceExError());
        Mgr.addHandler(new
ParenExError());
        Mgr.addHandler(new
InTypesError());
        Mgr.addHandler(new
CantFindSymbolError());
        Mgr.addHandler(new
SemicolonExError());
    }

    public void startup(BlueJ
bluej) {
bluej.addCompileListener(this);
        this.bluej = bluej;

        // Frame parent = null;
        // try {
        // parent =
bluej.getCurrentPackage().getFram
e();
        // }
        // catch(Exception e) {
        // parent =
bluej.getCurrentFrame();
        // }

        frame = new JFrame() {
            public void
frameInit() {
this.setTitle("NAJI");
                //
                this.setSize(400, 200);

                //
                this.setVisible(true);

                this.setLayout(new
                BorderLayout());
                messageField =
                new JLabel();
                messageArea = new
                JTextArea();
                JPanel panelTop =
                new JPanel();
                messageArea.setWrapStyleWord(true
                );
                messageArea.setLineWrap(true);
                messageArea.setEditable(false);
                messageArea.setOpaque(false);
                messageArea.setPreferredSize(new
                Dimension(380, 480));
                //
                panelTop.add(messageField);
                panelTop.add(messageArea);
                //
                this.add(messageField);
                this.add(panelTop);

                this.setDefaultCloseOperation(Win
                dowConstants.DO_NOTHING_ON_CLOSE)
                ;
                //
                this.setContentPane(panelTop);

                this.setPreferredSize(new
                Dimension(400, 500));

                this.setMaximumSize(new
                Dimension(400, 500));

                this.setMinimumSize(new
                Dimension(400, 500));
                //
                this.setResizable(false);
                this.pack();

                this.setVisible(true);

            }
        };

        public void terminate() {}

        public boolean isCompatible()
        {
            return true;
        }
    }
}
```



```

    }

    public void
compileError(CompileEvent event)
{
    String message =
event.getErrorMessage();

    int lineNumber =
event.getErrorLineNumber();
    File target = null;
    for(File f :
event.getFiles()) {
        target = f;
        break;
    }
    if(target != null) {

        try {
            Scanner reader =
new Scanner(target);
            StringBuilder sb
= new StringBuilder();

while(reader.hasNextLine()) {

sb.append(reader.nextLine() +
"\n");
                }
                String result =
Mgr.process(message, lineNumber,
sb.toString());
                //
messageField.setText(result);

messageArea.setText(result);
                frameToFront();
            }
            catch (Exception e) {
e.printStackTrace();
            }
        }

        public void
compileFailed(CompileEvent event)
{

        }

        public void
compileStarted(CompileEvent
event) {}

        public void
compileSucceeded(CompileEvent
event) {
            messageArea.setText("");
//refresh
            //frameToFront();
        }

        public void
compileWarning(CompileEvent
event) {}

        public String getVersion() {
            return "2016.05.21";
        }
    }

```

```

        public String getName() {
            return "IDE Extension";
        }

        public String
getDescription() {
            return "A helper for the
java compiler, which aids in
determining " +
            "the actual error more
precisely.";
        }
    }

Error.java
package src.SP.NAJI.processor;

public interface Error {
    public boolean
isQualified(String errString);
    public String
OutputResults(int linenum, String
code);
}

CodeScanner.java
package src.SP.NAJI.processor;

public class CodeScanner {

    String code;

    int index;

    public CodeScanner(String text) {

        this.code = text;

        index = 0;

    }

    public void goToEndOfLine(int
line) {

        index = -1;

        while(line-- > 0) {

            index =
code.indexOf("\n", index+1);

        }

        if(index == -1) index = 0;

    }
}

```

```

        public boolean
searchForward(String word) {
            int temp =
code.indexOf(word, index+1);

            if(temp > 0) {
                index = temp;
                return true;
            }
            return false;
        }

        public boolean
searchBackward(String word) {
            int temp =
code.lastIndexOf(word, index-1);

            if(temp > 0) {
                index = temp;
                return true;
            }
            return false;
        }

        protected void goToStartOfWord()
{
            if(isWhitespace(index)) {
                do {
                    index++;
                }
                while(isWhitespace(index));
            }
            /*
            else {
                while(isNotWhitespace(index-1)) {
                    index--;
                }
            }
            */
        }

        protected void goToEndOfWord() {
            if(isNotWhitespace(index)) {
                if(!isIdentifier(index)) {
                    //index++;
                    return;
                }
                while(isIdentifier(index+1)) {
                    index++;
                }
            }
            else {
                while(isWhitespace(index)) {
                    index--;
                }
            }
        }

        protected boolean
isWhitespace(int index) {
            if(index < 0) return
false;
            if(index >= code.length())
return false;
            return
Character.isWhitespace(code.charAt(index)
);
        }

        protected boolean
isIdentifier(int index) {
            if(index < 0) return
false;
            if(index >= code.length())
return false;
            String curr =
code.substring(index, index+1);

```

```

        return
!curr.matches("[=\\+\\|-
\\*\\/\\{\\}\\(\\);\\s]");
    }

    protected boolean
isNotWhitespace(int index) {

    false;

    if(index < 0) return
    false;

    if(index >= code.length())
return false;

    return
!Character.isWhitespace(code.charAt(index
));
    }

    public String nextWord() {

    goToStartOfWord();

    if(index == code.length())
{

        return null;

    }

    int start = index;

    goToEndOfWord();

    String retval =
code.substring(start, index+1);

    index++;

    return retval;

    }

    public String previousWord() {

    goToEndOfWord();

    if(index == code.length())
{

        return null;

    }

    int start = index;

    goToStartOfWord();

    String retval =
code.substring(index, start+1);

    }

    }

    public boolean
searchOnLine(String word) {

    int temp = index;

    if(code.charAt(temp) ==
'\n') temp--;

    int low =
code.lastIndexOf("\n", temp);

    int high =
code.indexOf("\n", temp);

    int result =
code.indexOf(word, low);

    if(result < high && result
>= 0) {

        index = result;

        return true;

    }

    return false;

    }

    public boolean
searchForward(String word, int wordLimit)
{

    int initial = index;

    for(int i = 0; i <
wordLimit; i++) {

        String nw =
nextWord();

        if(nw == null) {

            break;

        }

        else

        if(nw.equals(word)) {

            return
true;

        }

    }

    }

```

```

        }
        index = initial;
        return false;
    }

    public int getIndex() {
        return index;
    }

    public void setIndex(int index) {
        this.index = index;
    }

    public String getLine(int line) {
        return getLines(line, line);
    }

    public String getLines(int start, int end) {
        goToEndOfLine(start-1);
        int startIndex = getIndex();
        goToEndOfLine(end);
        int endIndex = getIndex();

        return code.substring(startIndex,
            endIndex);
    }

    public static void main(String[] args) {
        String test = "int a=
5+67;";

        CodeScanner cs = new
CodeScanner(test);

        System.out.println(cs.nextWord());
;

        System.out.println(cs.nextWord());
;

        }
        System.out.println(cs.nextWord());
;

        System.out.println(cs.nextWord());
;

        System.out.println(cs.nextWord());
;

        System.out.println(cs.previousWord());
    }
}

Tools.java
package src.SP.NAJI.processor;

public class Tools {

    public static String
getMethodName(String method) {
        return method.substring(0,
            method.indexOf("("));
    }

    public static int
levenshteinDistance(String wordA, String
wordB) {
        // d is a table with m+1
rows and n+1 columns

        int m = wordA.length();
        int n = wordB.length();

        int[][] d = new
int[m+1][n+1];

        for(int i = 0; i <= m;
i++) {
            d[i][0] = i;
        }

        for(int j = 0; j <= n;
j++) {

```

```

        d[0][j] = j;
    }
    for(int j = 1; j <= n;
j++) {
        for(int i = 1; i
<= m; i++) {
            if(wordA.charAt(i-1) ==
wordB.charAt(j-1)) {
                d[i][j] = d[i-1][j-1];
            }
            else {
                d[i][j] = Math.min(Math.min(d[i-
1][j]+1, d[i][j-1]+1), d[i-1][j-1]+1);
            }
        }
    }
    return d[m][n];
}
}

ErrorManager.java
package src.SP.NAJI.processor;

import java.util.Set;
import java.util.HashSet;
import java.util.Scanner;
import java.io.File;

public class ErrorManager {

    private Set<Error> errors;

    public ErrorManager() {
        errors = new HashSet<Error>();
    }

    public void addHandler(Error e) {
        errors.add(e);
    }

    public String process(String
errorMessage, int lineNumber, String code
) {
        String newline =
System.lineSeparator();

        for(Error e : errors) {
            // return
            "h.accepts(errorMessage): " +
h.accepts(errorMessage)+ "\n
errorMessage:" + errorMessage

            // + "
lineNumber: " + lineNumber + " code:" +
code ;

            if(e.isQualified(errorMessage)) {
                return
e.OutputResults(lineNumber, code);
            }
        }

        return "Sorry :(" +newline
+ "NAJI can't provide a more
detailed explanation for this BlueJ
compile error." + newline +newline
+ "Echoing right now the BlueJ
compile error:" + newline
+ "Line number: " + lineNumber +
newline
+ "Error message: " +
errorMessage;
    }
}

```

```

SemicolonExError.java
package src.SP.NAJI.processor;

public class SemicolonExError implements
Error {

    public boolean isQualified(String
errorMessage) {

        return errorMessage.equals("'" +
expected") ||

            errorMessage.startsWith("'" +
expected");

    }

    public String OutputResults(int
lineNumber, String code) {

        //return "Test:" + test;

        CodeScanner cs = new
CodeScanner(code);

        String newline =
System.lineSeparator();

        String COE = cs.getLine(lineNumber);

        if(code.matches("(?s).*\\s*\\s*(.*)") {

            //return "you have probably used a
>(' instead of a '{';

            return "Code: " + COE.trim() +
newline + newline//cause of error

                + "Did you use a parenthesis
instead of a brace to start your code
block?" + newline + newline

                + "Curly braces {} mark the start
and end of a code block." + newline +
"Examples:" + newline

                + "1. while (number < 10) {" +
newline + "do this> } " + newline

                + "2. public static void
main(String args[]) {" + newline + "<code
block> }" + newline + newline

                + "One the other hand, parentheses
() mark the start and of the parameter
list of a method." + newline +
"Examples:" + newline

                + "1. getAverage(num1, num2);" +
newline + "2. input.nextWord();" +
newline + "3. System.out.print(\"Enter a
number: \");";

        }

        String localizedCode =
cs.getLines(lineNumber - 1, lineNumber +
1);

        if(localizedCode.matches("(?s).*\\d+|\\(|
\\d+|\\w+\\)|) (\\w+|\\(|(\\d+|\\w+\\)|).*
") {

            return "Code: " + COE.trim() +
newline + newline//cause of error

                + "Did you forget to use * for
multiplication?" + newline + newline

                + "The Java programming languages
supports various arithmetic operators." +
newline

                + "For multiplication:" + newline

                + " Operator: *" + newline

                + " Use: op1 * op2" + newline +
newline

                + "Examples: " + newline

                + "1. int x = 34 * 3;" + newline

                + "2. double y = 27.32 * 5.91;" +
newline

                + "3. double z = x * y;";

        }

        if(localizedCode.matches("(?s).*\\w+\\s+[
xX]\\s+\\w+.*")) {

            return "Did you forget to use * for
multiplication?" + newline + newline

                + "The Java programming languages
supports various arithmetic operators." +
newline

                + "For multiplication:" + newline

                + " Operator: *" + newline

                + " Use: op1 * op2" + newline +
newline

                + "Examples: " + newline

```

```

    + "1. int x = 34 * 3;" + newline
    + "2. double y = 27.32 * 5.91;" +
newline
    + "3. double z = x * y;";
}

if(localizedCode.matches("(?s).*\\w\\s*(\\
\\+|\\+|\\-|\\-|\\s*[^];\\s].*)" ) {

    //return "the ++ or -- is a unary
operator";

    return "Code: " + COE.trim() +
newline + newline//cause of error

    + "Did you use a unary instead of a
binary operator?" + newline + newline

    + "Unary operators require only one
operand." + newline

    + "They are used to increment or
decrement a value by 1." + newline +
newline + "Examples:" + newline

    + "1. int result = 0;" + newline

    + "2. result++; //result is now 1"
+ newline

    + "3. result--; //result is now 0"
+ newline + newline

    + "Binary operators require two
operands." + newline

    + "It follows the format :
<Operand1> <operator> <Operand2>" +
newline + newline

    + "Examples:" + newline +"1. 5 + 1"
+ newline + "2. x = y - z" + newline +
"3. num = a + b";

}

if(localizedCode.matches("(?s).*\\w+[a-
zA-Z](\\(.*\\))?:\\s+.*") ) {

    //return "you may have used a :
instead of a ;";

    return "Code: " + COE.trim() +
newline + newline//cause of error

    + "Did you use a : instead of a ;
to end your statement?" + newline +
newline

    + "Semicolons (;) are a part of the
syntax of the programming language. It is
used at the end of the statement to help
the compiler identify when the statement
ends." + newline + newline

    + "Examples:" + newline

    + "1. inputField.getText();" +
newline

    + "2. double y = 27.32 * 5.91;" +
newline

    + "3. button.setText(\"close\");";
}

if(localizedCode.matches("(?s).*for\\(.*,
.*\\).*)" ) {

    //return "you may have used a ,
instead of a ;";

    return "Code: " + COE.trim() +
newline + newline//cause of error

    + "Did you mistype the method
call?" + newline + newline

    + "You may have used a comma
instead of a period." + newline

    + "In Java, calling a method
follows the following format:" + newline

    + "
<class>.<methodname>( <parameters>);" +
newline + newline

    + "Examples:" + newline

    + "1. inputField.getText();" +
newline

    + "2. centerPanel.add(inputField);"
+ newline

    + "3. button.setText(\"close\");";
}

}

if(localizedCode.matches("(?s).*\\w+,*
.*") ) {

    //return "you may have used a ,
instead of a ;";

    return "Code: " + COE.trim() +
newline + newline//cause of error

```

```

        + "Did you mistype the method
call?" + newline + newline

        + "You may have used a comma
instead of a period." + newline

        + "In Java, calling a method
follows the following format:" + newline

        + "
<class>.<methodname><parameters>;" +
newline + newline

        + "Examples:" + newline

        + "1. inputField.getText();" +
newline

        + "2. centerPanel.add(inputField);"
+ newline

        + "3. button.setText(\"close\");"

    }

    //return "localizedCode:" +
localizedCode;

    String lineCode =
cs.getLine(lineNumber).trim();

    lineCode = lineCode.replace("public
", "");

    lineCode = lineCode.replace("private
", "");

    lineCode =
lineCode.replace("protected ", "");

    lineCode = lineCode.replace("static
", "");

    lineCode = lineCode.replace("abstract
", "");

    lineCode = lineCode.replace("final ",
"");

    lineCode =
lineCode.replace("synchronized ", "");

    if(lineCode.matches("\\w+ \\w+
\\w+\\((.*\\).*)") {

        return "you may have used a space
in your method name";

    }

    if(lineCode.matches("\\w+ \\w+
\\w+;.*)") {

        return "you may have used a space
in your variable name";

        }

        cs.goToEndOfLine(lineNumber);

        //return
"cs.goToEndOfLine(lineNumber)";

        //return "cs.searchOnLine(''):" +
cs.searchOnLine(")";

        if(cs.searchOnLine(")")) {

            cs.nextWord();

            String nextWord = cs.nextWord();

            //return "word:" + word+ "
nextWord:" + nextWord + "
cs.previousWord():" + cs.previousWord()+
" nextNext:" + nextNext ;

            if(nextWord.equals("{}")) {

                cs.goToEndOfLine(lineNumber - 1);

                if(!cs.previousWord().equals("{}")) {

                    return "you may have forgotten
to close the previous block";

                }

                else if(nextWord.equals(";")) {

                    if(cs.getLines(lineNumber - 3,
lineNumber +
1).matches("(?s).*[^\\s\\{,}\\s*\\n.*")
{

                        return "the previous line
does not have a semicolon";

                    }

                }

            }

            else if(nextWord.equals("")) &&
cs.previousWord().equals(";")) {

                return "Code: " + COE.trim()
+ newline + newline//cause of error

                + "Did you type an extra ')'
in the method call?" + newline + newline

                + "This error occurs when
there's a typographical error in the
method call." +newline

                + "Calling a method follow
the format below:" + newline

```



```

        + "
<class>.<methodname>(<parameters>);" +
newline + newline

        + "Examples:" + newline

        + "1. branch.setName(\"Coffee
Bean\");" + newline

        + "2. branch.getAddress();"

    }

}

else {

    if(cs.getLines(lineNumber - 3,
lineNumber +
1).matches("(?s).*[^\s\\{,]\\\s*\\\n.*")
{

        return "the previous line does
not have a semicolon";

    }

}

return "Semicolon expected error";

}

}

CantFindSymlError.java

package src.SP.NAJI.processor;

import java.util.*;

public class CantFindSymbolError
implements Error {

    static HashSet<String> methods; {

        methods = new HashSet<String>();

        methods.add("println");

        methods.add("printf");

    }

    static HashSet<String> classes; {

        classes = new HashSet<String>();

        classes.add("String");

        classes.add("void");

        classes.add("boolean");

        classes.add("int");

        classes.add("long");

        classes.add("double");

        classes.add("float");

        classes.add("byte");

        classes.add("char");

        classes.add("return");

    }

    String errorMessage;

    public boolean isQualified(String
errorMessage) {

        this.errorMessage = errorMessage;

        return
errorMessage.startsWith("cannot find
symbol");

    }

    public String OutputResults(int
lineNumber, String code) {

        CodeScanner cs = new
CodeScanner(code);

        String[] temp =
errorMessage.split(" +");

        String symbol = temp[4];

        String name = temp[5];

        String newline =
System.lineSeparator();

        String COE =
cs.getLine(lineNumber);

        if(symbol.equals("variable")) {

            cs.goToEndOfLine(lineNumber);

```

```

        if(cs.searchOnLine("=")) {
            cs.nextWord();

if(cs.nextWord().equals("(")) {
            return "Code: " +
COE.trim() + newline + newline//cause of
error

            + "Did you use an
equal sign in calling a method?" +
newline + newline

            + "In Java, calling a
method follows the format below:" +
newline

            + "
<class>.<methodname><parameters>);" +
newline + newline

            + "Examples:" +
newline

            + "1.
Student.getName();" + newline

            + "2.
Student.setCourse(\"CS\");" + newline

            + "3.
Student.getStudentNumber();" + newline;

        }

    }

    return "Code: " + COE.trim()
+ newline + newline//cause of error

    + "Did you forget to declare
'" + name + "'?" + newline + newline

    + "The Java programming
language is statically-typed, which means
that all variables must first be declared
before they can be used. "

    + "This involves stating the
variable's type and name before assigning
a value to it." + newline + newline

    + "Examples:" + newline

    + "1. int " + name + ";" +
newline

    + "2. float " + name + ";" +
newline

    + "3. String " + name + ";" +
newline;

    }

    if(symbol.equals("method")) {

```

```

        String missingMethod =
Tools.getMethodName(name);

        int min = 6;

        String best = "";

        for(String method : methods)
    {

        if(method.equals(missingMethod)) {

            return "you probably
forgot to type the fully qualified name
of the method, or are passing it the
wrong parameters";

        }

        int dist =
Tools.levenshteinDistance(method,
missingMethod);

        if(dist < min) {

            min = dist;

            best = method;

        }

    }

    if(min < 6) {

        return "Code: " +
COE.trim() + newline + newline//cause of
error

        + "Did you misspell " +
best + " as " + name + "?" + newline +
newline

        + "Primitive data types
are predefined by the language and named
by a keyword." + newline + newline

        + "Below are some of the
data types used in Java:" + newline

        + "int" + newline +
"long" + newline + "byte" + newline +
"float" + newline+ "String" + newline +
"boolean" +newline + newline

        + "Examples: " + newline

        + "1. String studName;" +
newline

        + "2. int studNumber;" +
newline

        + "3. boolean
isEnrolled;"

    }

}

```

```

        return "Code: " + COE.trim()
+ newline + newline//cause of error

        + "Did you misspell the
method, '" + missingMethod + "'" +
newline + newline

        + "This error occurs when a
method is called incorrectly: either its
name is mistyped(incorrect casing of
letters)," + " "

        + "or a method is called with
wrong types of parameters, or a method is
called for a wrong type of object or a
wrong class." + newline + newline

        + "For example, the same
error will be reported if you type" +
newline

        + "
input.setText(\"test\");" + newline

        + "instead of" + newline

        + "
input.setText(\"test\");";

    }

    if(symbol.equals("class")) {

        int min = 3;

        String best = "";

        for(String clas : classes) {

            int dist =
Tools.levenshteinDistance(clas, name);

            if(dist < min) {

                min = dist;

                best = clas;

            }

        }

        if(min < 3) {

            return "Code: " +
COE.trim() + newline + newline//cause of
error

            + "Did you misspell " +
best + " as " + name + "?" + newline +
newline

            + "Primitive data types
are predefined by the language and named
by a keyword." + newline + newline

            + "Below are some of the
data types used in Java:" + newline

```

```

        + "int" + newline +
"long" + newline + "byte" + newline +
"float" + newline+ "String" + newline +
"boolean" +newline + newline

        + "Examples: " + newline

        + "1. String studName;" +
newline

        + "2. int studNumber;" +
newline

        + "3. boolean
isEnrolled;";

    }

    return "Code: " + COE.trim()
+ newline + newline//cause of error

    + "Did you forget to import
the required package for '" + name + "'" +
+ newline + newline

    + "When the class you are
using is in a package, importing the
package is needed to be able to access
the class." + newline

    + "Unless installed as a
package and properly imported, files for
classes used in the program should be
available in the same folder." + newline
+ newline

    + "Example:" + newline + "To
be able to use Scanner in the following
code" + newline

    + "    Scanner input = new
Scanner(System.in)" + newline

    + "You need" + newline + "
import java.util.Scanner" + newline + "at
the top of your program.";

    //return "you may have
mistyped '" + name + "' or you have
forgotten to import the required
package";

    }

    return "Can't find symbol error";

}

ParenExError.java

package src.SP.NAJI.processor;

```

```

public class ParenExError implements
Error {

    String errorMessage;

    public boolean isQualified(String
errorMessage) {

        this.errorMessage = errorMessage;

        return errorMessage.equals("'" +
expected") ||

            errorMessage.equals("class,
interface, or enum expected") ||

            errorMessage.equals("'" +
expected") ||

errorMessage.startsWith("<identifier>
expected");

    }

    public String OutputResults(int
lineNumber, String code) {

        CodeScanner cs = new
CodeScanner(code);

        String localizedCode =
cs.getLine(lineNumber);

        CodeScanner cslc = new
CodeScanner(localizedCode);

        String newline =
System.lineSeparator();

        String COE =
cs.getLine(lineNumber);

        if (localizedCode.indexOf("<") >
0 ) {

            return "Code: " + COE.trim()
+ newline + newline//cause of error

            + "Did you forget to give a
complete condition list in your if
block?" + newline + newline

            + "The condition is needed to
be evaluated for the succeeding code
block to be executed." + " "

            + "It tells the program to
execute a certain section of code only if
the condition evaluates to true." +
newline

```

```

+ "The condition statement
should always return a boolean." +
newline+newline

        + "    Format: if
(<condition>) { } " + newline + newline

        + "Some conditional operators
you can use:" + newline

        + "    > greater than, <
less than, >= greater than or equal to,
<= less than or equal to, "+ " "

        + "    && and, || or, ==
equal, ! not, != not equal" +
newline+newline

        + "Examples: " + newline

        + "1. age > 20" + newline +
"2. input != null" + newline + "3. speed
<= 80";

    } //naji

cs.goToEndOfLine(lineNumber);

int newIndex = 0;

if(cs.searchBackward("new")) {

    newIndex = cs.getIndex();

}

//    if (true) return
"localized code: " + localizedCode;

//    if
(localizedCode.indexOf("if") > 0 ||
localizedCode.indexOf("while") > 0 ||
localizedCode.indexOf("for") > 0) {

//    return "did
you forget to give a complete parameter
list in your loop block?";

//    } //naji

if(localizedCode.indexOf("if") >
0 || localizedCode.indexOf("while") > 0
|| localizedCode.indexOf("for") > 0) {

//||
localizedCode.searchOnLine("while") ||
localizedCode.searchOnLine("for")

```

```

        //return localizedCode +
newline +"did you forget the
parentheses?";

        return "Code: " + COE.trim()
+ newline + newline//cause of error

        + "Did you forget the
parentheses in your condition?" + newline
+ newline

        + "Conditions are to be
placed inside parentheses for loops. It
follows the format:" + newline

        + "    <loop type>
(<condition>) { }" + newline + newline

        + "Examples:" + newline

        + "1. if (num > 6) { }" +
newline

        + "2. while (age <= 10) { } "
+ newline

        + "3. for (int x = 0; x < 3;
x++) { }";
    }

    cs.goToEndOfLine(lineNumber);

    cs.searchBackward("void");

    if(cs.getIndex() > newIndex) {

        cs.nextWord();

        String word2 = cs.nextWord();

        String word3 = cs.nextWord();

        //debug test

        //            return
"cs.getIndex(): "+cs.getIndex() + newline
+ "newIndex: " + newIndex + newline

        //            + "word2: " +
word2 + newline + "word3: " + word3;

        if(word3 != null) {

            if(word3.equals("{}")) {

                return "'(' expected
after " + word2;

            }

        }

    }

    else
if(word3.equals("{}")) {

        return "use a
parenthesis '(', not a bracket '['";

    }

    else
if(word3.equals(";")) {

        //return "void cannot
be used as a variable type";

        return "Code: " +
COE.trim() + newline + newline//cause of
error

        + "Did you use void a
data type for your variable?" + newline +
newline

        + "void is not a
valid type. However, void is a valid
keyword used to indicate that a method
does not return a value."
+newline+newline

        + "Below are some of
the data types used in Java:" + newline

        + "int" + newline +
"long" + newline + "byte" + newline +
"float" + "String" + newline + "boolean"
+newline+newline

        + "Examples: " +
newline

        + "1. String
studName;" + newline

        + "2. int
studNumber;" + newline

        + "3. boolean
isEnrolled;";

    }

    else for(String modifier
: BraceExError.modifiers) {

        if(Tools.levenshteinDistance(modifier,
word3) < 2) {

            return "you
forgot the parameter list and opening
'{"";

        }

    }

}

```

```

        if
(BraceExError.modifiers.contains(word2))
{
        return "Code: " +
COE.trim() + newline + newline//cause of
error

        + "Did you declare a data
type in your void method?" + newline +
newline

        + "Void methods don't
return any value." + newline

        + "However, it can still
a execute a job for you. Below is a void
method that prints some text." +
newline+newline

        + "Example: " + newline

        + "void print_text(String
someText) {" + newline

        + "
System.out.println(someText);" + newline
+ "}" + newline + newline

        + "For it to have a
return value that you can assign, the
return type should be changed to the same
data type of the return value." +
newline+newline

        + "Example: " + newline

        + "int getSum (int x, int
y) {" + newline

        + "
int sum = x + y;"

+ newline

        + "
return sum;" +
newline + "}";
    }

//          for(String
modifier :
BraceExpectedErrorHandler.modifiers) {

//          return
"Too many modifiers" +newline+"modifier:
" + modifier + newline + "word2: " +
word2 +

//          newline +
"Tools.levenshteinDistance(modifier,
word2): " +
Tools.levenshteinDistance(modifier,
word2);

// //
if(Tools.levenshteinDistance(modifier,
word2) < 2) {

// //
return "you may be using too many
modifiers";

// //          }

//          }

cs.goToEndOfLine(lineNumber);

/*if(cs.searchOnLine("if")||
cs.searchOnLine("while") ||
cs.searchOnLine("for")) {

return "did you forget to
give a parameter list in your
if/for/while statement";

}*/

//return "spaces are not
allowed in the method declaration";
}

else {

if(errorMessage.equals("'['
expected")) {

return "you forgot to
specify the size of the array, or you are
using new with a primitive incorrectly";

}

else {

return "you are using new
incorrectly or forgot to specify the size
of the array";

}

}

return "Code: " + COE.trim() +
newline + newline//cause of error

+ "Did you include some illegal
characters in your method name?" +
newline + newline

+ "Method names are identifiers
in Java. Java identifiers may be composed
of letters, numbers, underscore, and a
dollar sign." + newline

+ "However, it may only start
with a letter, underscore, and a dollar
sign." + newline + newline

+ "Examples:" + newline

```

```

        + "1. MyMethod" + newline + "2.
        _MyMethod" + newline + "3. My_Method" +
        newline + "4. $newMethod" + newline + "5.
        _1stMethod" +newline+newline

        + "Not allowed names:" + newline

        + "1. My method //has a space" +
        newline + "2. 1stMethod //begins with
        number" + newline + "3. Test1-2-3
        //hyphen is not an alphanumeric
        character";

    }

}

```

InTypesError.java

```
package src.SP.NAJI.processor;
```

```
import java.util.*;
```

```
public class InTypesError implements
Error {
```

```
    String errorMessage;
```

```
    public boolean isQualified(String
errorMessage) {
```

```
        this.errorMessage = errorMessage;
```

```
        return
errorMessage.equals("incompatible types")
||
```

```
errorMessage.startsWith("incompatible
types");
```

```
    }
```

```
    public String OutputResults(int
lineNumber, String code) {
```

```
        CodeScanner cs = new
CodeScanner(code);
```

```
        cs.goToEndOfLine(lineNumber);
```

```
        String localizedCode =
cs.getLine(lineNumber);
```

```
        CodeScanner cslc = new
CodeScanner(localizedCode);
```

```
        String[] codeList = code.split("
+");//code.split(" +");
```

```
        String found = "";//temp[4];
```

```
        String expected = "";//temp[7];
```

```
        String newline =
System.lineSeparator();
```

```
        String COE = cs.getLine(lineNumber);
```

```
        if(cs.searchOnLine("(")) {
```

```
            String[] temp =
Tools.getMethodName(localizedCode).split(
" +");
```

```
            String method = temp[temp.length
- 1];
```

```
            int index =
Arrays.asList(codeList).indexOf(method+"(
)");
```

```
            String datatype = codeList[index
- 1];
```

```
            //          return method + newline +
index ;//+ datatype;
```

```
            if
(datatype.trim().equals("void")) {
// return "there is nothing to assign
since the method you are using returns
void";
```

```
                return "Code: " +
COE.trim() + newline + newline//cause of
error
```

```
                + "Did you assign a void
method?" + newline + newline
```

```
                + "Void methods don't
return any value." + newline
```

```
                + "However, it can still a
execute a job for you." + newline+
newline +"Example:" + newline
```

```
                + "void print_text(String
someText) { " + newline + "          //a void
method that prints some text" + newline
```

```
                + "
System.out.println(someText);" + newline
+ "}" + newline + newline
```

```
                + "For it to have a return
value that you can assign, the return
type should be changed to the same data
type of the return value." + newline
```

```

        + "Example:" + newline
        + "int getSum (int x, int
y) {" + newline
        + "    int sum = x + y;"
+ newline
        + "    return sum;" +
newline + "};"
    }
}

//    return "localizedCode:" +
localizedCode + "
cslc.searchForward(null):"+
cslc.searchForward("null");

    if(cslc.searchForward("null")) {

        //return "you cannot assign null
to a primitive type";

        return "Code: " + COE.trim() +
newline + newline//cause of error

        + "Did you assign null to a
primitive type?" + newline + newline

        + "Primitive data types in java
represent \"value\" while objects
represent references." + newline

        + "There is no null value in
general, it is a special keyword that
\"references to nothing\" or empty
reference." +newline+newline

        + "Examples: " +newline

        + "1. Integer num1 = null;" +
newline + "2. int num2 = 1;" +
newline+newline

        + "In this case, num1 is just a
pointer to an Integer object. Assigning
null to it means that the data of the
object is in null -- that is, not
assigned." + newline

        + "On the other hand, num2 has a
value of 1;"

    }

    if (cs.searchOnLine("if")) {

        cs.nextWord();

        cs.nextWord();

        String varName = cs.nextWord();

        cs.searchBackward(varName);

        cs.searchBackward(varName);

        cs.nextWord();

        cs.nextWord();

        String value = cs.nextWord();

        //String prev =
cs.previousWord();

        if (value != "true" || value !=
"false") { //not a boolean

            //return "1 you may only use
booleans in an if statement";

            return "Code: " + COE.trim()
+ newline + newline//cause of error

            + "Did you forget to use a
boolean in the if block?" + newline +
newline

            + "An if statement consists
of a boolean expression followed by one
or more statements." + " "

            + "It tells the program to
execute a certain section of code only if
the condition evaluates to true." +
newline

            + "The condition statement
should always return a boolean." +
newline+newline

            + "    Format: if
(<condition>) { } " + newline + newline

            + "Some conditional operators
you can use:" + newline

            + "    > greater than, <
less than, >= greater than or equal to,
<= less than or equal to, "+ " "

            + "    && and, || or, ==
equal, ! not, != not equal" +
newline+newline

            + "Examples: " + newline

            + "1. age > 20" + newline +
"2. input != null" + newline + "3. speed
<= 80";

        }

    }

}

```



```

        if((cs.searchOnLine("if") ||
cs.searchOnLine("while") ||
cs.searchOnLine("for")) &&
expected.equals("boolean")) {

            if(cs.searchOnLine("=") &&
cs.nextWord().equals("=")) {

                return "you probably meant
'==' instead of just '='";

            }

            else {

                return "2 you may only use
booleans in an if statement";

            }

        }

        if(cs.searchOnLine("return")) {
//wrongreturntype

            //return "the variable you are
returning has a different type from the
method return type";

            return "Code: " + COE.trim() +
newline + newline//cause of error

            + "Did you return a different
data type from the declared method type?"
+ newline +newline

            + "A method follows the
declaration below:" + newline

            + "<return type> <methodName>
(<parameters>) {" + newline

            + "    //statements" + newline

            + "    return <value>;" +
newline+ "}" + newline

            + "NOTE : data types of <return
type> should be the same with <value>" +
newline + newline

            + "Examples: " + newline

            + "1. int getSum (int x, int y)
{" + newline

            + "    int sum = x + y;" +
newline

            + "    return sum;" +
newline + "}" + newline + newline

            + "2. String getName () {" +
newline

            + "    return \"Naji\";" +
newline + "}"

        }

    }

}

//    return "unknown";

//unfinished statement

        if(cs.getLines(lineNumber - 3,
lineNumber +
1).matches("(?s).*[^\s\\{,}\\s*\\n.*")
{

            //return "the previous line
does not have a semicolon";

            return "Code: " + COE.trim()
+ newline + newline//cause of error

            + "Did you forget to finish
the assignment statement of the previous
line?" + newline + newline

            + "An assignment statement in
Java uses the assignment operator (=) to
assign the result of an expression to a
variable."+ newline

            + "You code it like this:" +
newline

            + "    variable =
expression;" + newline + newline

            + "Examples: " + newline

            + "1/ int a = (b * c) / 4;" +
newline

            + "2. int x;" + newline + "3.
int y = 4;" + newline

            + "4. x = 7 + y;"

        }

        return "Incompatible types error";

    }

}

}

BraceExError.java

package src.SP.NAJI.processor;

import java.io.File;

import java.util.Scanner;

import java.util.HashSet;

```

```

public class BraceExError implements
Error {

    static HashSet<String> modifiers; {
        modifiers = new
HashSet<String>();
        modifiers.add("public");
        modifiers.add("private");
        modifiers.add("protected");
        modifiers.add("int");
        modifiers.add("double");
        modifiers.add("String");
        modifiers.add("float");
        modifiers.add("long");
        modifiers.add("byte");
        modifiers.add("char");
        modifiers.add("boolean");
    }

    public boolean isQualified(String
errorMessage) {
        return errorMessage.equals("'{'
expected") ||
        errorMessage.startsWith("'{'
expected");
    }

    public String OutputResults(int
lineNumber, String code) {

//        return "brace process";

        String newline =
System.lineSeparator();

        CodeScanner cs = new
CodeScanner(code);

        String COE =
cs.getLine(lineNumber); //cause of error

        cs.goToEndOfLine(lineNumber);

        cs.searchBackward("class");

        boolean inClass = false;

        boolean foundBrace =
cs.searchBackward("{}");

        if(foundBrace) inClass =
cs.searchBackward("class");

        if(foundBrace && !inClass)
cs.searchForward ("class");

        if(inClass)
cs.searchForward("class");

        cs.nextWord();

        cs.nextWord();

        String word3 = cs.nextWord();

        String word4 = cs.nextWord();

//        return "word3:" + word3 + "
word4:" + word4 + " inClass:" + inClass;

        if(word3.equals("")) {

            if(inClass) {

                return "you may have used
class as a type";

            }

            if(word4.equals("")) {

                //return "you may have
put parameters '()' in the class
declaration";

                return "Code: " +
COE.trim() + newline + newline//cause of
error

                + "Did you put '()' in
the class declaration?" + newline +
newline

                + "Classes don't have a
parameter list when declared. Parameter
lists are used in methods and
constructors." + newline + newline

                + "Examples:" + newline

                + "1. class NameOfClass
{" + newline

                + "        //field,
constructor, and" + newline

                + "        //method
declarations" + newline

                + "}" //end of class" +
newline +newline

```

```

                + "2. public class Student
{ " + newline
                + "    private String
name;" + newline
                + "    private String
subject;" + newline
                + "    private String
grade;" + newline
                + "    }";
            }

            //return "you may have used a
parentheses '(', instead of a brace '{";
+ newline + newline//cause of error

            + "Did you type '(' instead
of a brace '{' as opening of the code
block?" + newline + newline

            + "The code block after the
class name is opened and closed by
braces, {}." + newline + newline

            + "Examples: " + newline
+ newline
            + "1. public class Data { }"
+ newline
            + "2. class Rectangle { } " +
newline
            + "3. public class Person {
}";
        }

        if(word3.equals("[")) {

            //return "you may have used a
bracket '[', instead of a brace '{";

            return "Code: " + COE.trim()
+ newline + newline//cause of error

            + "Did you type '(' instead
of a brace '{' as opening of the code
block?" + newline + newline

            + "The code block after the
class name is opened and closed by
braces, {}." + newline + newline

            + "Examples: " + newline
+ newline
            + "1. public class Data { }"
+ newline
            + "2. class Rectangle { } " +
newline
            + "3. public class Person {
}";
        }

        if(word3.equals(";")) {

            if(word4.equals("{}") ||
word4.equals("[") || word4.equals("(")) {

                //return "there may be an
extra semicolon in the class
declaration";

                return "Code: " +
COE.trim() + newline + newline//cause of
error

                + "Did you put a
semicolon after the class declaration?" +
newline + newline

                + "Semicolons are used to
terminate. Class definitions are not
terminated. It's followed by a code block
{}." + newline + newline

                + "A class declaration
looks like this:" + newline

                + "    [modifiers] class
ClassName { ... }"+ newline + newline

                + "Examples:" + newline +
"1. public class ImaginaryNumber { }" +
newline
                + "2. class
RecursiveNumber { }" + newline + "3.
public class BraceExError{ }";

            }

            return "you may have used
class as a type";

        }

        if(word3.equals("throws")) {

            //return "you may have put a
throws statement in the class
declaration";

            return "Code: " + COE.trim()
+ newline + newline//cause of error

            + "Did you use a throws
statement in the class declaration?" +
newline + newline

            + "Exceptions can only be
thrown from methods or constructors." +
newline + newline

            + "Instead of" + newline +
newline

            + "    class TestClass
throws Exception {}" + newline + newline

```

```

        + "You may follow the
following:" + newline

        + "1. public class TestClass
{" + newline

            + "    public TestClass()
throws Exception { } //Exception thrown
from class constructor" + newline

            + "}" + newline + newline

        + "2. public class TestClass
{" + newline

            + "    public
methodInClass() throws Exception { }
//Exception thrown from method in the
class" + newline

            + "}";

        }

        if(word3.equals("")) {

            return "you may have used
class as a type";

        }

        for(String modifier : modifiers)
{

if(Tools.levenshteinDistance(modifier,
word3) < 2) {

            return "you forgot the
opening '{'";

        }

        //return "you may have used a
space in the class name";

        return "Code: " + COE.trim() +
newline + newline//cause of error

        + "Did you include some illegal
characters in your class name?" + newline
+ newline

        + "Class names are identifiers in
Java. Java identifiers may be composed of
letters, numbers, underscore, and a
dollar sign." + newline

        + "However, it may only start
with a letter, underscore, and a dollar
sign." + newline + newline

        + "Examples:" + newline

        + "1. MyClass" + newline + "2.
_MyClass" + newline + "3. My_Class" +

```

```

newline + "4. $newClass" + newline + "5.
_1stClass" + newline+newline

        + "Not allowed names:" + newline

        + "1. My class //has a space" +
newline + "2. 1stClass //begins with
number" + newline + "3. Test1-2-3
//hyphen is not an alphanumeric
character";

    }

}

```

XI. ACKNOWLEDGEMENT

Thank you to everyone who never gave up on me, made sure I was on the right track, supported me all throughout this struggle (lol), and believed I can make it!

This is for all of us! 😊