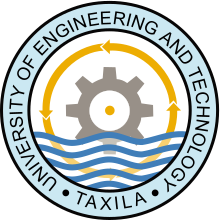
GORCAN 3D



**Submitted By:**

|  |  |
| --- | --- |
| Kinza Nisar | 17-CS-44 |
| Jaudet Ahsan | 17-CS-24 |
| Saman Ejaz | 17-CS-27 |

**Instructor:**

Ma’am Momina Masood

DEPARTMENT OF COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE UNIVERISTY OF ENGINEERING AND TECHNOLOGY TAXILA

DECEMBER 2019

**ACKNOWLEDGEMENT**

We have worked very hard for this project. However, it would not have been possible without our team work and collaboration.

We are highly thankful to Sir Rao Wakeel for assigning us this amazing project and for his amazing support, guidance and time. And for believing in the two of us.

We would like to thank our parents and each other for the corporation which help us to complete this project.

Table of Contents

[**ACKNOWLEDGEMENT** 1](#_Toc533581932)

[**CHAPTER 1: INTRODUCTION** 4](#_Toc533581933)

[Introduction to problem: 4](#_Toc533581934)

[Solution to the problem: 4](#_Toc533581935)

[**Algorithms:** 4](#_Toc533581936)

[(1) Main program: 4](#_Toc533581937)

[(2) Autocomplete 5](#_Toc533581938)

[(3) Getwords 6](#_Toc533581939)

[(4) Getchild: 6](#_Toc533581940)

[(5) Tostring 7](#_Toc533581941)

[(6) Insert 7](#_Toc533581942)

[(7) Search: 7](#_Toc533581943)

[**CHAPTER 2: FUNCTIONAL REQUIREMENTS** 9](#_Toc533581944)

[ File formation 9](#_Toc533581945)

[ Insertion in a trie 9](#_Toc533581946)

[ Input from user 9](#_Toc533581947)

[ Main Functionality 9](#_Toc533581948)

[ BindAutocompletion 9](#_Toc533581949)

[**CHAPTER 3: LANGUAGE AND DATA STRUCTURE** 10](#_Toc533581950)

[**Introduction to language:** 10](#_Toc533581951)

[**Introduction to java:** 10](#_Toc533581952)

[Java Requirements: 10](#_Toc533581953)

[**Introduction to javafx** 10](#_Toc533581954)

[JavaFX: 10](#_Toc533581955)

[JavaFX FXML: 11](#_Toc533581956)

[Properties in FXML: 11](#_Toc533581957)

[Library used in FXML: 11](#_Toc533581958)

[JavaFX Scene Builder: 11](#_Toc533581959)

[**Introduction to data structures used:** 12](#_Toc533581960)

[**(1)** **Trie:** 12](#_Toc533581961)

[Definition: 12](#_Toc533581962)

[Template to make Trie node: 12](#_Toc533581963)

[Applications of Trie Data Structure: 12](#_Toc533581964)

[Use of Trie in our Code: 13](#_Toc533581965)

[**(2)** **Doubly LinkedList** 13](#_Toc533581966)

[Definition: 13](#_Toc533581967)

[Template to make a node for doubly liked list: 13](#_Toc533581968)

[Memory Representation of a doubly linked list: 13](#_Toc533581969)

[Use of doubly Linked List in our Code: 13](#_Toc533581970)

[**(3)** **ArrayList:** 14](#_Toc533581971)

[Definition: 14](#_Toc533581972)

[Template to make an Arraylist: 14](#_Toc533581973)

[**(4)** **Lists:** 15](#_Toc533581974)

[Definition: 15](#_Toc533581975)

[Template to make an Array: 15](#_Toc533581976)

[Functionality: 15](#_Toc533581977)

[**(5)** **Arrays:** 15](#_Toc533581978)

[Definition: 15](#_Toc533581979)

[Template to make an Array: 15](#_Toc533581980)

[**CHAPTER 4: FLOWCHARTS AND GUI** 16](#_Toc533581981)

[**Introduction:** 16](#_Toc533581982)

[**Flowcharts** 16](#_Toc533581983)

[(1) Tostring 16](#_Toc533581984)

[(2) Main program 17](#_Toc533581985)

[(3) Autocomplete 18](#_Toc533581986)

[(4) Getwords 19](#_Toc533581987)

[(5) Getchild 20](#_Toc533581988)

[(6) Insert 21](#_Toc533581989)

[(7) Search 22](#_Toc533581990)

[**Class Diagrams** 22](#_Toc533581991)

[**CHAPTER 5: CODE** 24](#_Toc533581992)

[**CHAPTER 6: REFERENCES** 25](#_Toc533581993)

**CHAPTER 1: INTRODUCTION**

Introduction to problem:

A program that takes a few characters and predicts the whole word. Communication in the modern world is mostly being carried out electronically. We write and edit hundreds of text messages e-mails, blogs and documents. Word prediction is an essential part of every text based modern applications. Because it lets the user write continuously without mistakes. We are going to mimic the same concept in our little desktop application. This is done in the following steps.

1. Insert a document of English words in a data structure which is easy to make insertions and efficient for prefix searching.
2. After selection of the best data structures for the task, through file handling a huge vocabulary of words is inserted into the data structure.
3. After implementation of this data structure the word is searched in the console program.
4. Development of interactive user interface is done after development of the console program
5. The console program is then implemented in the user interface through event handling.

Solution to the problem:

Trie is the data structure which is best for prefix searching.it stores the data in the form of characters and marks the end of a string with a simple Boolean variable isend. Searching is easy in this data structure. After this we looked for a huge file of English vocabulary word. We found one of 450k words. We file handled it into our program. And inserted it into the tree.

After searching for words which was a success on console program we moved towards the development of User interface. We used javaFx and scenebuilder for the development of the interface. We made objects of the console program and implemented the program in the user interface.

**Algorithms:**

We developed the program in functional as well as object-oriented paradigm. So, there are many algorithms. Main algorithm and then one for each of the procedures used. Here’s the list of algorithms in the order that they were used in the program itself.

1. Main program
2. Autocomplete
3. Getwords
4. Getchild
5. Tostring
6. Insert
7. Search
8. Main program:

**( void MAIN () )** This algorithm passes strings to the trie structure and get them inserted. A file is being read for words.

**STEP1:** [initialize] BUFFEREDREADER BR = NULL.

**STEP2:** BR = NEW BUFFEREDREADER (NEW FILEREADER("WORDS\_ALPHA.TXT"))

**STEP3**: [initialize] String line

**STEP4:** until LINE=BR.READLINE())! =NULL

1. T. INSERT(LINE)

**STEP5**: [initialize] string PREFIX

**STEP6:** LIST<STRING> A= T. AUTOCOMPLETE(X) , String[] Array = new String[a.size()]

**STEP7:** [initialize counter] INT I = 0

**STEP8**: until I<A.SIZE() repeat A

1. ARRAY[I]=A.GET(I)

**STEP9**: I=I+1 [incremented]

[end of loop]

**STEP10**: [FOREACH LOOP] STRING LIST: ARRAY then,

Write LIST

1. Autocomplete

**(List<String> AUTOCOMPLETE (String prefix))** This procedure takes a string as an argument PREFIX and returns a list of strings.

**STEP1**: [initialize] A trie node LASTNODE: =ROOT [where ROOT is the actual root of the trie.] integer I: =0 [counter]

**STEP2**: [increment counter] Set I=I+1.

**STEP3:** [test counter] If I<prefix. Length () [length of prefix], then:

1. LASTNODE: = LASTNODE.GETCHILD(PREFIX.CHARAT(I)) [get child is a procedure \* defined later, this procedure takes in each and every character from the PREFIX string entered by the user. The string is converter to characters by CHARAT java function.
2. If LASTNODE: =NULL then,

Return new ARRAYLIST<string> () [if the trie is empty then the procedure returns a new Array List of strings.]

1. [end of if].

**STEP4**: [repeat loop] go to step 2

**STEP5:** Return LASTNODE.GETWORDS() [this procedure \* gathers related words from the trie structure and returns them in the form of array list.

1. Getwords

**(List<String> GETWORDS ())** This subroutine when called does traversing in the trie structure. It looks for each character and when reaches the Boolean variable ISEND ends the particular string. And returns the list of such strings back to the Calle. The traversal is preorder.

**STEP1: [**create a java list\*\* of datatype string and initialize it with a new object of ArrayList\*\* of built-in java class] List<String> list = new ArrayList<String> ()

**STEP2:** If ISEND [Boolean variable marks the end of a string in the trie structure] is true then,

List. Add(toString()) [This procedure is built-in java list function it adds the object to the list, the argument passed is toString which is also an overwritten method\* toString method converts the characters back to the string and return them.]

**STEP3:** [preorder traversal] if CHILDREN is not equal to NULL then,

1. **[**initialize]integer I: =0 [counter]
2. [increment counter] Set I=I+1.
3. [Test counter] if I<CHILDREN.SIZE() [ this function returns size or number of elements in he linked list] then,

If CHILDREN.GET(I) is not equal to NULL then,

List.addAll(CHILDREN.GET(I). GETWORDS ()) [The get method is used to obtain or retrieve a particular variable value from a class addAll appends the words in the end of the list. This is a recursive procedure]

1. [repeat LOOP] go to step 2.

**STEP4:** Return list.

1. Getchild:

**( TrieNode GETCHILD (char c) )** This algorithm receives characters C of a PREFIX string entered by the user. It returns child nodes after searching for the received characters in the trie structure.

**STEP1: [**initialize]CHILDREN [A linked list structure that has datatype TrieNode which has information in char DATA. Parent node and a Boolean is End variable.]

**STEP2**: If CHILDREN is not equal to NULL, then

1. [for-each loop] TRIENODE EACHCHILD: CHILDREN [the for-each loop will run until each and every node of the children LinkedList has been traversed
2. If EACHCHILD.DATA == C [the data of the linked list is compared to the characters from the PREFIX, if they match they are returned] then,

Return EACHCHILD.

1. [end of if.]

**STEP3:** Return NULL [ if CHILDREN LinkedList if empty]

1. Tostring

**(String TOSTRING ())** this procedureoverrides ToString methodof java and returns either empty string or string representation of the characters.

**STEP1:** if PARENT==NULL then, return “”.

Else, return PARENT.TOSTRING() + NEW STRING (NEW CHAR [] {DATA}) [this statement returns string which is a concatenated parent and the string form of all the characters that match with the user entered STRING.]

**STEP2:** [end of if structure]

1. Insert

**(void INSERT (STRING WORD))** This method inserts strings WORD in the trie after converting them into characters.

**STEP1:** if SEARCH(WORD) == TRUE [search\* is a Boolean procedure which searches the word in the trie structure and return true or false] then, return

**STEP2:** [initialize] TrieNode CURRENT: =ROOT TrieNode PRE **[**root is the starting node of a trie]

**STEP3:** [for each loop] (char CH: WORD.TOCHARARRAY()) [ the string WORD is converted to characters and until all the characters are traversed.

1. PRE: = CURRENT
2. TrieNode CHILD = CURRENT.GETCHILD(CH).
3. If CHILD is not equal to NULL then,
4. CURRENT: =CHILD
5. CHILD.PARENT: = PRE

Else

1. CURRENT.CHILDREN.ADD (NEW TRIENODE(CH))
2. CURRENT = CURRENT.GETCHILD(CH)
3. CURRENT.PARENT = PRE

**STEP4**: [END OF FOR EACH LOOP]

**STEP5**: CURRENT.ISEND = TRUE

1. Search:

**(Boolean SEARCH (STRING WORD))** this procedurereceives an argument WORD and looks for the word in the trie structure after converting it into characters.

**STEP1:** [initialize] TRIENODE CURRENT: = ROOT

**STEP2**: [for-each loop] CHAR CH: WORD.TOCHARARRAY() [the loop will run till each character of the WORD is traversed]

1. If CURRENT.GETCHILD(CH) == NULL [the root is sent to be searched for each character of the string WORD] then, return false.

Else,

CURRENT = CURRENT.GETCHILD(CH)

1. [end of if]

**STEP3**: [end of for-each loop]

**STEP4**: If CURRENT.ISEND == TRUE [if end of a string in the trie structure is reached] return true

**Step5:** Return FALSE

**CHAPTER 2: FUNCTIONAL REQUIREMENTS**

In this project, following functionalities are performed to attain a our goal:

* File formation
* Insertion in a trie
* Input from user
* Main Functionality
* BindAutocompletion

**File Formation:**

In this part, buffered reader is used to read a text file line by line by using builtin function **readLine()** and stored it in a variable of string data type. Then we call an **insert()** function to insert the words line by line to a trie data structure.

**Insertion in a trie:**

Insertion function takes string as parameter and convert it to the characters. Now we have a alphabets and can easily be stored in a trie . There is variable of bool data type in which after the completion of the word we put true.

**Input from user:**

In this section, input is taken from the user and stored it in the variable of string data type word by word and passed it to the function **autocomplete().**

**Main functionality:**

**Autocomplete()** function passed the characters from a string entered by user to the function **getchild()** in a loop**.** After receiving the characters getchild function will search the received characters in a trie and then returns the child nodes. Then after checking the last node is NULL or not it is decided to return the arraylist or again call the function **getWords().**

**GetWords()** will traverse the trie structure and check the bool variable for each character and when found it true, it will ends the particular string and returns the list of strings.

After receiving the list of strings these strings are stored in an array and that final array is passed to the the function **bindAutoCompletion().**

**BindAutoCompletion:**

BindAutoCompletion() is a built-in function of the JAVAFX which takes the array as a parameter. It takes input in a text field. After entering a character in text field, a user will get a list of words as suggestions that can be form by that character so that the user can easily choose the word which he wants to type. After typing two characters, there will only be the suggestions of the words started by these two characters. So list of strings are received after calling a autocomplete() function that are stored in an array and that array is passed to the BindAutoComplete() which will show the predictions of the charater entered by the user.

**CHAPTER 3: LANGUAGE AND DATA STRUCTURE**

**Introduction to language:**

Any software, application or program development in computer science requires selection of a programming language. there are many programming languages under use today. Many have been upgraded to newer and better versions. like C programming language got upgraded to c++ and later to C#. Similarly, latest languages that are trending are android and python.

A few languages got left behind because of their old programming paradigms/like functional programming language like C had to be upgraded to be object-oriented language. Now known as C++.

For our desktop application we choose java. Java is an object-oriented language with humongous APIs and free packages for developers.

**Introduction to java:**

Java is a programming language created by James Gosling from Sun Microsystems (Sun) in 1991. The target of Java is to write a program once and then run this program on multiple operating systems. The first publicly available version of Java (Java 1.0) was released in 1995. Sun Microsystems was acquired by the Oracle Corporation in 2010.

JRE (JAVA RUNTIME ENVIRONMENT) is a part of JDK (JAVA DEVELOPMENT KIT) which means that JDK includes JRE. When you have JRE installed on your system, you can run a java program however you won’t be able to compile it. JRE includes JVM, browser plugins and applets support. When you only need to run a java program on your computer, you would only need JRE.

Java Requirements:

To make a java application we need a development tools like JDK. The JDK includes the Java Runtime Environment, the Java compiler and the Java APIs.

**Introduction to javafx**

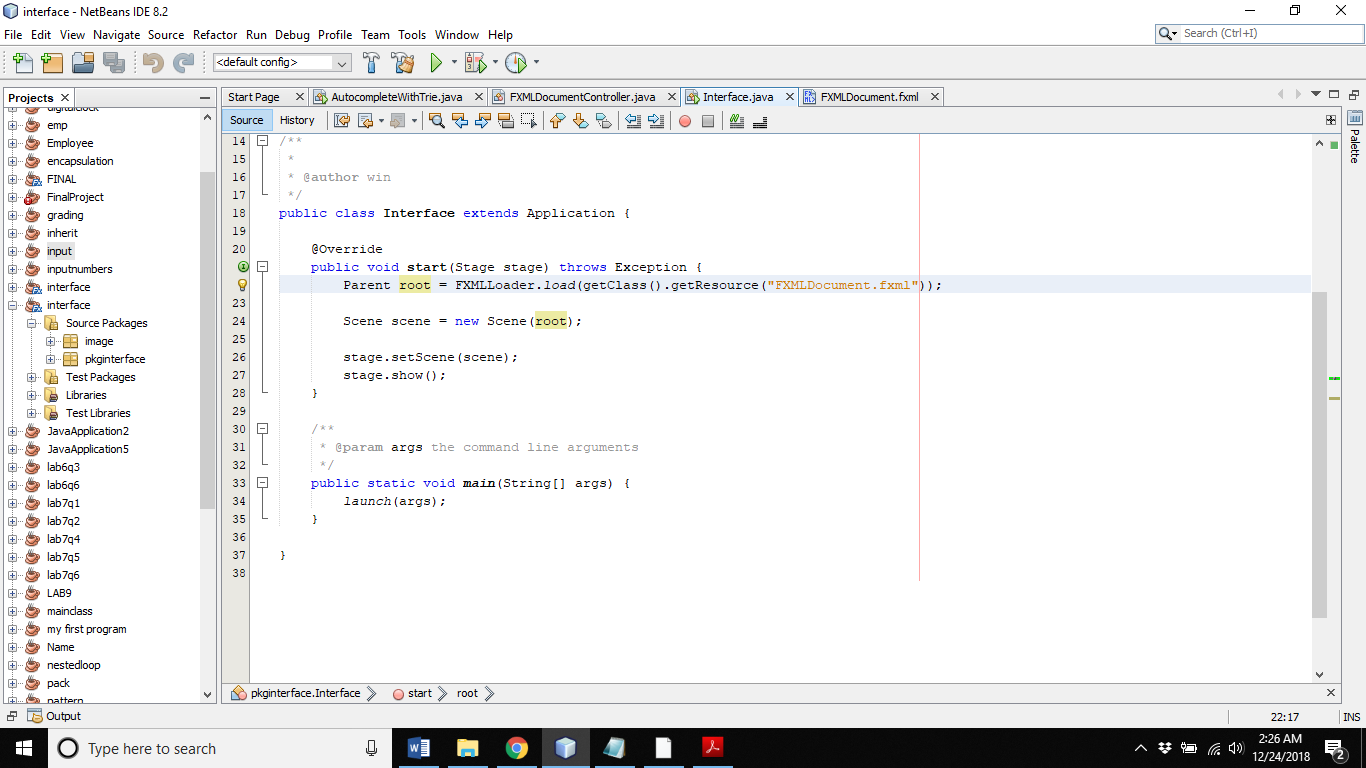
JavaFX:

JavaFX is a [software platform](https://en.wikipedia.org/wiki/Computing_platform) for creating and producing [desktop applications](https://en.wikipedia.org/wiki/Application_software). It has two parts:

Javafx stage

JavaFX Scene

A JavaFX Stage, represents a window in a JavaFX desktop application. Inside a JavaFX Stage you can insert a JavaFX Scene which represents the content displayed inside a window. First of all, we have to create a stage, show a stage and then set a scene on a stage. Here is the screenshot of the use of javafx stage and scene:

****

JavaFX FXML:

JavaFX FXML enables to compose JavaFX GUIs in HTML. FXML enable to separate your JavaFX layout code from the rest of your application code. Here the steps to create an FXML file:

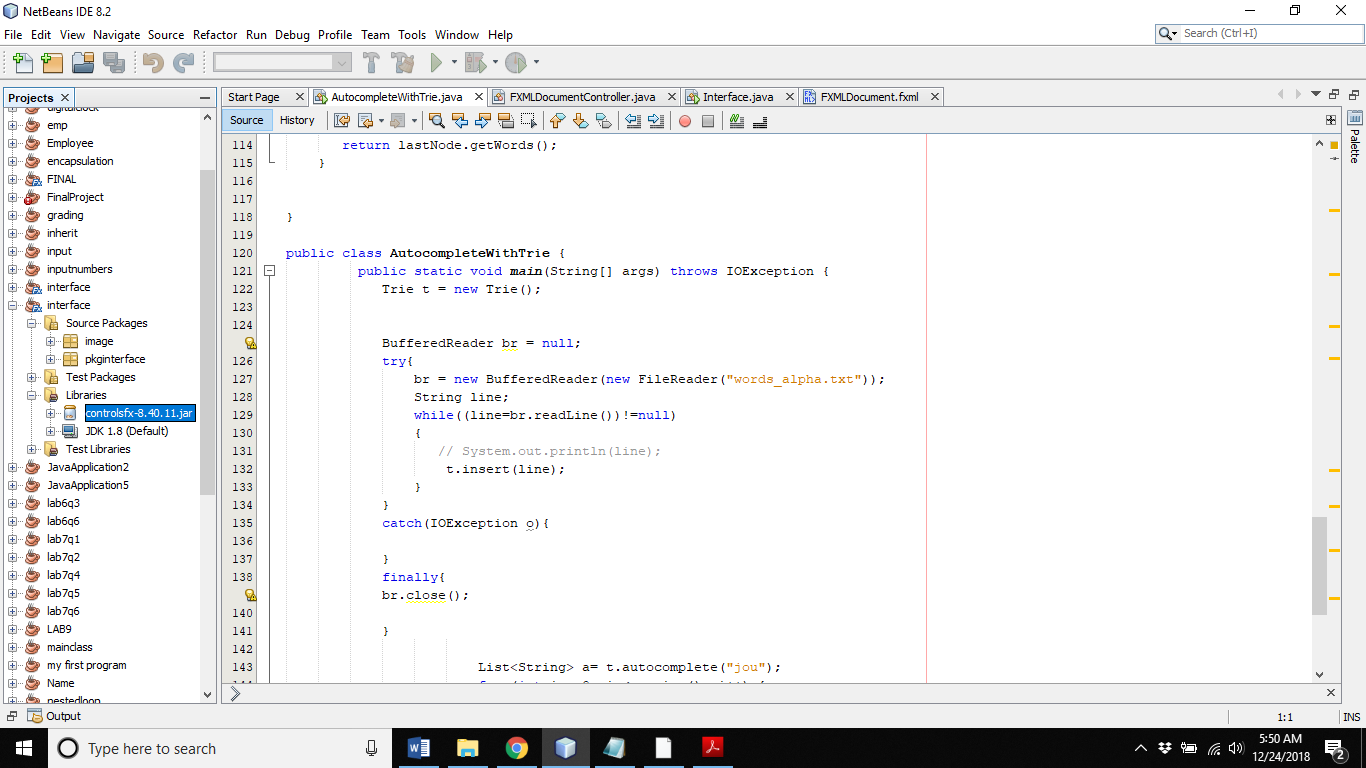
* Loading an FXML File
* Importing Classes in FXML
* Creating Objects in FXML

Properties in FXML:

Some properties of the FXML file are:

* FXML Namespace
* FXML Element IDs
* FXML Event Handlers
* FXML CSS Styling
* FXML Controller Classes

## Library used in FXML:

****JavaFX libraries are used, when something that you often need in a project. Controlsfx is an [open source project](http://code.controlsfx.org/) for JavaFX that aims to provide really high-quality user interface controls and other tools to complement the core JavaFX distribution. Controlsfx 8.40 11 library is used in this project in building the Graphical user Interface. Here is the screenshot of the library used:

JavaFX Scene Builder:

# ***Figure 1: Library used***

JavaFX Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding. Users can drag and drop UI components to a work area, modify their properties, apply style sheets, and the FXML code for the layout that they are creating is automatically generated in the background. The result is an FXML file that can then be combined with a Java project by binding the UI to the application’s logic. The JavaFX Scene Builder tool simplifies the discovery of these different aspects by using and manipulating the JavaFX components directly from the tool.

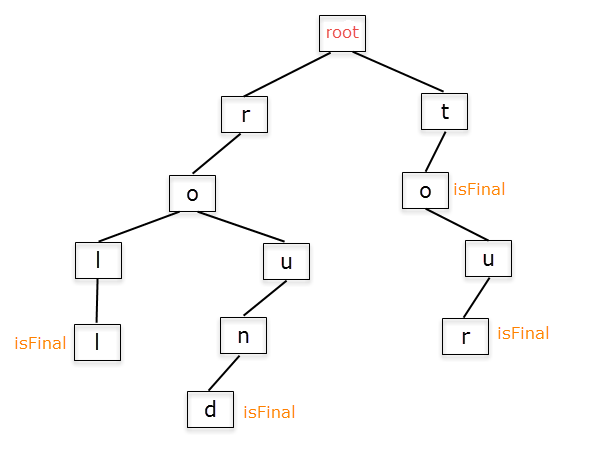
**Introduction to data structures used:**

Data structure as the name suggest is the structure used to store data. Huge amounts of data needs be handled every single day in the world. We need organizational systems for this data. Programming data structures are so used to store, traverse and organize the data. From Data type of the data to the structure best suitable for the particular data are the requirements for organization of that data. Our program to needed huge data handling. Also, because there was prefix searching involved.so the most suitable basic data structure was the trie data structure. Other structures are also used here are the few famous ones.

1. **Trie:**

Definition:

Trie is an ordered tree structure, which stores keys usually strings. Every node of Trie consists of multiple branches. Each branch represents a possible character of keys. We need to mark the last node of every key as end of word.

Template to make Trie node:

struct TrieNode

{

    TrieNode \* children[size];

    bool END;

};

Applications of Trie Data Structure:

We use Trie Data Structure in Several Applications having specific features. Some of them are mentioned below:

* Pattern Searching
* Word Games
* Length of word
* Print words in Alphabetical order
* Prefix search (Autocomplete)

Use of Trie in our Code:

Trie data structure can store the multiple strings. In the form of branches, the words are stored and the address of each node is stored in a doubly linked list so that can access any node easily. So, by using the bool variable we can stop traversing by reaching the end of the word.

1. **Doubly LinkedList**

Definition:

Doubly linked list (DLL) is a complex type of linked list contains an extra pointer than in a simple linked list, in which a node contains two address pointers, one to the previous and second to the node forward to it. Therefore, in a doubly linked list, a node consists of three parts:

* Node data value,
* Pointer to the next node (next pointer),
* Pointer to the previous node (previous pointer).



Template to make a node for doubly liked list:

struct Node  {

int nodeData;

Node \*Forward;

Node \*Backward;

};   Node \*head;

Memory Representation of a doubly linked list:

Generally, doubly linked list consumes more space for every node and therefore, causes more expansive basic operations such as insertion and deletion. However, we can easily manipulate the elements of the list since the list maintains pointers in both the directions (forward and backward).

Use of doubly Linked List in our Code:

Doubly linked list is used in making the tree nodes. As doubly linked list can store the address of the nodes forward and backward to the current node. So in a trie formation we have to use the doubly linked list so can easily access the nodes forward and backward to it.

1. **ArrayList:**

Definition:

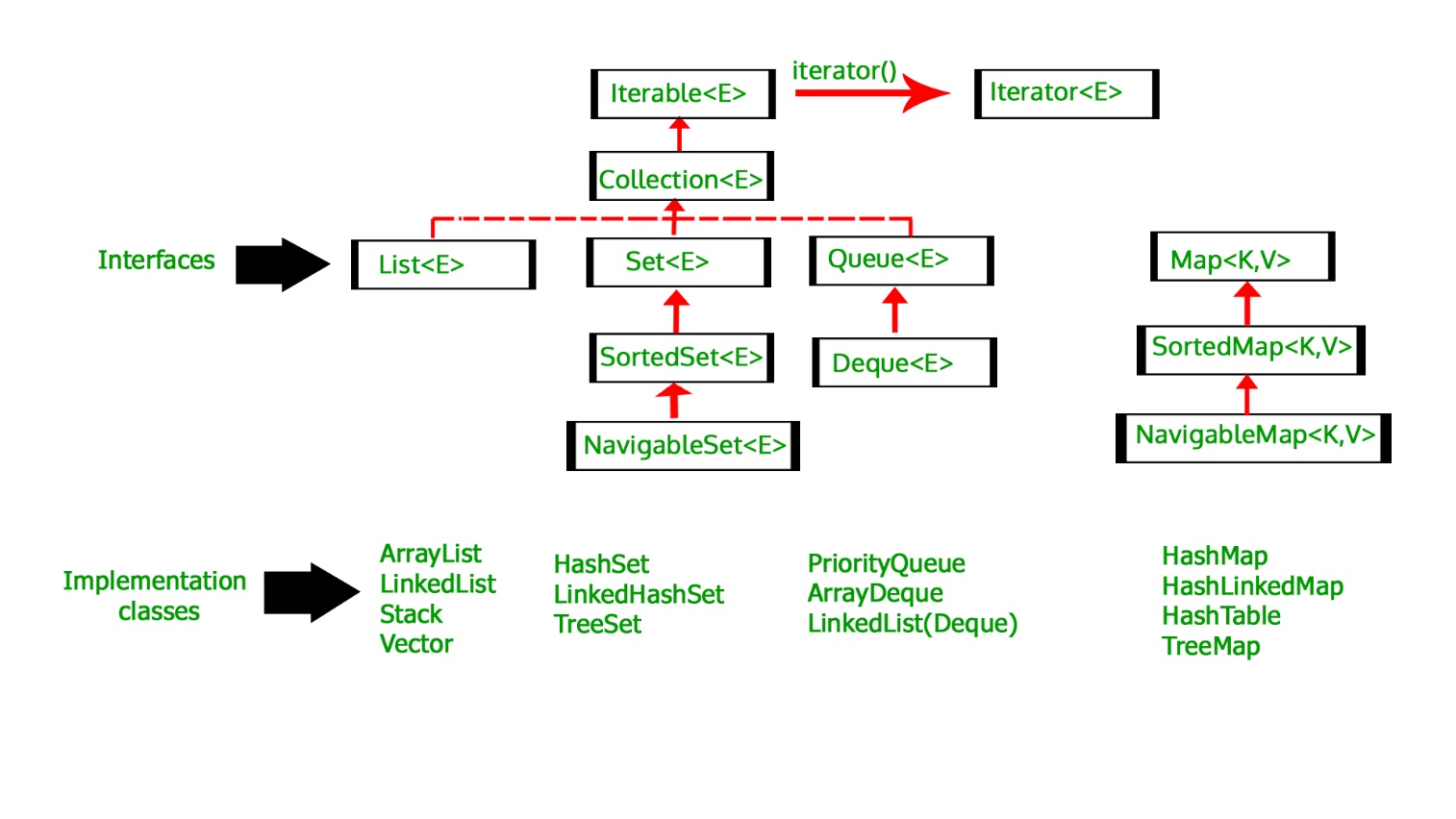
ArrayList is a part of [collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) and is present in java.util package. It provides us dynamic arrays in Java (link).there are different operations that you can perform on a arraylist. There are above 28 procedures that you can perform to an arraylist of java here are a few used in our program:

1. forEach​(Consumer action)
2. get​(int index):
3. size​()
4. void add(int index, Object element)
5. boolean addAll(Collection C):

Template to make an Arraylist:

ArrayList<Integer> arraylist = new ArrayList<Integer>();

Here’s a diagram of arraylist class from java built-in classes

****

1. **Lists:**

Definition:

The Java.util.List is a child interface of [Collection](https://www.geeksforgeeks.org/collections-in-java-2/). It is an ordered collection of objects in which duplicate values can be stored. (htt5)

Template to make an Array:

List<Object> list1 = new List<Object> ();

Functionality:

One can positionally access a list.it allows a wide range of functions. Searching and iteration and range-view are its major functions. The ones that are used in our program are

* **void add (int index, Object O)**
* **Boolean addAll (int index, Collection c)**

Important part to note is that there’s a difference between arraylist and list in java. The important difference is List is an Interface and ArrayList is a standard Collection Class. Also, list is used to create a list of elements or objects which are associated with their index numbers. ArrayList is used to create a dynamic array that contains objects. (htt6)

1. **Arrays:**

Definition:

Array is a container which can hold a fix number of items and these items should be of the same data type. Array can contain primitive’s data types as well as store the addresses of objects of a class depending on the way we are defining that array. In case of primitive’s data types, the actual values are stored in defined memory locations however, in case of objects of a class [the actual objects are stored in heap memory](https://www.geeksforgeeks.org/g-fact-46/). There are two types of Arrays:

* One-dimensional Array
* Two dimensional Arrays like matrix.



Template to make an Array:

Int[] ArrayNumbers = new int[]{2,3,5,7,8,6,8};

**CHAPTER 4: FLOWCHARTS AND GUI**

**Introduction:**

This chapter includes flowcharts, class diagrams and screenshots of our project. Flow charts shows that how control moves in our program and the functionalities that we provide, and how it all works.

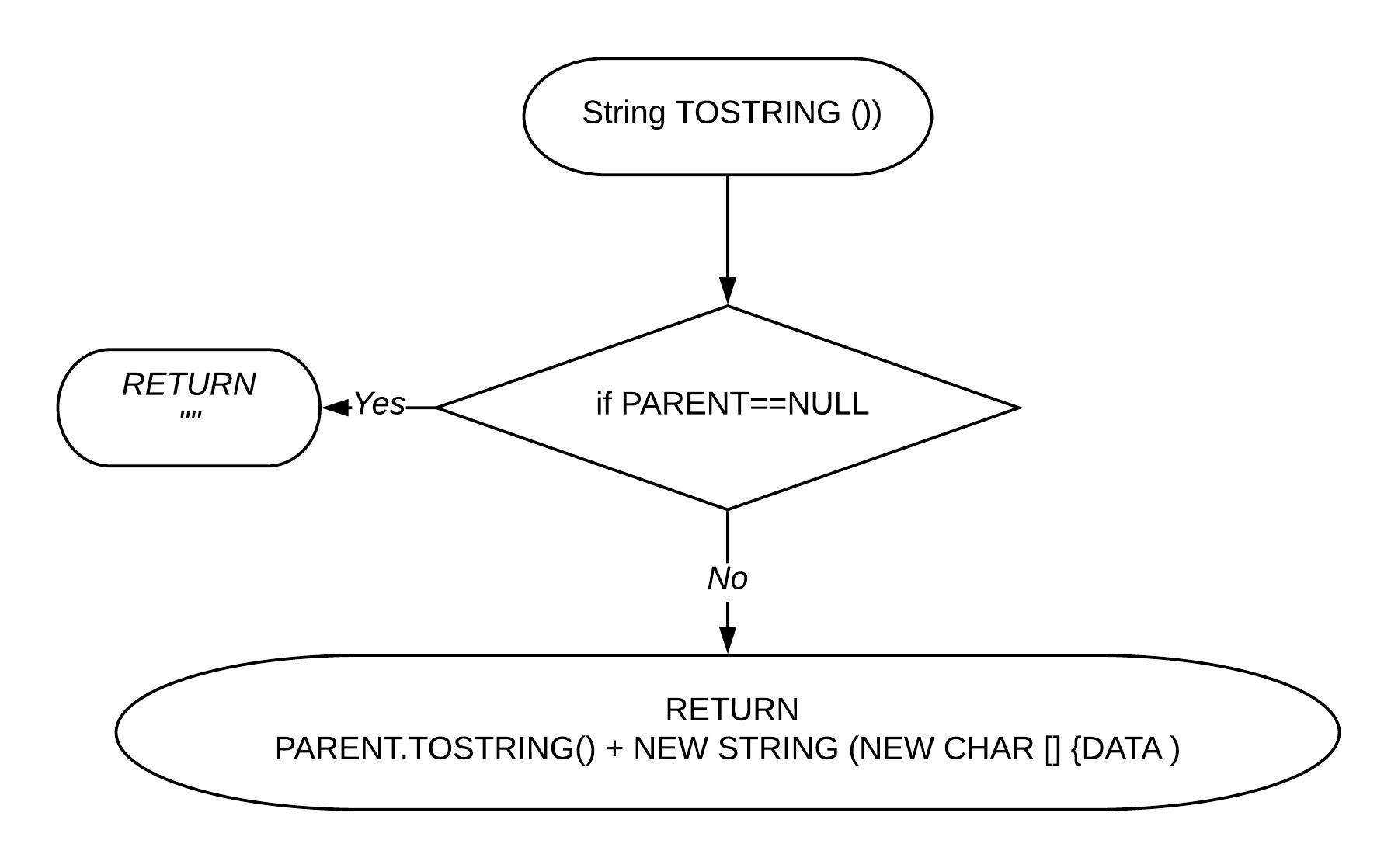
The class diagram is the basic structure of our project that how we assemble our data and work in OOP.

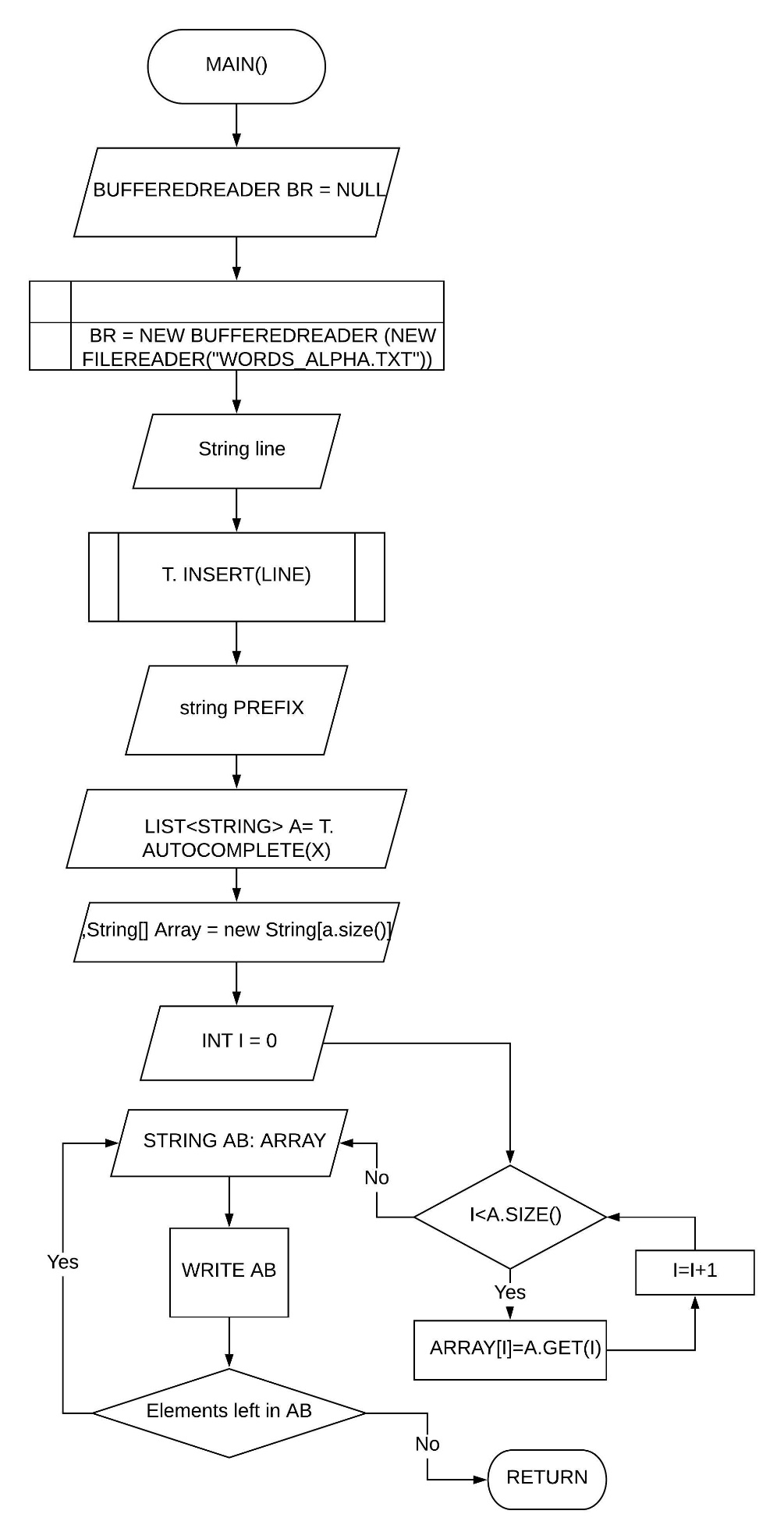
The GUI screen shots give you the different look of our program in different input and outputs.

**Flowcharts**

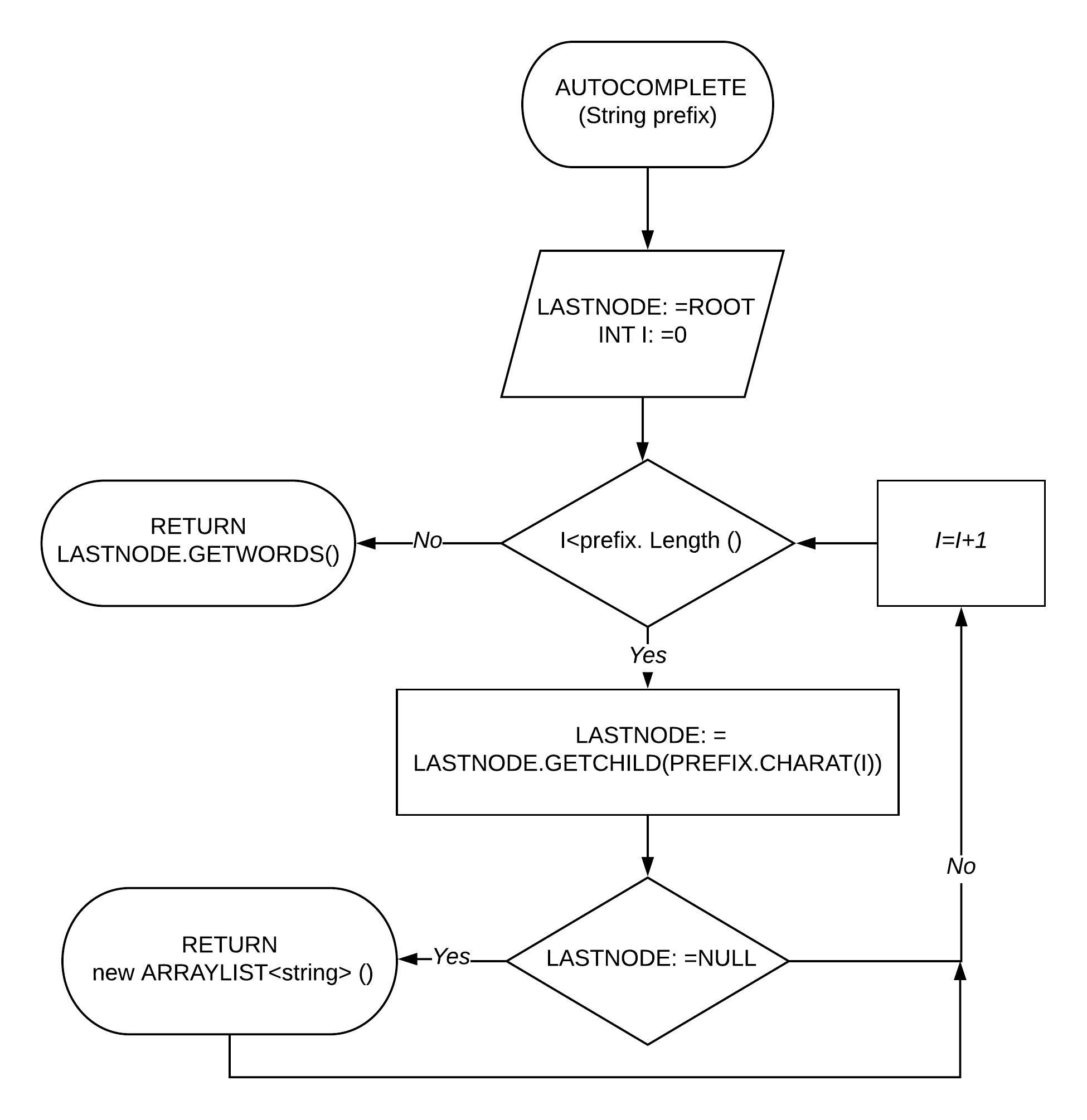
The flowcharts of our program are of the subroutines that are involved in our program.

1. Tostring
2. Main program
3. Autocomplete
4. Getwords
5. Getchild
6. Insert
7. Search
8. Tostring

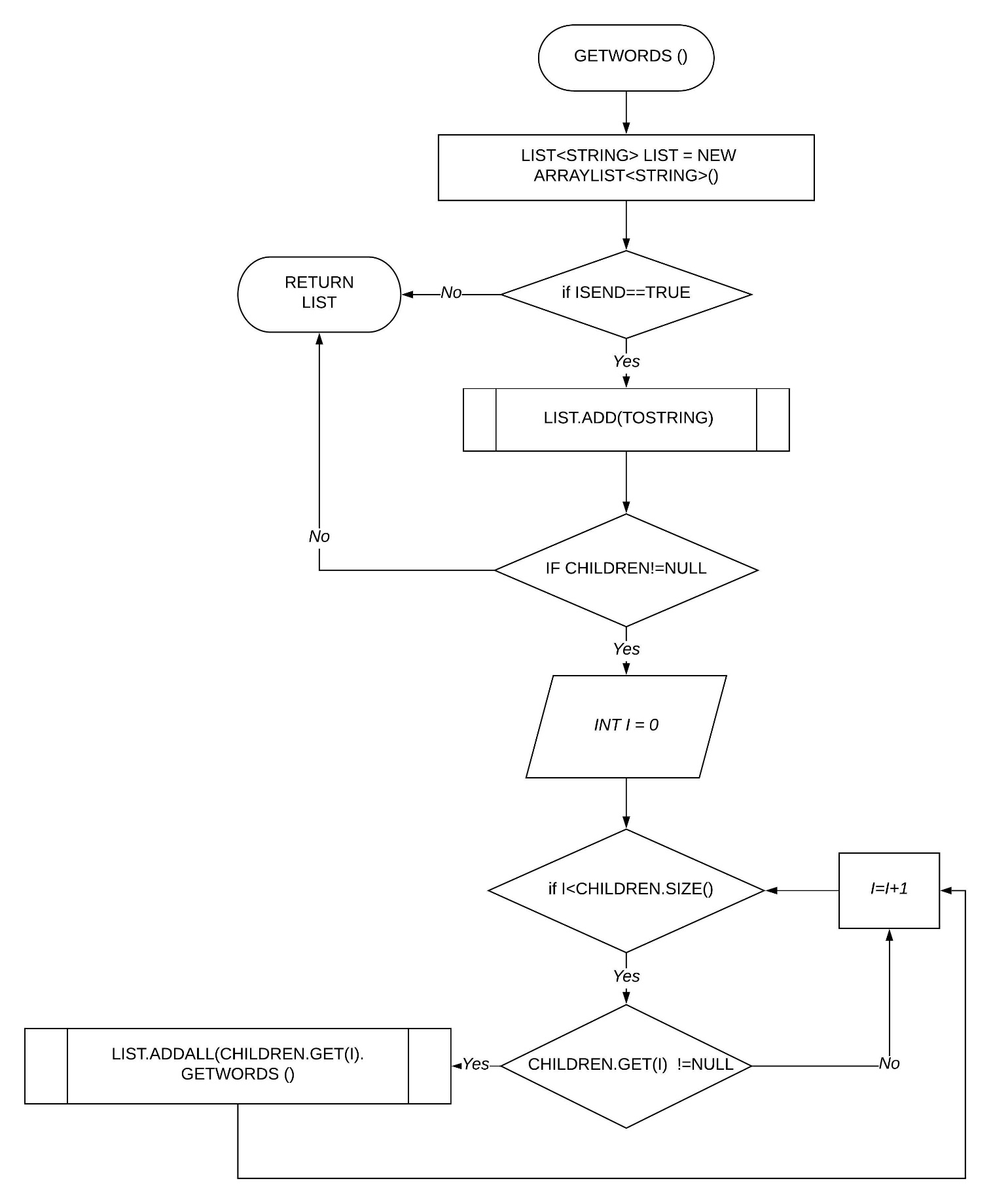


1. Main program

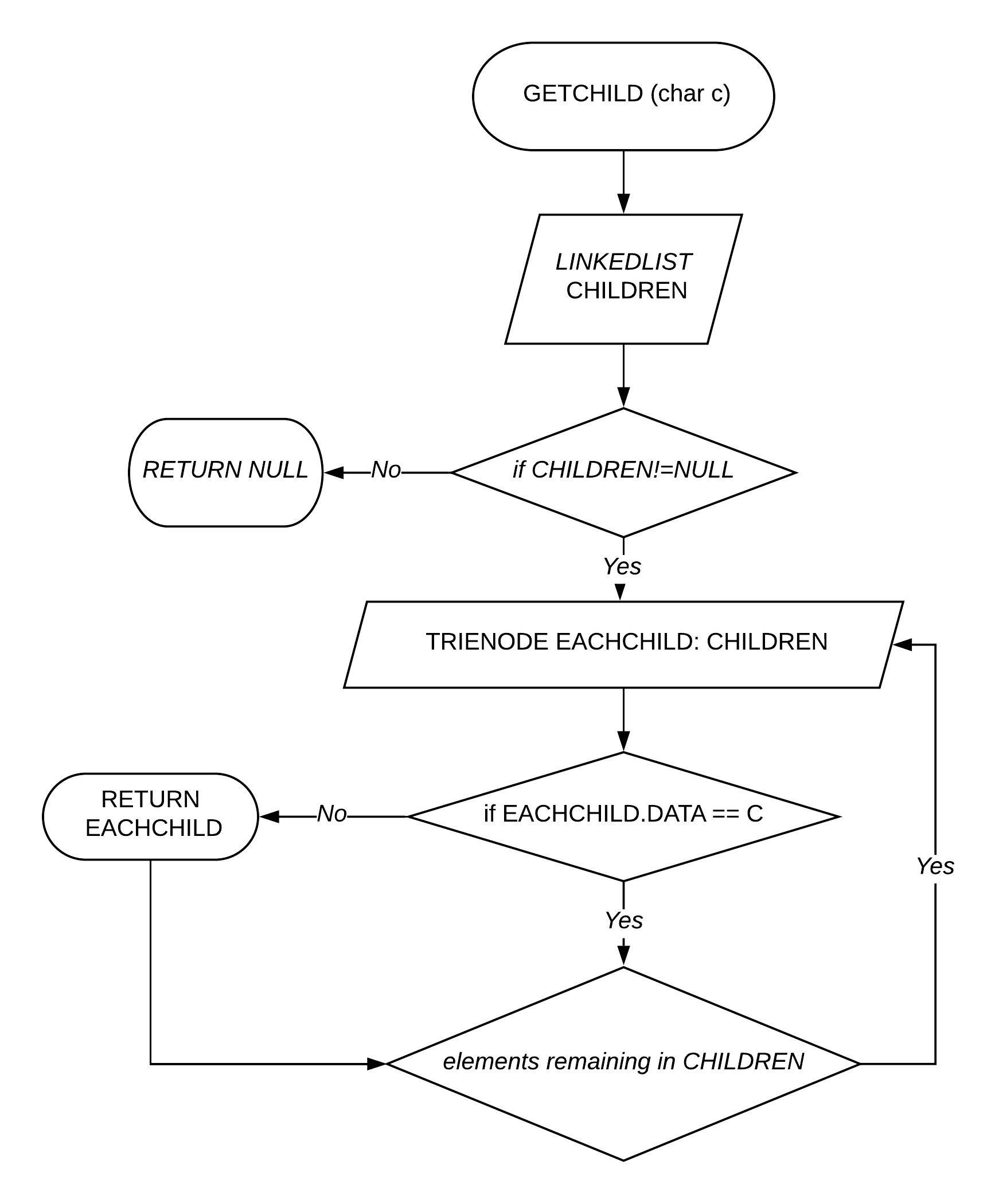
1. Autocomplete



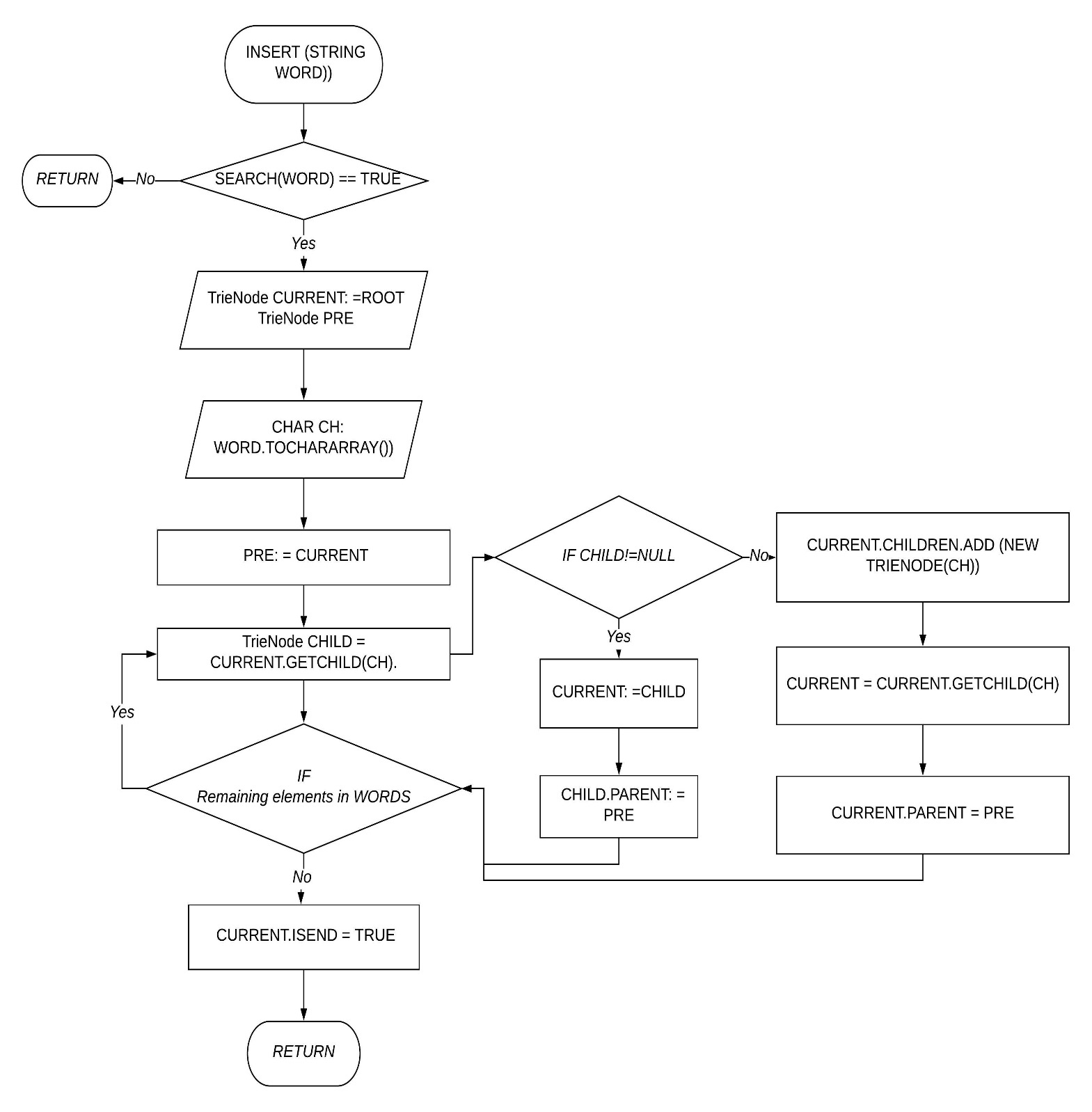
1. Getwords



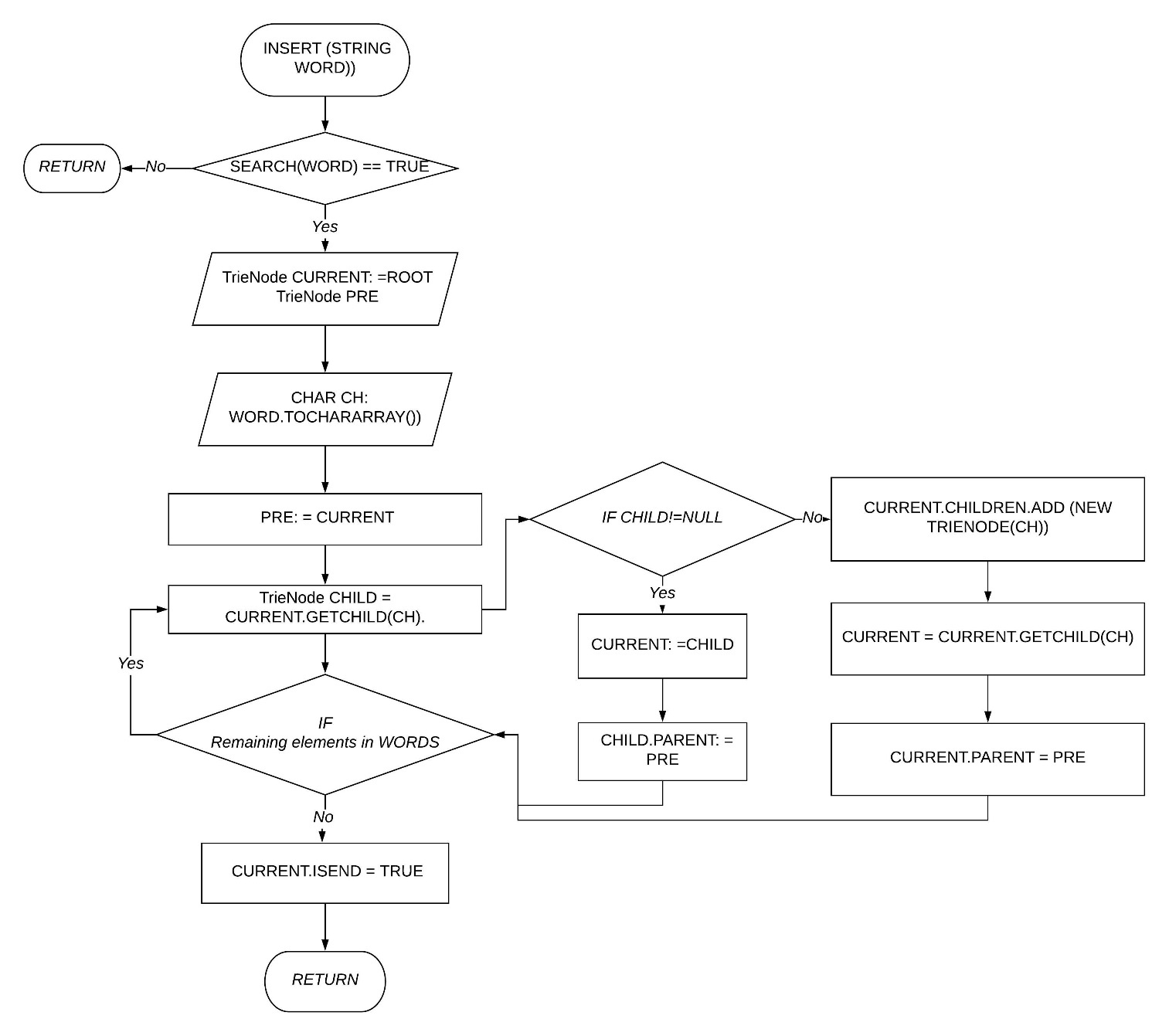
1. Getchild



1. Insert



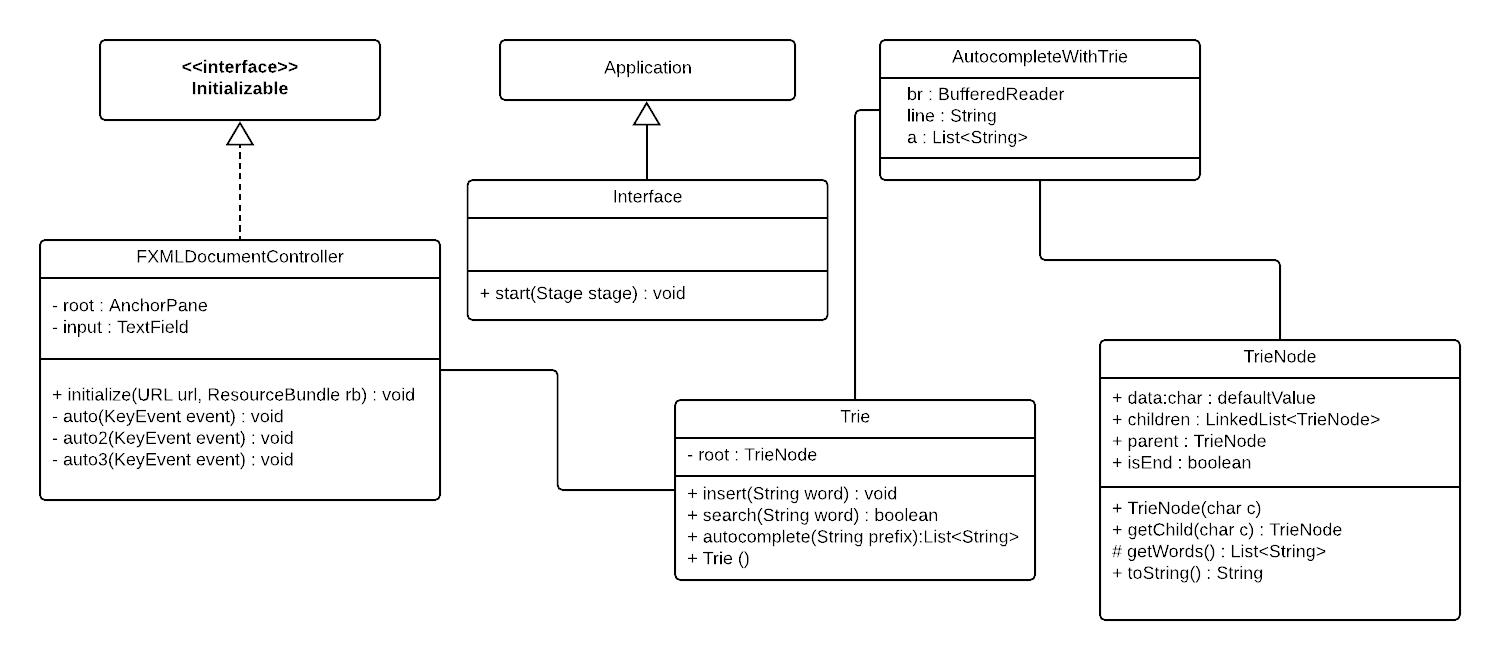
1. Search



**Class Diagrams**

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. (web)

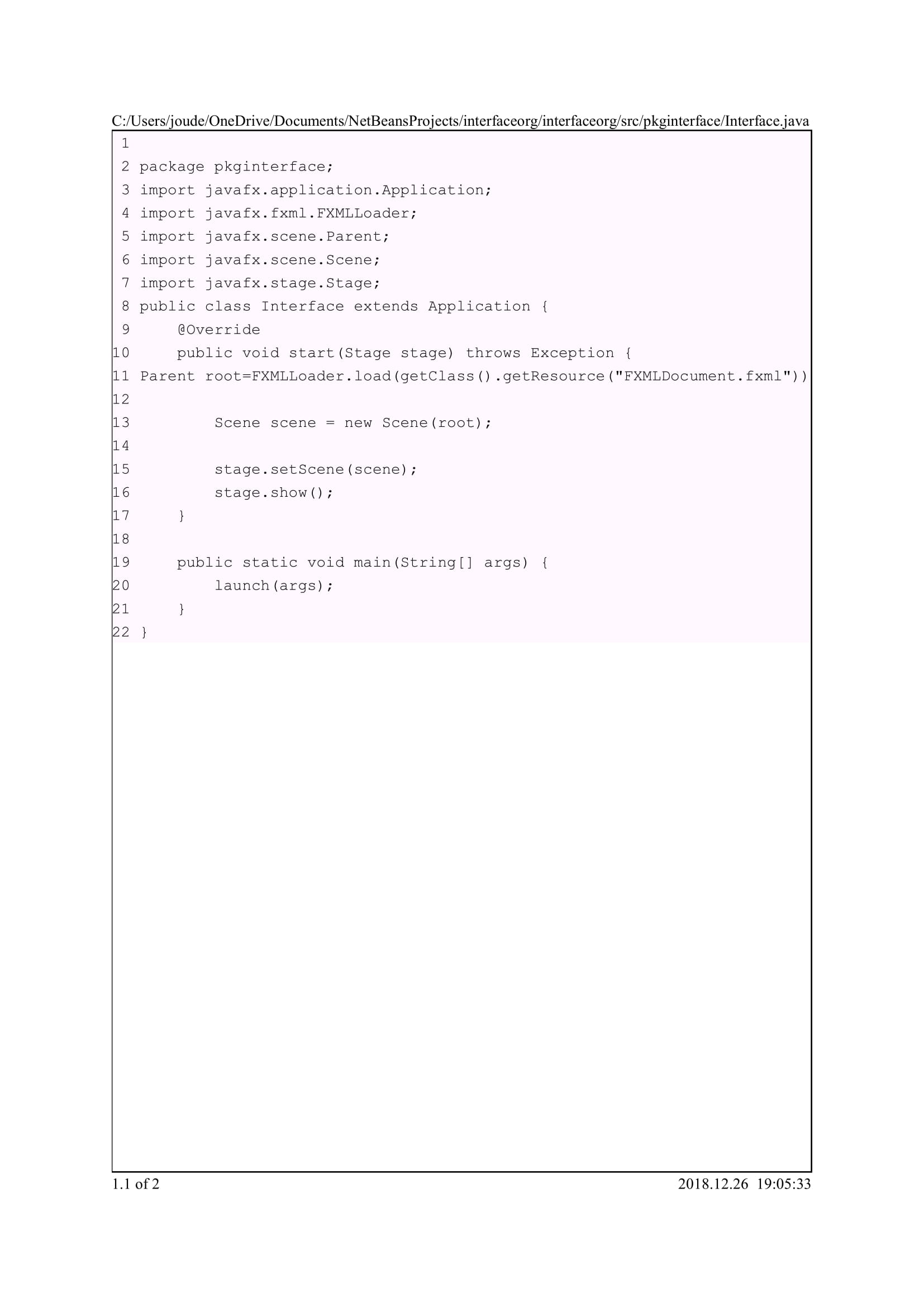
Here is the class diagram of our user interface:

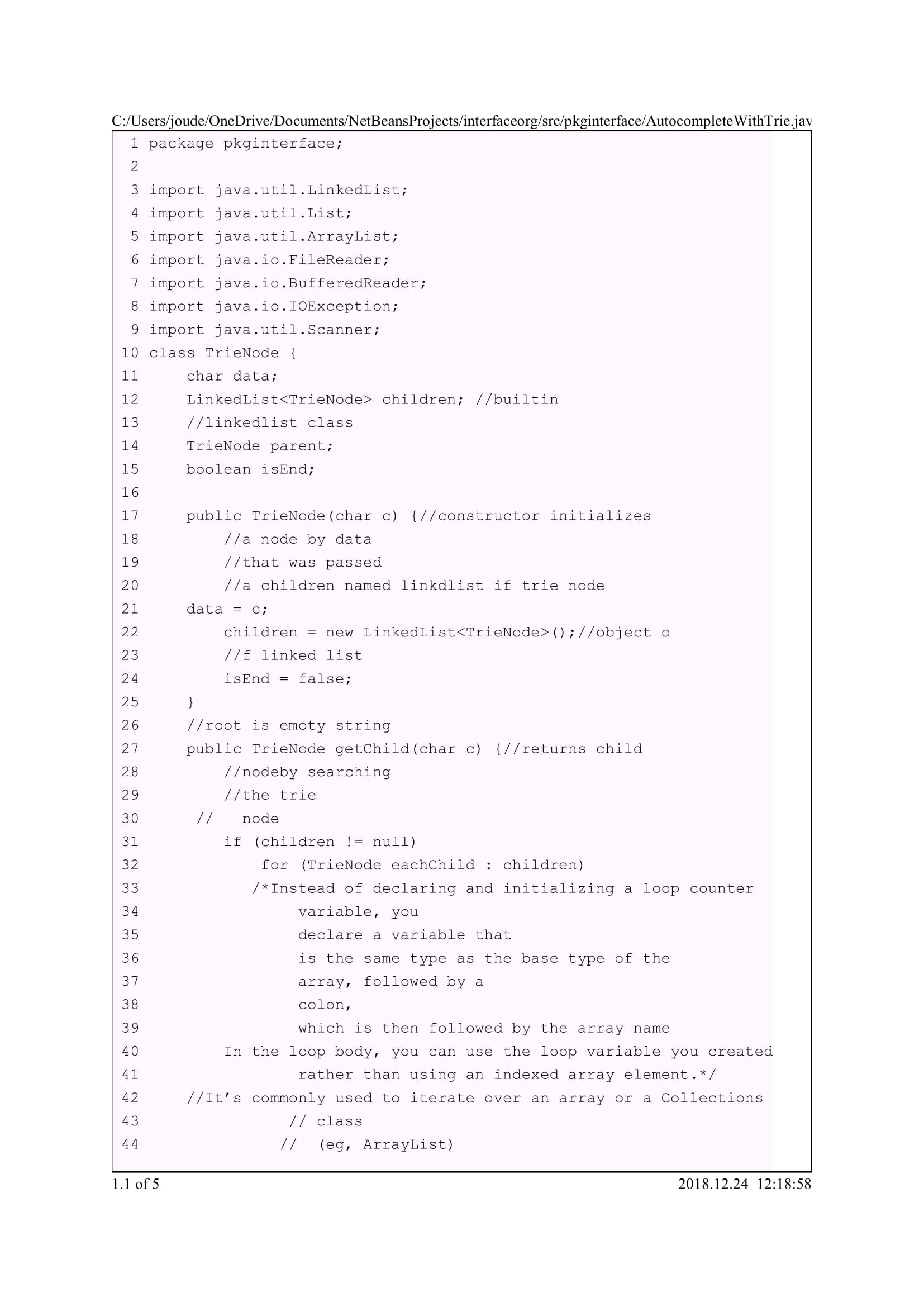


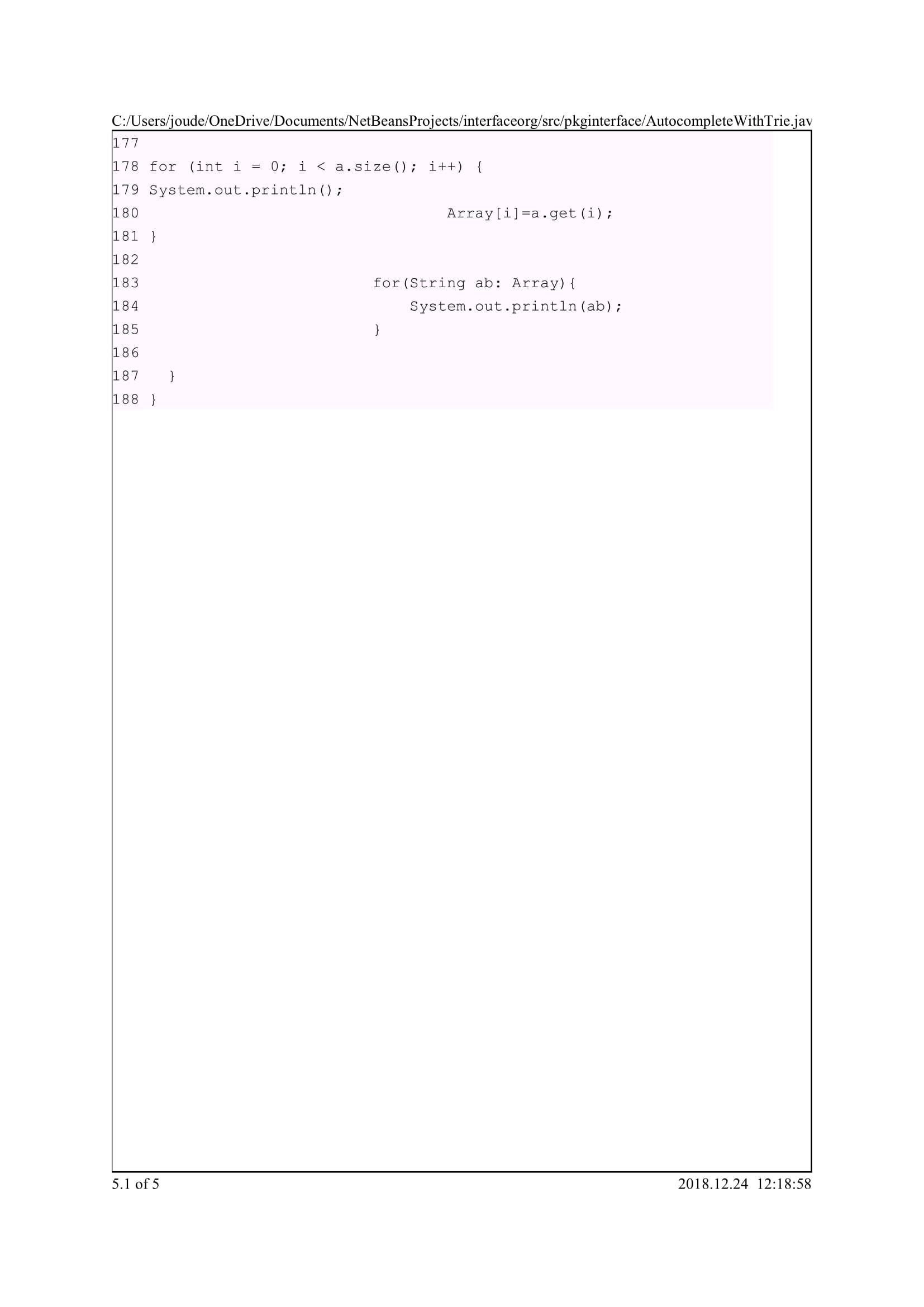
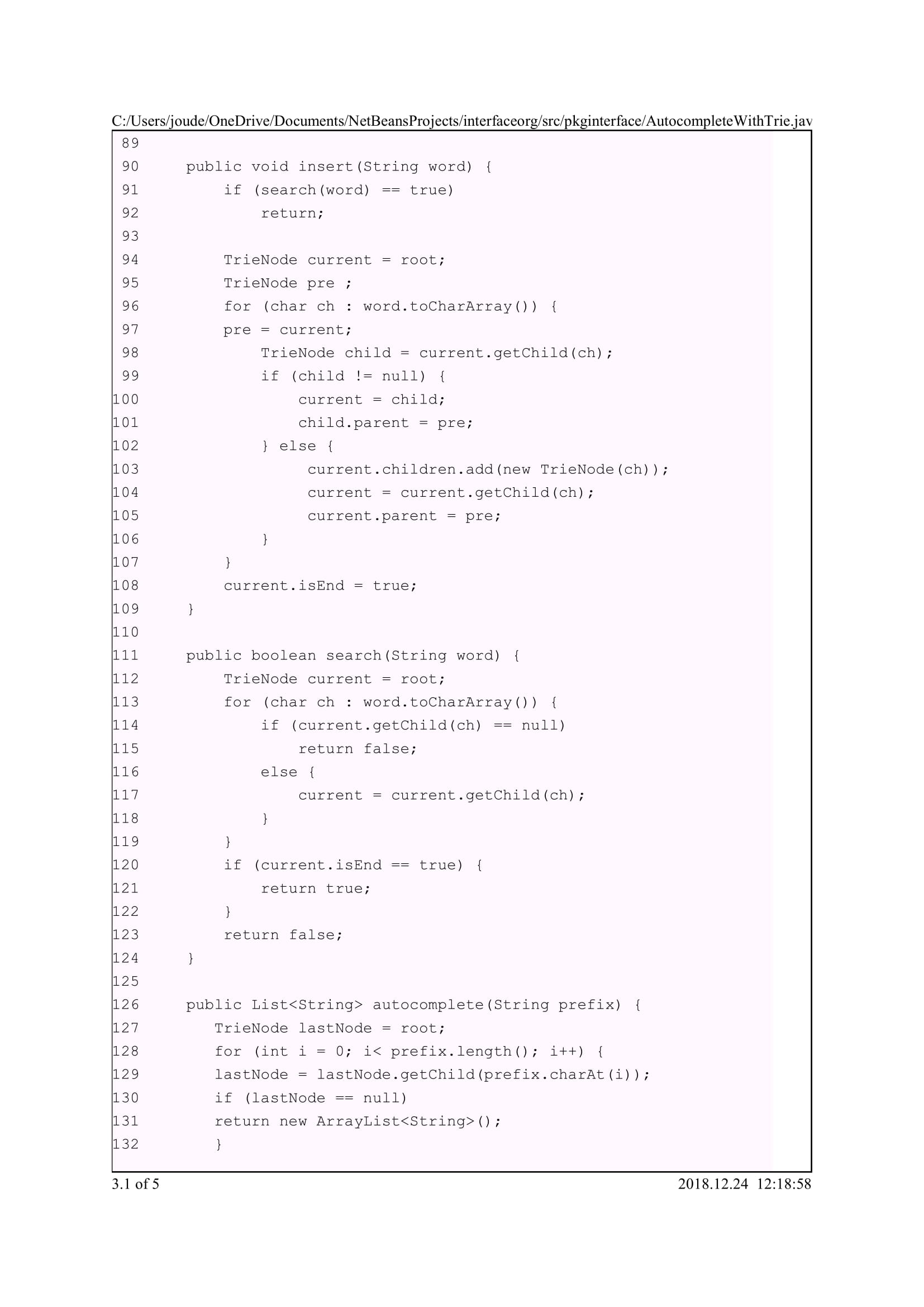
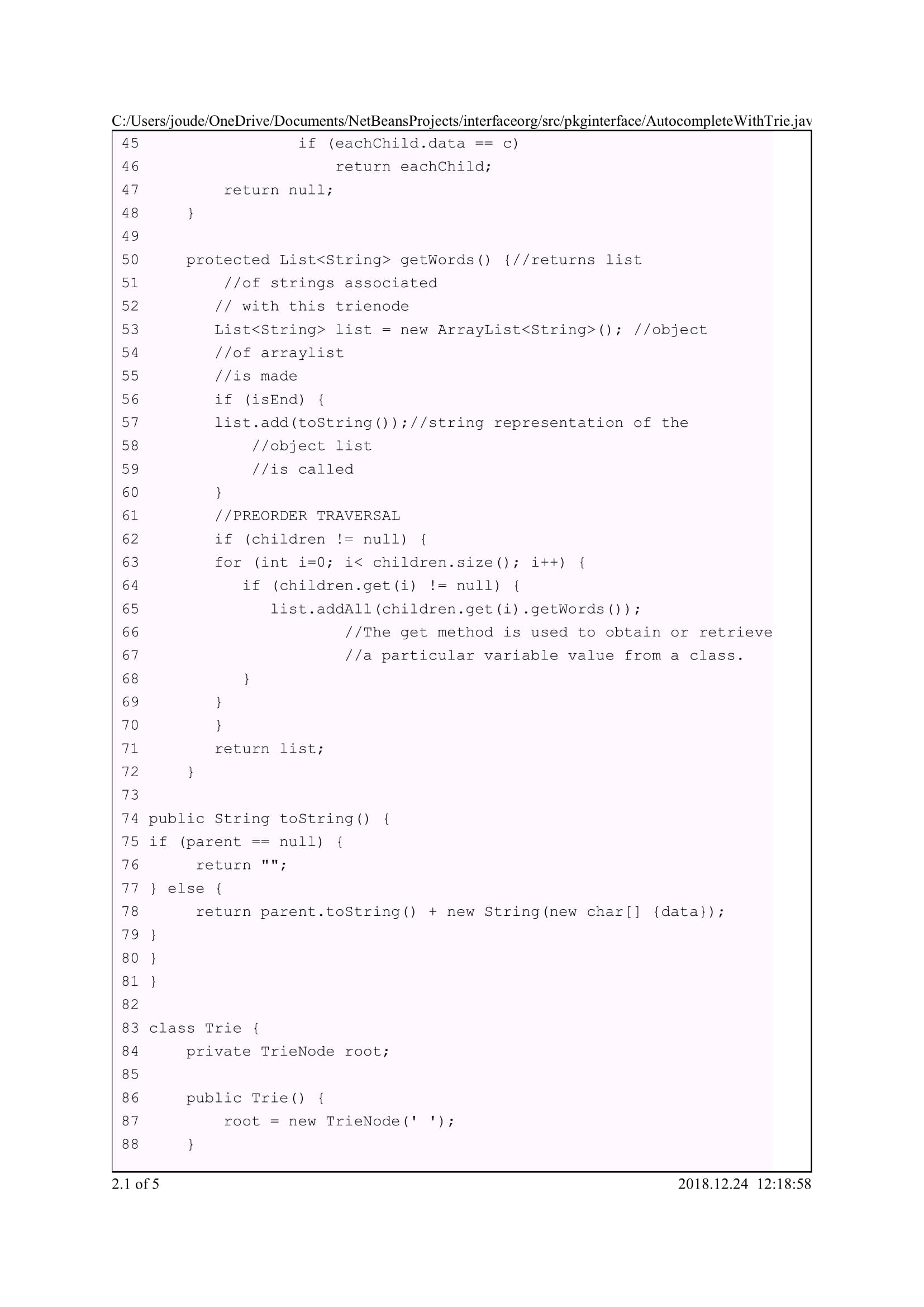
Total five classes are used in this project. AutocompleteWithTrie, TrieNode and Trie are the classes of our main project while Interface and FXMLDocumentController are the classes for making the Graphical User Interface. Interface inherits from the Application class which is Java built-in class and FXMLDocumentController implements the Initializable interface. Inheritance allows the all the methods and variables of parent class to be used in child class.

**CHAPTER 5: CODE**

Heres the program code of our projects







**CHAPTER 6: REFERENCES**

(n.d.). Retrieved from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

(n.d.). Retrieved from https://www.geeksforgeeks.org/arraylist-in-java/

(n.d.). Retrieved from https://www.geeksforgeeks.org/list-interface-java-examples/

(n.d.). Retrieved from https://techdifferences.com/difference-between-list-and-arraylist-in-java.html

<http://fxexperience.com/controlsfx/>

<http://tutorials.jenkov.com/javafx/fxml.html>

<https://www.java.com/en/download/faq/develop.xml>

<https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm>

<https://en.wikiversity.org/wiki/Introduction_to_Programming_in_Java/Introduction_to_Java>