Polytechnic Institute of Coimbra

Instituto Superior de Engenharia de Coimbra

Department of Computer Engineering and Systems

Implementation of a Lexer and a Parser in Java

Pohle Maxime,

Student of Lille 1

DUT Informatique

University Year 2015-2016

List of Contents

List of Figures

List of Tables

Special Thanks

Before to start this final report, I would like to thanks every person who supports me in this project and in my internship.

Firstly, I want to thank professor João Costa, who was my tutor during my work placement; for gave me some clues, answered my questions even if he was busy he always found a moment to help me, answer me during my working period at Coimbra.

I would like to thank Mr. Patrick Lebegue, for his precious advice and the follow up of my work placement.

I would like to thank the International Relations office, particularly Mrs Dália Pires of ISEC and the International office of "Lille 1", for their support and help.

Thanks to all the teachers of the Computer Sciences department , "IUT A", the University of Lille who gave me all the knowledge I needed to do this project.

Introduction

In this chapter, we present the training and expose the initial proposal of the internship subject and contributions. Also presented is the structure of the report.

The Technology University degree (DUT), a shorter cycle of studies (2 years) after the baccalaureate, allows a quick insertion in the world of work. Indeed, DUT can be used to provide system administrators or databases administrators as well as web, mobiles or software developers.

These days, companies are increasingly looking for an employee with multiple qualifications, like management, economy or the right. That's why the DUT training gives general teachings like communication class next to the computer courses.

The faculty purpose courses of enterprise management, accounting or mathematics for instance to improve the general education of each student. Moreover, throughout the training, we must define ours professional project in order to know what we want to do after the training.

Globally, we have two main choices: moving into the workforce or continuing the studies in the last year of degree then integrate master or engineer school. In general, the students who want to continue their studies choose to carry out the internship abroad to be put into practice the acquired knowledge during the training. And the students who want to work after the DUT complete their internship in IT Services Company to integrate the working world and to be put into practice their knowledge.

For this reason, I chose to carry out my internship abroad, in the Polytechnic Institute of Coimbra. I would like to continue my studies in Computer Sciences training who allows you to train more with different programming languages. Thus, I have made the choice of studying the subject: Implementation of a lexer and a parser in Java.

Today, we need to find other ways to understand and use computers in faster way and without problems.

To program and interact with computers programs we need to express the things that we want and need to do. For that there's the need to have a way to express those things and have computers to understand them. That’s the main function of lexers and parses.

The main problem that many lexers and parser are existing today but we need that this lexer and this parser must be the better. We need a lexer and parser who correspond with the ways we used computers today.

First, I will present my project, by telling the language, tools I used. Finally, I will detail my project, the creation of the lexer and then the creation of the parser.

1. Presentation of my project

To do my presentation of my project, I will describe what is a lexer and a parser. After this, I will detail the language I will use in this project, Java. And to end this presentation, I will tell what are tools I used in this project.

1.1. Lexer and parser

For my project, I need to do a lexer and a parser in Java. I will describe the lexer.

In computer science, lexical analysis is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings with an identified "meaning"). A program that performs lexical analysis may be called a lexer, tokenizer, or scanner (though "scanner" is also used to refer to the first stage of a lexer). Such a lexer is generally combined with a parser, which together analyze the syntax of programming languages, web pages, and so forth.[1]

And now, what is a Parser ?

A parser is a software component that takes input data (frequently text) and builds a data structure – often some kind of parse tree, abstract syntax tree or other hierarchical structure – giving a structural representation of the input, checking for correct syntax in the process. The parsing may be preceded or followed by other steps, or these may be combined into a single step. [2]

And for create this parser and this lexer I will use the programming language named Java.

1.2. Java Language

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

Figure 1: Java Logo

Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2016, Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform.

The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them. Java was the first language that I saw in my studies because it's the language of choice of OOP (Object-oriented programming).

Figure 2: Logo of Oracle, owner of Java

And for use Java, I need to have an IDE (integrated development environment).

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion. [3]

We will see two of them : Eclipse and NetBeans.

1.3. Tools

At the begginning, I used the IDE Eclipse that I will dexcribe just after. And to test another IDE, I choose to pick NetBeans.

a) Eclipse

Figure 3: Eclipse Logo

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. Eclipse was created in November 2001 by Eclipse Foundation. It contains a base workspace and an extensible plug-in system for customizing the environment.

It's written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages through the use of plugins. It can also be used to develop documents with LaTeX (through the use of the TeXlipse plugin) and packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

b) NetBeans

Figure 4: NetBeans Logo

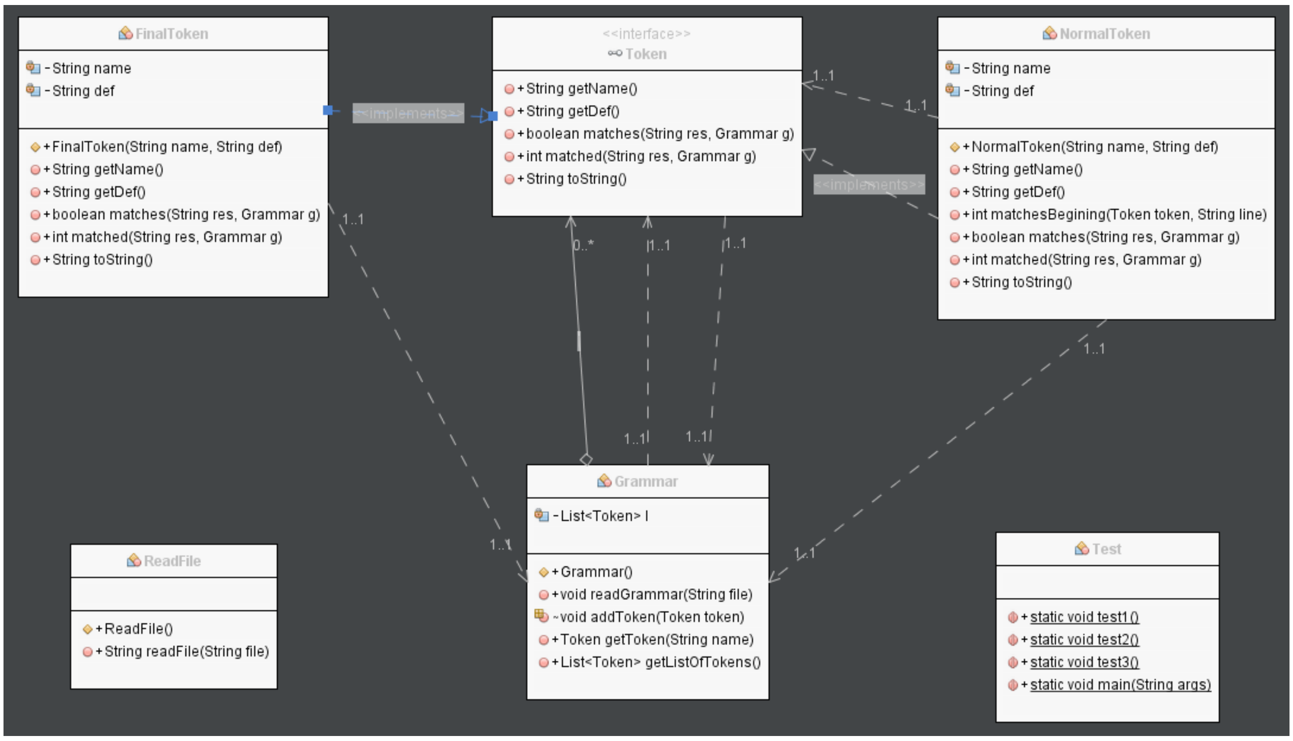
NetBeans is a software development platform written in Java. NetBeans was created in 2000 by Oracle. The NetBeans Platform allows applications to be developed from a set of modular software components called modules. Applications based on the NetBeans Platform, including the NetBeans integrated development environment (IDE), can be extended by third party developers.

The NetBeans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++ and HTML5.

2. **Project**

To describe my project, I will show what I have done first. After that, I will show the conception of my project. To end, I will conclude the project and tell what are the possible evolutions.

2.1. Goal



FinalToken
- String name
- String def
+ FinalToken(String name, String def)
+ String getName()
+ String getDef()
+ boolean matches(String res, Grammar g)
+ int matched(String res, Grammar g)
+ String toString()
<<implements>>
<<interface>>
Token
+ String getName()
+ String getDef()
+ boolean matches(String res, Grammar g)
+ int matched(String res, Grammar g)
+ String toString()
NormalToken
- String name
- String def
+ NormalToken(String name, String def)
+ String getName()
+ String getDef()
+ int matchesBegining(Token token, String line)
+ boolean matches(String res, Grammar g)
+ int matched(String res, Grammar g)
+ String toString()
<<implements>>
1..1
1..1
0..*
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
Grammar
- List<Token> l
+ Grammar()
+ void readGrammar(String file)
~ void addToken(Token token)
+ Token getToken(String name)
+ List<Token> getListOfTokens()
ReadFile
+ ReadFile()
+ String readFile(String file)
Test
+ static void test1 ()
+ static void test2()
+ static void test3()
+ static void main(String args)

Figure 5: UML of my project

This is an UML (Unified Modeling Language) of my project. It describe what I have done.

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.[4]

To conclude on this, the lexer is working. We will now see how I organize my project. To do that, I will describe the different parts of my project.

2.2. Versions, evolutions

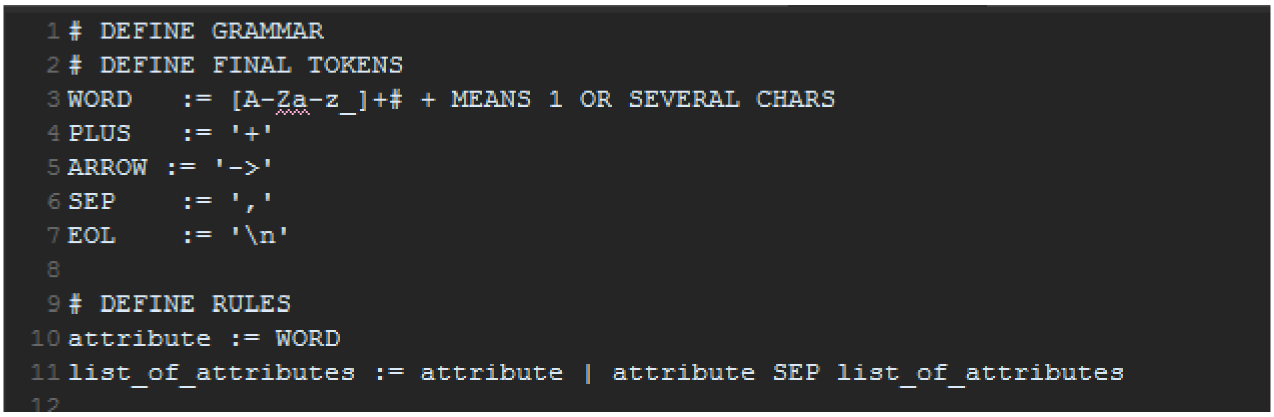
To describe my project, I decide to cut this in three parts. The three parts showed all the decisions I have made in my project.

a) First Part of the Lexer

To start the project, we need to create the lexer for that I decided at the beginning to have three first classes and two text files. These one :

Classes	
Name	Description
ReadFile	Read the Text Files
Token	Translate the Grammar into Tokens
Test	Main class
Text Files	
Name	Description
Grammar	Contains the Grammar
File	File to compare with the Grammar

ReadFile contains two methods readGrammar and readFile who used the class BufferedReader who reads a file. Token has a constructor with String,String because a token has a name and a definition. For exemple, this is a Grammar more particulary this is a part of thet Grammar that i used in this project.



```
# DEFINE GRAMMAR
# DEFINE FINAL TOKENS
WORD   := [A-Za-z_]+# + MEANS 1 OR SEVERAL CHARS
PLUS   := '+'
ARROW := '->'
SEP    := ','
EOL    := '\n'

# DEFINE RULES
attribute := WORD
list_of_attributes := attribute | attribute SEP list_of_attributes
```

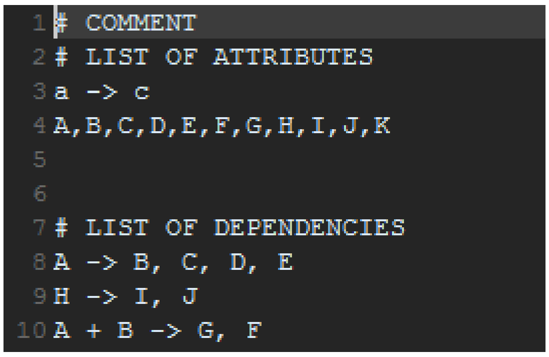
Figure 6: Extract of the text file Grammar

For understand the token “Word”, we used the class Pattern who is a compiled representation of a regular expression. A regular expression, specified as a string, must first be compiled into an instance of this class. The resulting pattern can then be used to create a Matcher object that can match arbitrary character sequences against the regular expression. All of the state involved in performing a match resides in the matcher, so many matchers can share the same pattern.

Moreover, we can see in the grammar, that we have two types of tokens :

- Final Token : who correspond as tokens that you know by what is it compose. You know that for exemple, the token named “PLUS” is a token who correspond at the symbol “+”.
- “Normal” (That's how I called them) Token : who correspond as tokens that you don't know what it correspond without another token. For exemple, the token named “attribute” is a token who correspond to “WORD”. But “WORD” is another token so we need the token “WORD” to understand “attribute”.

This is for exemple a part of a File who correspond with that Grammar.



```
# COMMENT
# LIST OF ATTRIBUTES
a -> c
A,B,C,D,E,F,G,H,I,J,K


# LIST OF DEPENDENCIES
A -> B, C, D, E
H -> I, J
A + B -> G, F
```

Figure 7: Exemple of a File

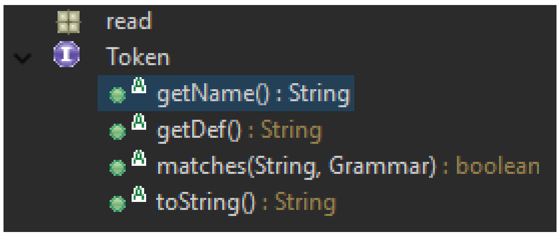
b) Second Part of the Lexer

I had some problems with the seperation of the token. Indeed, the grammar could understand the final tokens but it couldn't understand normal tokens.

So after I had some problems with the disposition of my classes, I decided to separate the class Token in three parts :

- Token, an interface who regroups all the methos used in Tokens
- FinalToken, a class who creates final tokens
- NormalToken, a class who creates “normal” tokens

The interface Token now contains all the method of the tokens :



```
read
Token
    getName() : String
    getDef() : String
    matches(String, Grammar) : boolean
    toString() : String
```

Figure 8: Composition of the interface Token

Later, I will separate the class ReadFile in two classes. Indeed, I will reate a class Grammar who allows you to read the text file Grammar and to translate this file in tokens. These tokens are putted into an List.

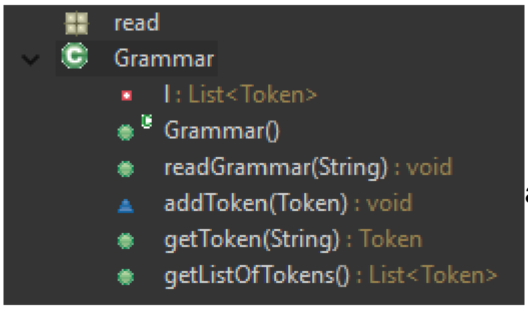

```
read
Grammar
    l : List<Token>
    Grammar()
    readGrammar(String) : void
    addToken(Token) : void
    getToken(String) : Token
    getListOfTokens() : List<Token>
```

Figure 9: Composition of the class Grammar

addToken give you a Token by giving his name.

c) <u>Last Part of the Lexer</u>

Classes	
Name	Description
FinalToken	Create a token that you know by what is it compose.
Grammar	Read the Grammar and put tokens in a List.
NormalToken	Create a token that you don't know what it correspond without another token.
ReadFile	Read the File who needs to be convert into objects
Test	Main class
Interfaces	
Name	Description
Token	Define what a token contains and what he can do.
Text Files	
Name	Description
Grammar	Contains the Grammar
File	File to compare with the Grammar

At the end for the Normal Token, I compare the number of part of a line separated with a space to know if the token matches with this line.

I met a problem with that so now the method matches of the Normal Token returns a integer instead of a boolean (true or false) and a new method is created he method matched to see if the number of the part of the token matches with the number of part in the line of the File.

2.3. **Conclusion of the project**

To do the conclusion of my project, I will cut it in two parts. The first will describe the conclusion of my project and the last part will talk about the possible evolutions of my project.

a) Conclusion of the project

To conclude with this project, I had very much problems with the lexer and the parser. The cause is that they aren't exemple of this in the documentations. It's only seemed to describe what is the lexer and the parser.

Moreover, I didn't exercise myself with that (lexer and parser) during my studies. It's all new for me. It's why I seemed lost at the beginning of the project. I lose a lost of time to understand what is a parser and a lexer exactly and after, how to create one.

b) Next evolutions

For the next evolutions of my project, we can tell that we can include in my project the parser.

A HCI in Java Swing can be also created to looks like a IDE.

Human–computer interaction (HCI) researches the design and use of computer technology, focusing on the interfaces between people (users) and computers. Researchers in the field of HCI both observe the ways in which humans interact with computers and design technologies that let humans interact with computers in novel ways. [5]

We can also allow the users to use his own grammar with his own file.

Conclusion

I'm glad of this internship because it have teached me a lot of things. I have figured out that I prefer to work in a team. It doesn't mean that I don't like to work alone but work in a team motivates me more. I do my best to go as far as possible but I'm a little bit disappointing that I couldn't finish the project.

During this study, I encountered several difficulties. The first difficulty is that I really had a problem with tokens, I always met a problem with token who couldn't match some line of my File.

The second difficulty is that the tutor let us organize myself to do my project. This help to understand my limits, since I had to organize time and effort that I had to spend in that project. For that, I started to put some schedules to give me some organization.

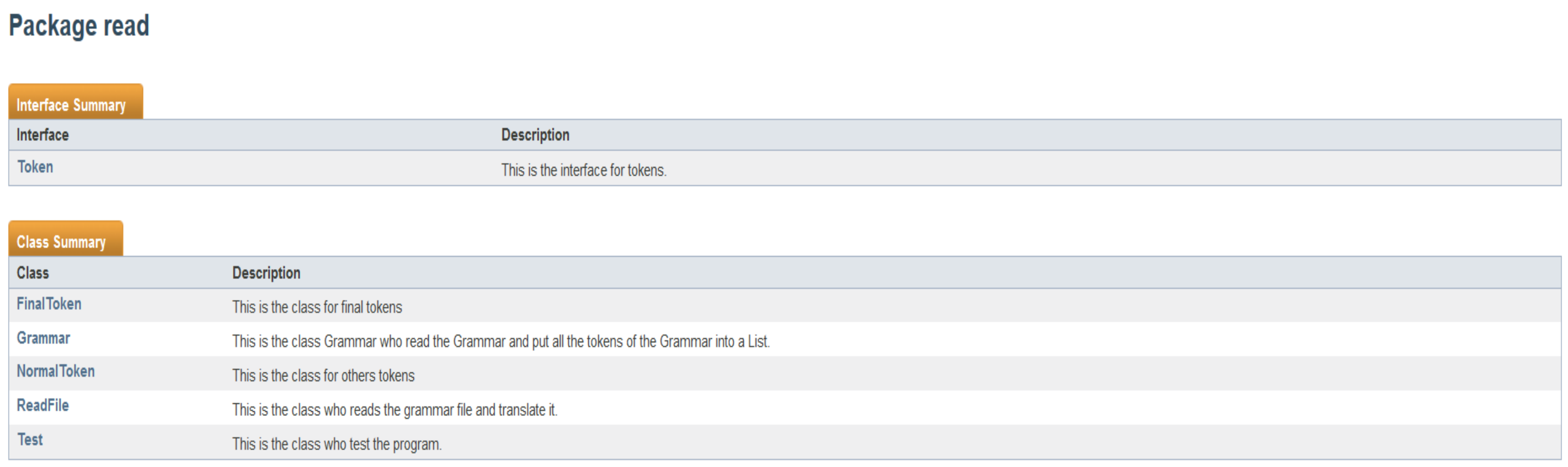
This internship also strengthened me in my idea that I loved Computer Sciences.

References

Wikipedia	https://en.wikipedia.org/wiki/Lexical_analysis	1
	https://en.wikipedia.org/wiki/Parsing#Computer_languages	2
	https://en.wikipedia.org/wiki/Integrated_development_environment	3
	https://en.wikipedia.org/wiki/Unified_Modeling_Language	4
	https://en.wikipedia.org/wiki/Human–computer_interaction	5
	https://en.wikipedia.org/wiki/Java_(programming_language)	
	https://en.wikipedia.org/wiki/Eclipse_(software)	
	https://en.wikipedia.org/wiki/NetBeans	
Java	https://www.java.com/en/	
Eclipse	https://eclipse.org/	
NetBeans	https://netbeans.org/	

Attachments

Javadoc of my project



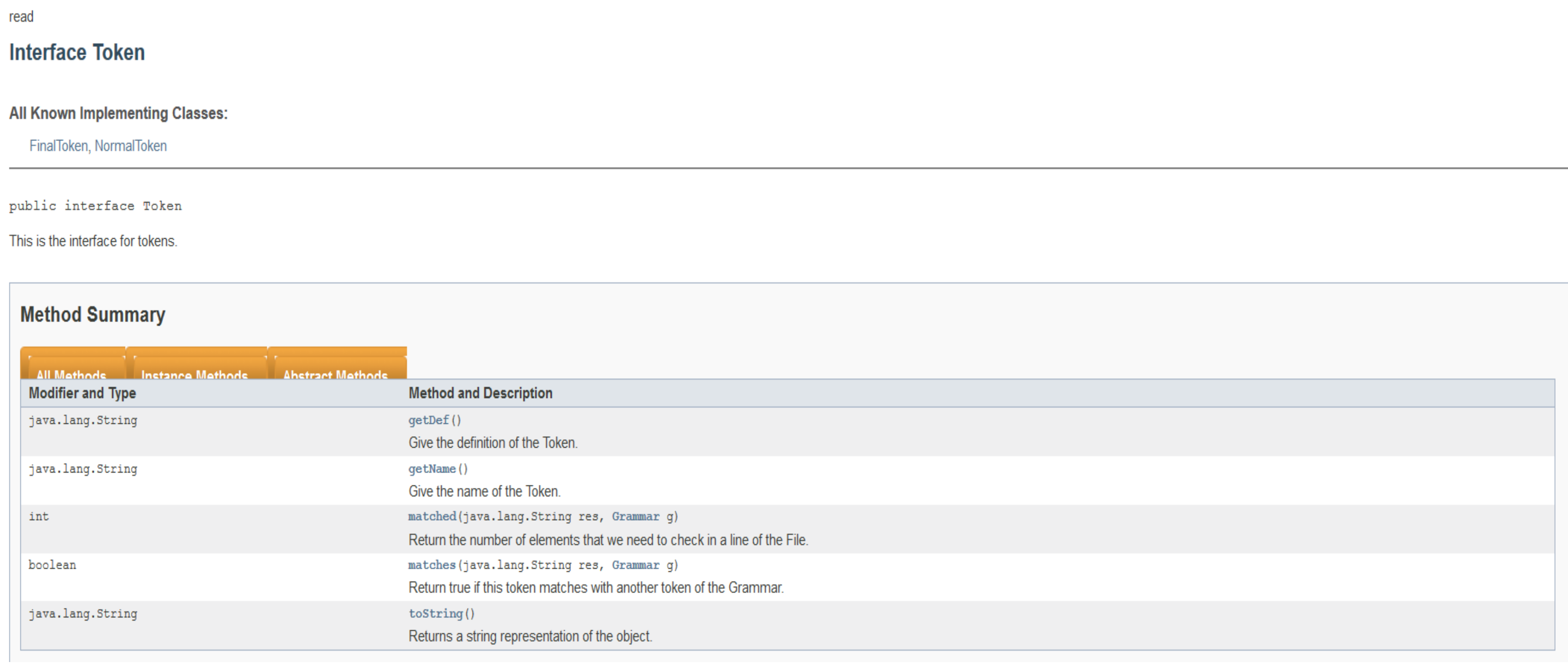
Package read

Interface Summary

Interface	Description
Token	This is the interface for tokens.

Class Summary

Class	Description
FinalToken	This is the class for final tokens
Grammar	This is the class Grammar who read the Grammar and put all the tokens of the Grammar into a List.
NormalToken	This is the class for others tokens
ReadFile	This is the class who reads the grammar file and translate it.
Test	This is the class who test the program.



read

Interface Token

All Known Implementing Classes:

FinalToken, NormalToken

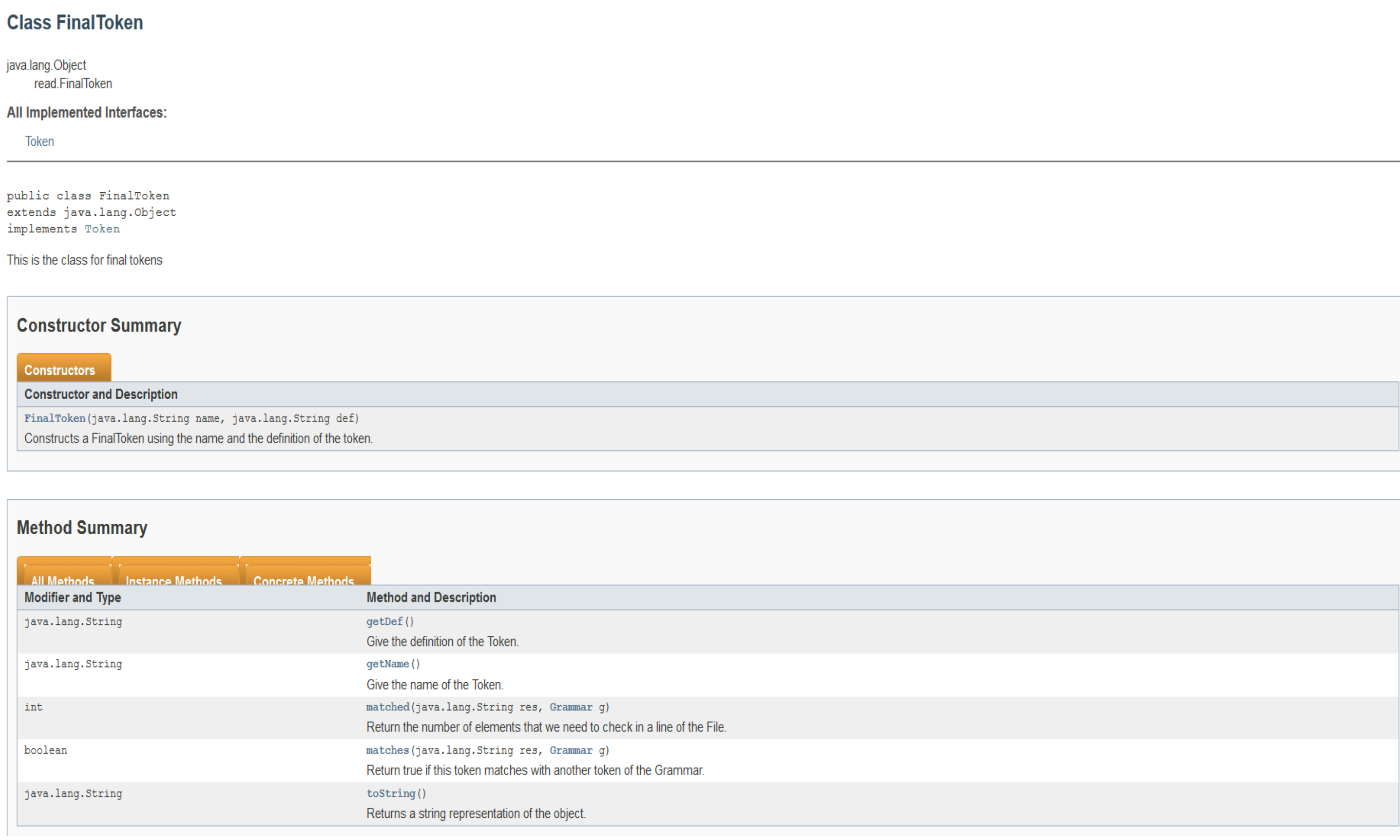
```
public interface Token
```

This is the interface for tokens.

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
`java.lang.String`	`getDef()` Give the definition of the Token.
`java.lang.String`	`getName()` Give the name of the Token.
`int`	`matched(java.lang.String res, Grammar g)` Return the number of elements that we need to check in a line of the File.
`boolean`	`matches(java.lang.String res, Grammar g)` Return true if this token matches with another token of the Grammar.
`java.lang.String`	`toString()` Returns a string representation of the object.



Class FinalToken

java.lang.Object
  read.FinalToken

All Implemented Interfaces:

Token

```
public class FinalToken
extends java.lang.Object
implements Token
```

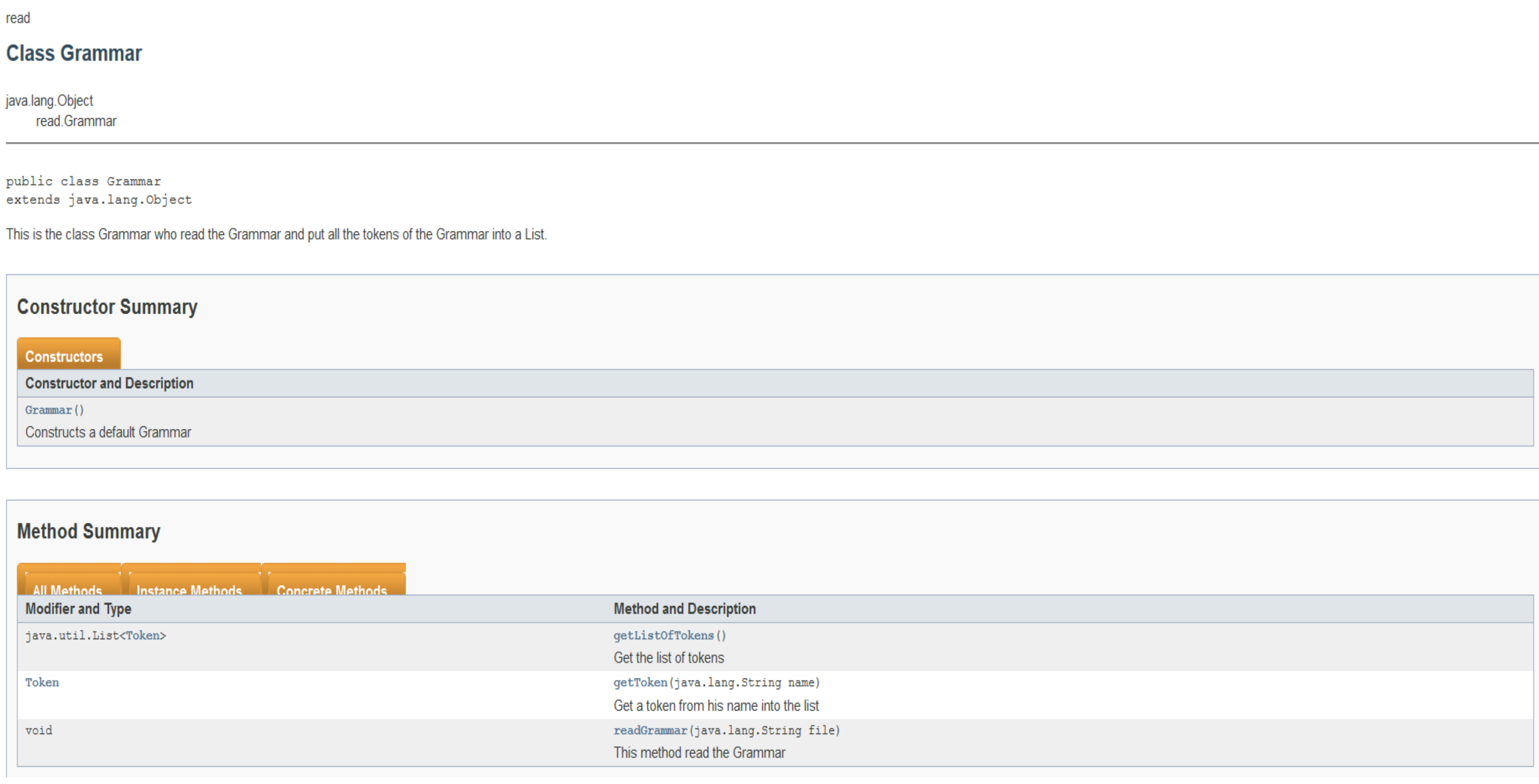
This is the class for final tokens

Constructor Summary

Constructor and Description
`FinalToken(java.lang.String name, java.lang.String def)` Constructs a FinalToken using the name and the definition of the token.

Method Summary

Modifier and Type	Method and Description
`java.lang.String`	`getDef()` Give the definition of the Token.
`java.lang.String`	`getName()` Give the name of the Token.
`int`	`matched(java.lang.String res, Grammar g)` Return the number of elements that we need to check in a line of the File.
`boolean`	`matches(java.lang.String res, Grammar g)` Return true if this token matches with another token of the Grammar.
`java.lang.String`	`toString()` Returns a string representation of the object.



read

Class Grammar

java.lang.Object
  read.Grammar

```
public class Grammar
extends java.lang.Object
```

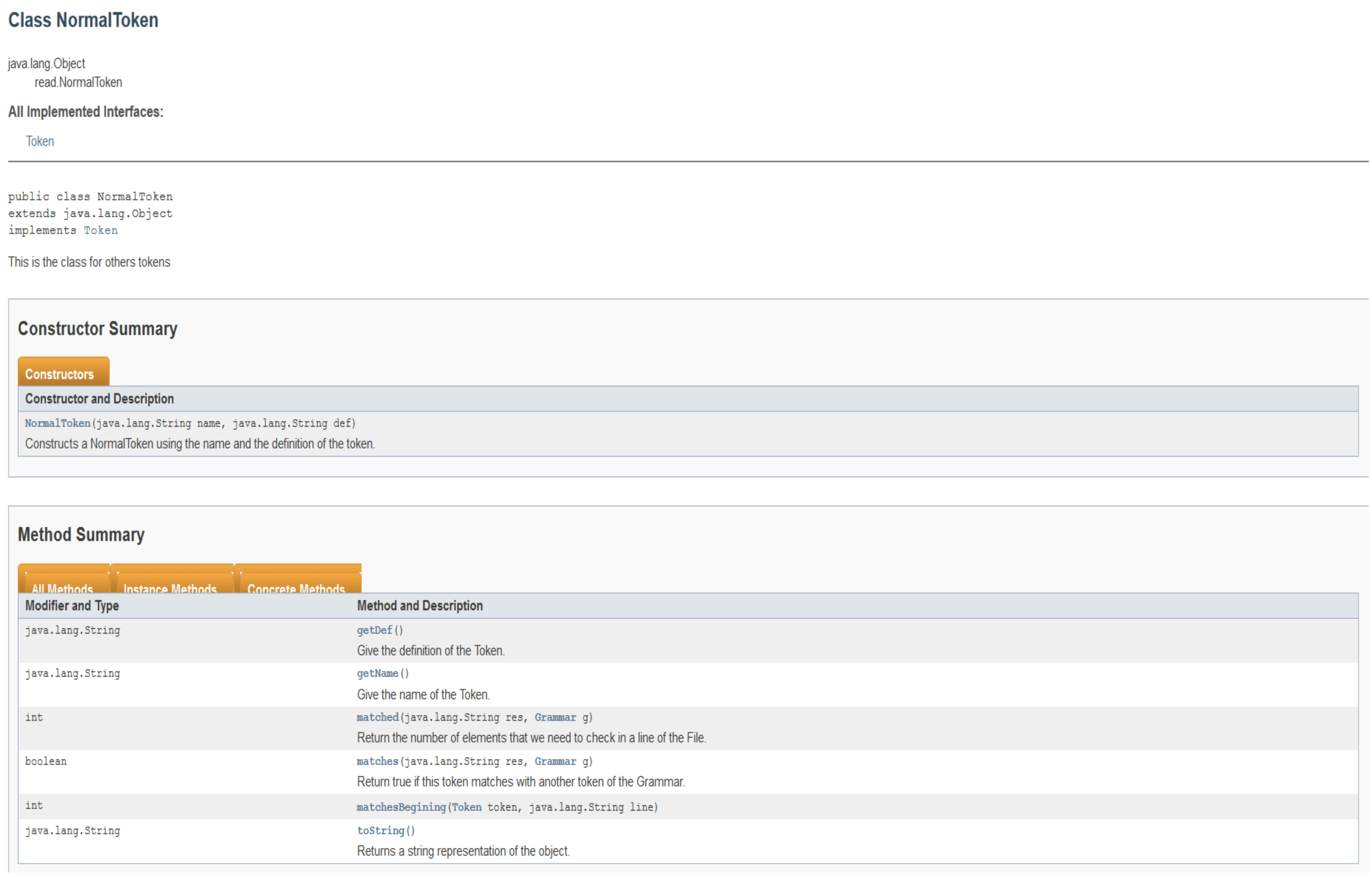
This is the class Grammar who read the Grammar and put all the tokens of the Grammar into a List.

Constructor Summary

Constructor and Description
`Grammar()` Constructs a default Grammar

Method Summary

Modifier and Type	Method and Description
`java.util.List<Token>`	`getListOfTokens()` Get the list of tokens
`Token`	`getToken(java.lang.String name)` Get a token from his name into the list
`void`	`readGrammar(java.lang.String file)` This method read the Grammar



Class NormalToken

java.lang.Object
    read.NormalToken

All Implemented Interfaces:

Token

```
public class NormalToken
extends java.lang.Object
implements Token
```

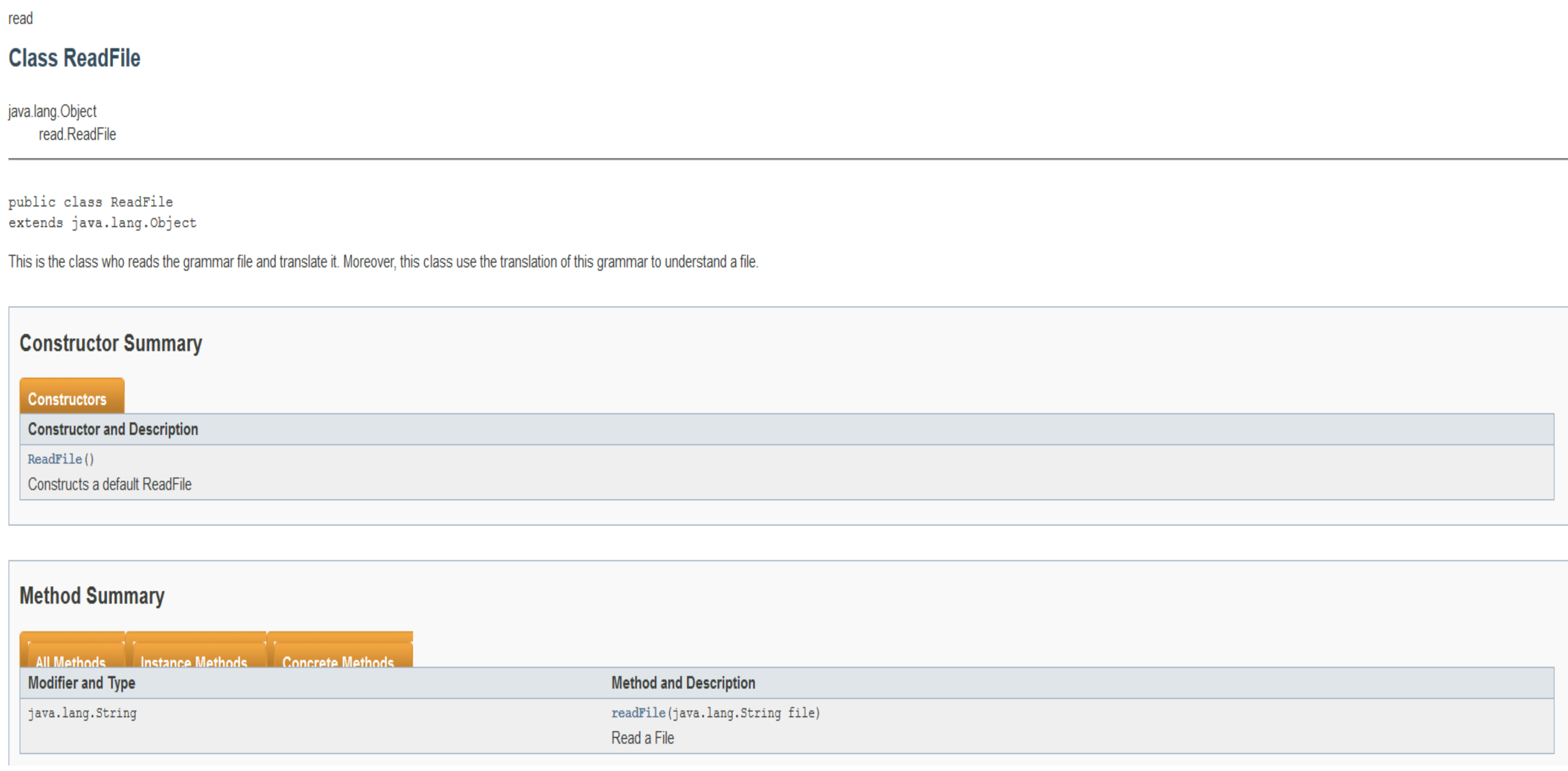
This is the class for others tokens

Constructor Summary

Constructor and Description
`NormalToken(java.lang.String name, java.lang.String def)` Constructs a NormalToken using the name and the definition of the token.

Method Summary

Modifier and Type	Method and Description
`java.lang.String`	`getDef()` Give the definition of the Token.
`java.lang.String`	`getName()` Give the name of the Token.
`int`	`matched(java.lang.String res, Grammar g)` Return the number of elements that we need to check in a line of the File.
`boolean`	`matches(java.lang.String res, Grammar g)` Return true if this token matches with another token of the Grammar.
`int`	`matchesBegining(Token token, java.lang.String line)`
`java.lang.String`	`toString()` Returns a string representation of the object.



read

Class ReadFile

java.lang.Object
    read.ReadFile

```
public class ReadFile
extends java.lang.Object
```

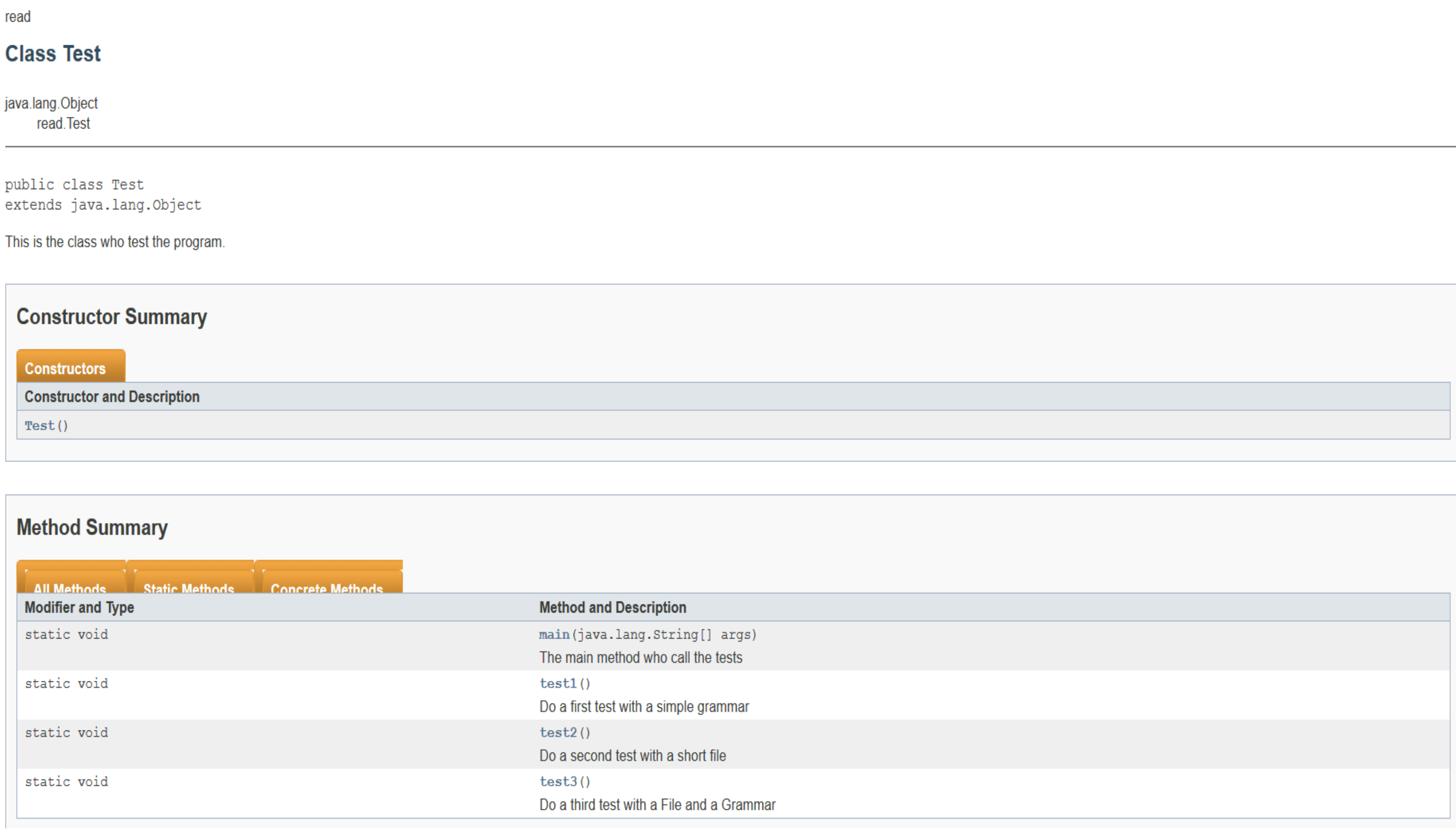
This is the class who reads the grammar file and translate it. Moreover, this class use the translation of this grammar to understand a file.

Constructor Summary

Constructor and Description
`ReadFile()` Constructs a default ReadFile

Method Summary

Modifier and Type	Method and Description
`java.lang.String`	`readFile(java.lang.String file)` Read a File



read

Class Test

java.lang.Object
    read.Test

```
public class Test
extends java.lang.Object
```

This is the class who test the program.

Constructor Summary

Constructors

Constructor and Description
`Test()`

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
`static void`	`main(java.lang.String[] args)` The main method who call the tests
`static void`	`test1()` Do a first test with a simple grammar
`static void`	`test2()` Do a second test with a short file
`static void`	`test3()` Do a third test with a File and a Grammar