Laith Hussin
921659619
Mar 25-2022

Assignment 3 Documentation

Github Repository

https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-3---parser-laith57th.git

Project Introduction and Overview

Using a code skeleton provided for the parser class and other corresponding classes, I was required to improve the output of the AST tree representation to make it more uniform, add the new tokens implemented in the lexer class to be understood by the parser, and create two new classes, offsetVisitor and DrawOffsetVisitor which assign offset values and uses them to plot and display an improved tree representation.

Scope of Work

Task		**Completed**
Accommodate new tokens in the parser		**X**
	Add utf16String and timestamp types and literals to be understood by the parser class.	**X**
	Add switch, case, and default tokens to the TokenSetup class and rerun to generate new tokens.	**X**
	Add new the additional relational operators: > and >=	**X**
Allow the parser to accept if without an else statement		**X**
	Add a condition inside the rStatement method that allows an if statement without an else block.	**X**
Implement the switch statement		**X**
	Inside the parser class, add a condition in the rStatement method that searches for a switch keyword.	**X**
	If a switch token is found, create a new switch tree.	**X**
	Add an identifier as a child node to the switch parent node.	**X**
	Add a case statement node to the tree followed by the list of cases between the brackets.	**X**
	Expect a hashDelimeter after each case statement and add the expression that follows to the tree.	**X**
	If a default statement is present, create a new defaultTree and add it to the parent switch block node.	**X**
Remove debug statements		**X**
	Removed lexer output	**X**
	Added the lines of code from the target file before the printVisitor output.	**X**
Implement offsetVisitor		**X**

	Create a HashMap containing the offset values for each node in the tree keyed by AST nodes at each offset level.	**X**
	Use the provided algorithm to perform the post order traversal, assign offset values to each node, and shift nodes as needed.	**X**
Implement DrawOffsetVisitor		**X**
	Rename drawVisitor to DrawOffsetVisitor	**X**
	Adjust the width and the height of the canvas to allow the full tree representation to be displayed properly on the screen.	**X**
	Draw the oval and the connecting lines and set appropriate colors.	**X**
Test switch statement, offsetVisitor, DrawOffsetVisitor, new token types, and if statement without else block		**X**

	program { int i utf16string str
	  i = i + j + 7
	  j = write(i)
	  j = 2022~02~15~12:15:22
	  c = \uD83D\uDC7D
	  if(mouse > 8) then {
	  return 0
	  }
	  switch (mouse) {
	  case [dogs, cats] # return 0
	  case [] # return 1
	  default # return 0
	  }
	}
	
	Result : successful compilation and correct tree displayed.
		X

Execution and Development Environment

To complete this project, I used Visual Studio Code on my Macbook M1 pro.

openjdk version "17.0.2" 2022-01-18

OpenJDK Runtime Environment Temurin-17.0.2+8 (build 17.0.2+8)

OpenJDK 64-Bit Server VM Temurin-17.0.2+8 (build 17.0.2+8, mixed mode)

Compilation Result

Using the terminal, I used the following commands on the project files:

```
> javac lexer/setup/TokenSetup.java
> java lexer.setup.TokenSetup
> javac compiler/Compiler.java
> java compiler.Compiler filename.x
```

The program ran as expected and no error messages were displayed

Assumptions

No assumptions made.

Summary of technical work

To complete this project I implemented a strategy design pattern to traverse through the AST containing the scanned tokens in multiple classes. Using object oriented programming, I called offsetVisitor and DrawOffsetVisitor along with the parser and lexer objects inside the compiler class to render the AST from the given token stream. I used single function methods inside the parser, offsetVisitor, and DrawOffsetVisitor classes to implement the switch statement grammar for the x language and render the AST.

Implementation

New Tokens

The easy part about this was adding the new keywords to the tokens file such as switch, case, and default, rerunning the TokenSetup and generating them. However, accommodating the new token types such as utf16String and timeStampType was a bit tricky as I had to go through the entire code and understand the algorithm in order to add these tokens to the implementation. As for the relational operators, it was easy to add them to the relationalOps enumset once I began to understand the algorithm.

If statement without an else block

Beginning this assignment, I anticipated that this part of the assignment would be complicated; however, it turned out to be the simplest change to the parser class which was refreshing.

Switch Statement

Implementing the switch statement into the parser was definitely one of the more difficult parts of this assignment. At times, things got really blurry, however, as I analyzed the code over and over everything fell into place and I was able to see the bigger picture. To make things easier, I used multiple methods that perform each part of the switch structure (i.e. switchBlock, caseStatement, CaseList, and default statement). This was really helpful when debugging as I was able to trace through the methods and fix issues as they appear.

OffsetVisitor

For this class, I mainly focused on understanding the algorithm provided in class to be able to implement the post order traversal and recursively assign the correct values to each node in the tree. I began by creating an integer array that holds the offset value of each node after the post order traversal and a hashMap that holds the offset values keyed by the tree nodes. Using a hashMap really helped me as I was able to manipulate the value of each node by using the get method whenever needed in my implementation. Following the provided algorithm, I was able to successfully reorganize and improve the displayed tree when running the compiler class.

DrawOffsetVisitor

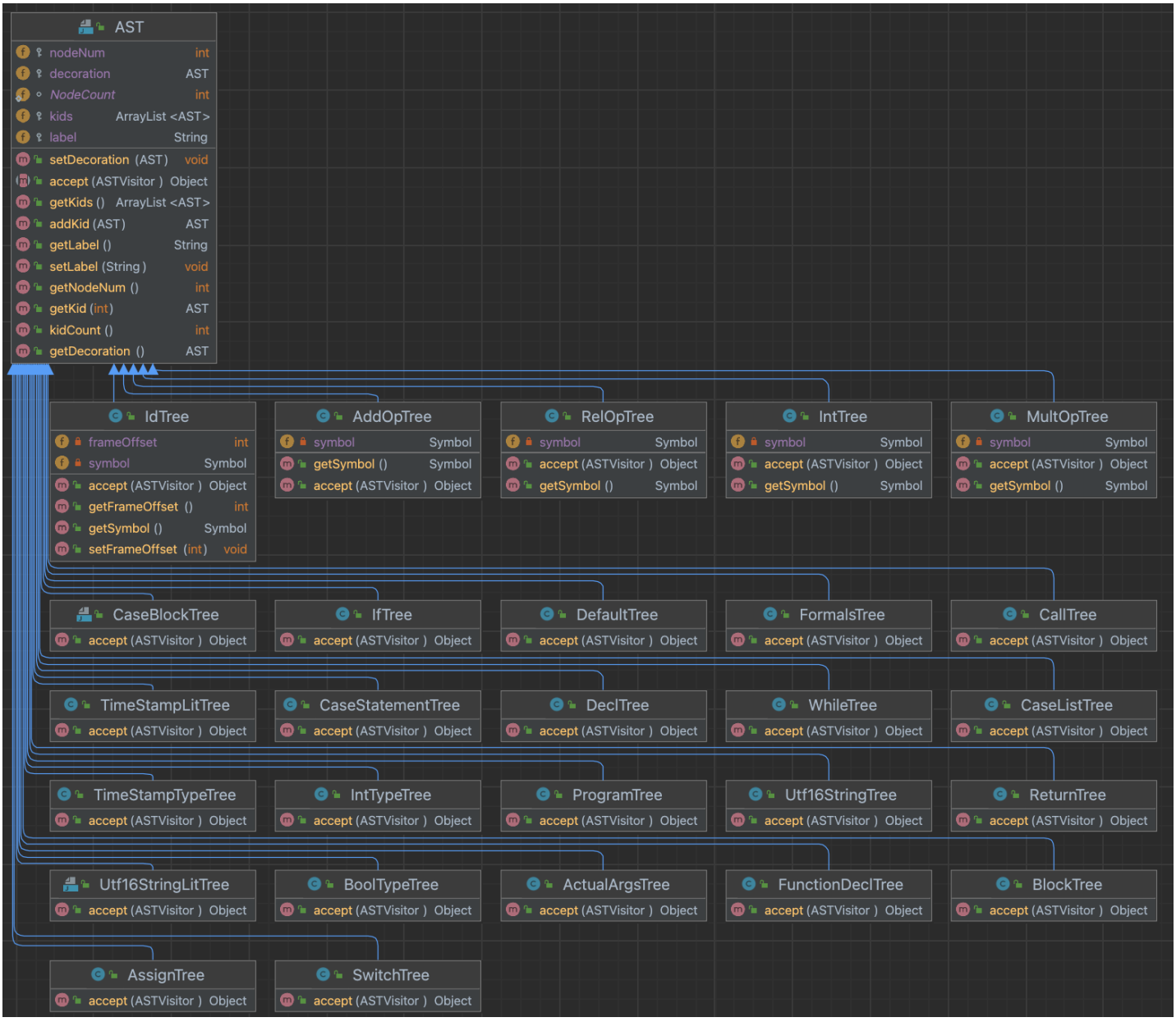
Implementing DrawOffsetVisitor was one of the easier tasks in this project. I started by adjusting the width and the height of the display canvas. Next, I used java's built-in graphics methods to draw the ovals and lines connecting the decorated AST. I then used the offset values from the OffsetVisitor class as reference for the position of each node in the tree using the rendering algorithm provided in class.

Code Organization

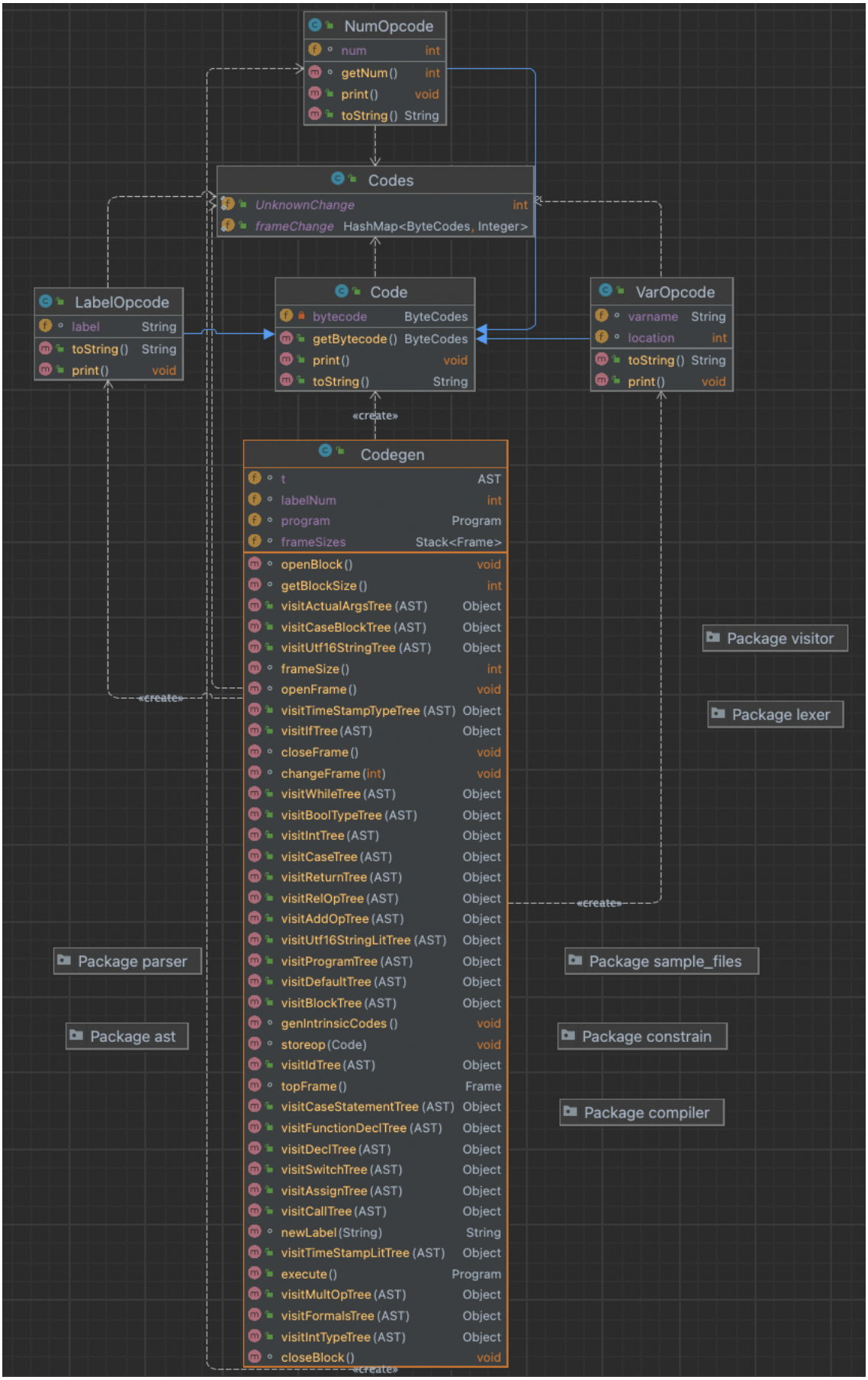
For the switch implementation, I created multiple methods to retrieve and error check the different parts of the switch structure: Switch block, case statement, case list, and optional default statement.

Class Diagram

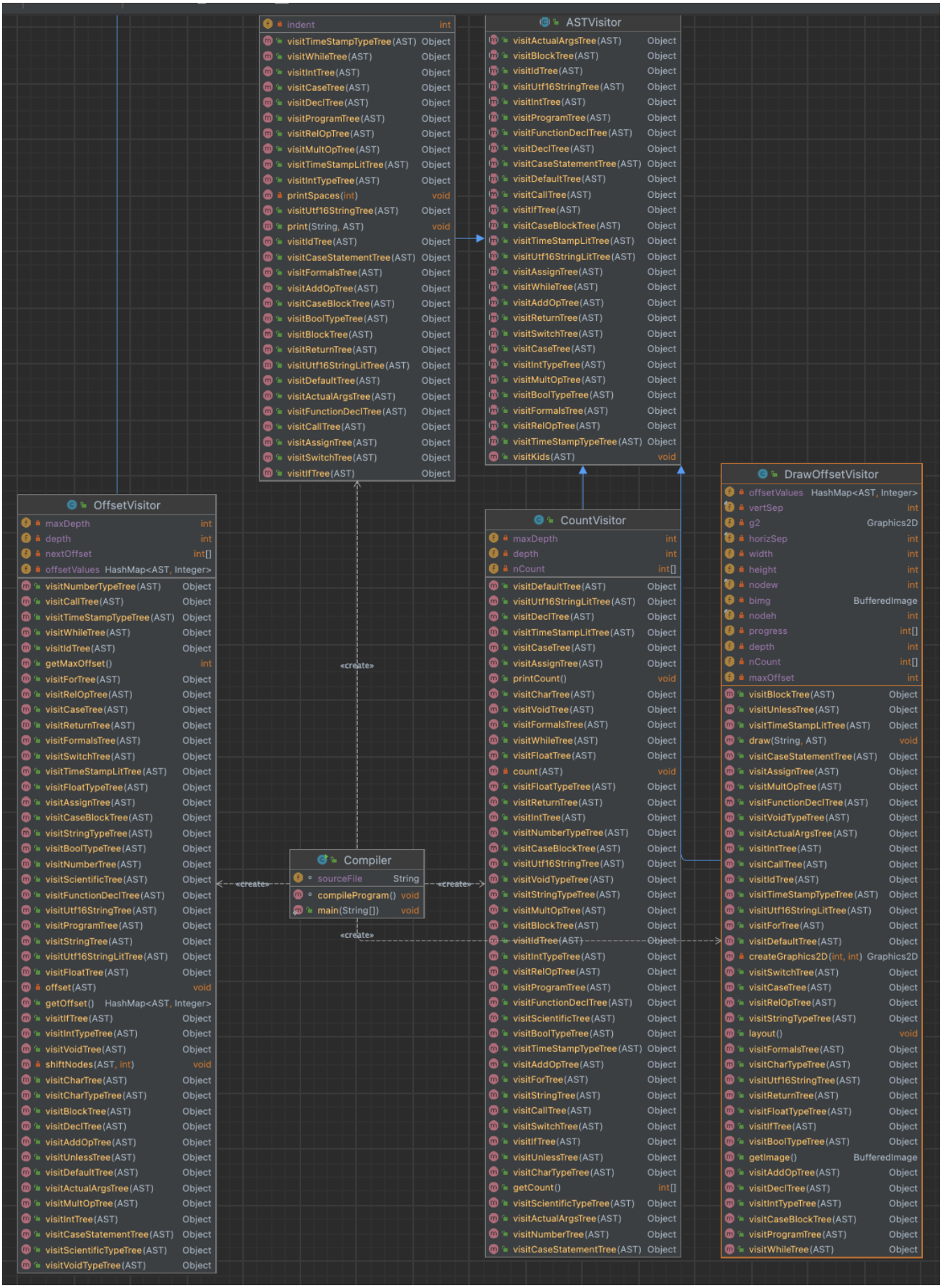
AST package



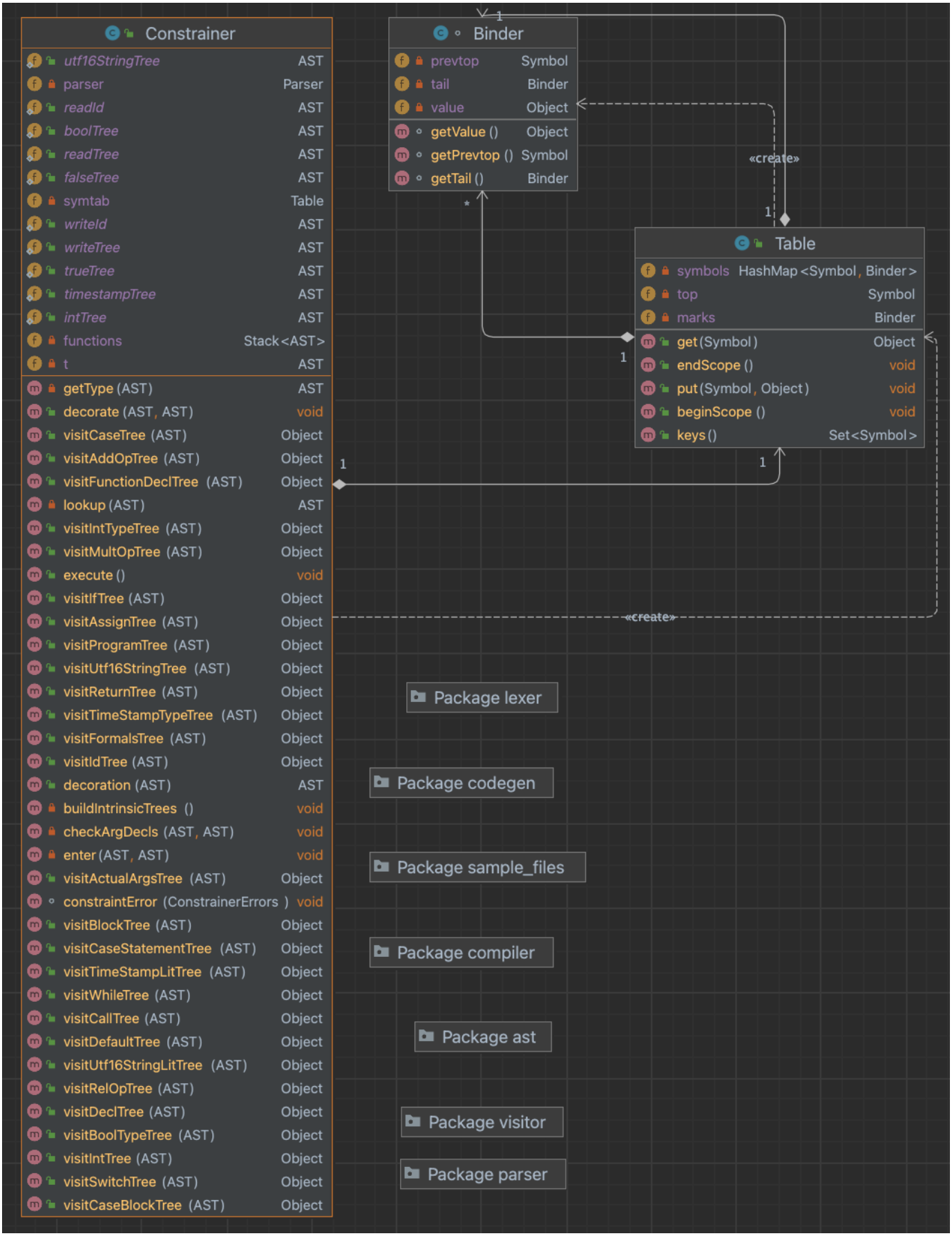
Codegen package



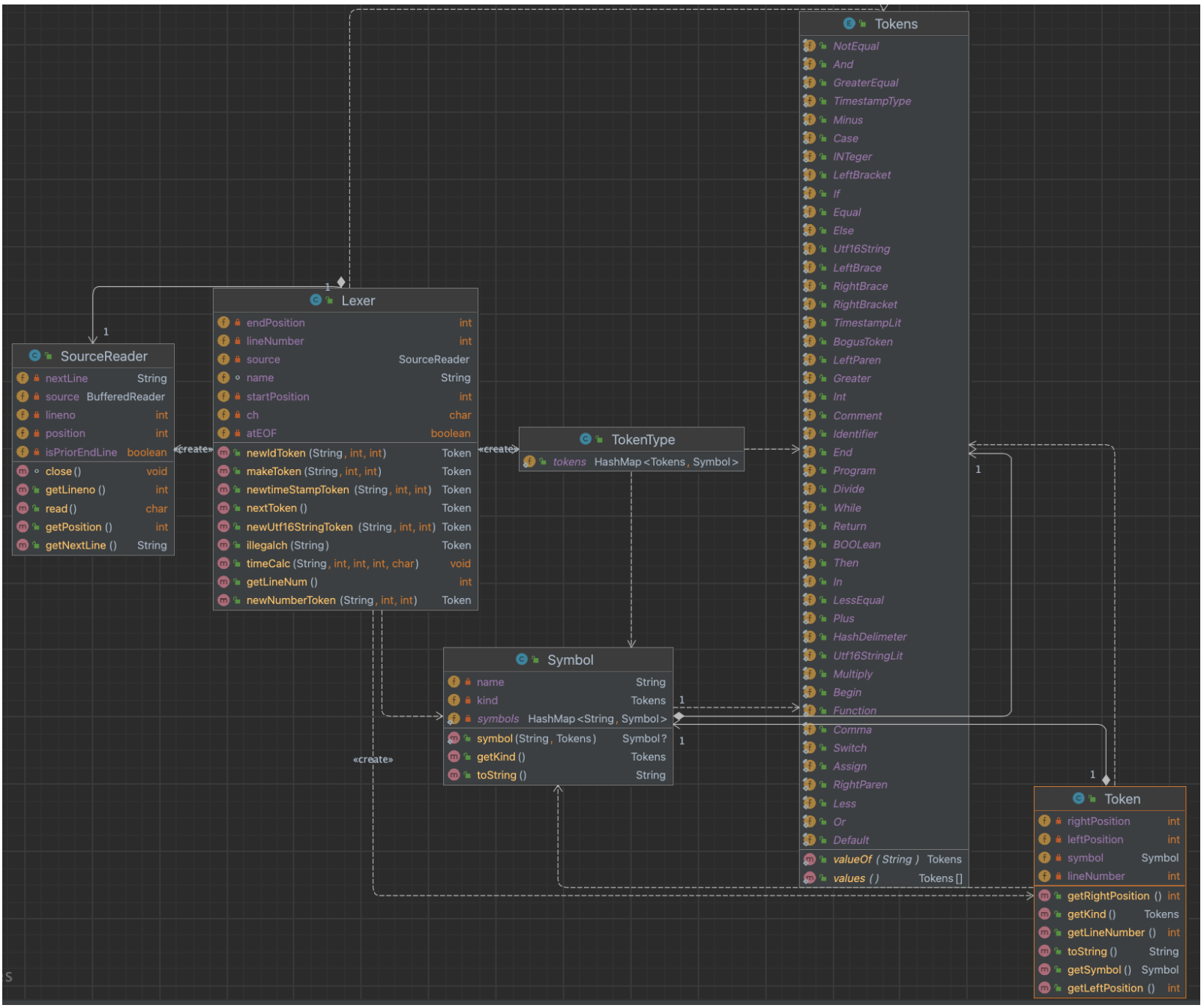
Compiler package



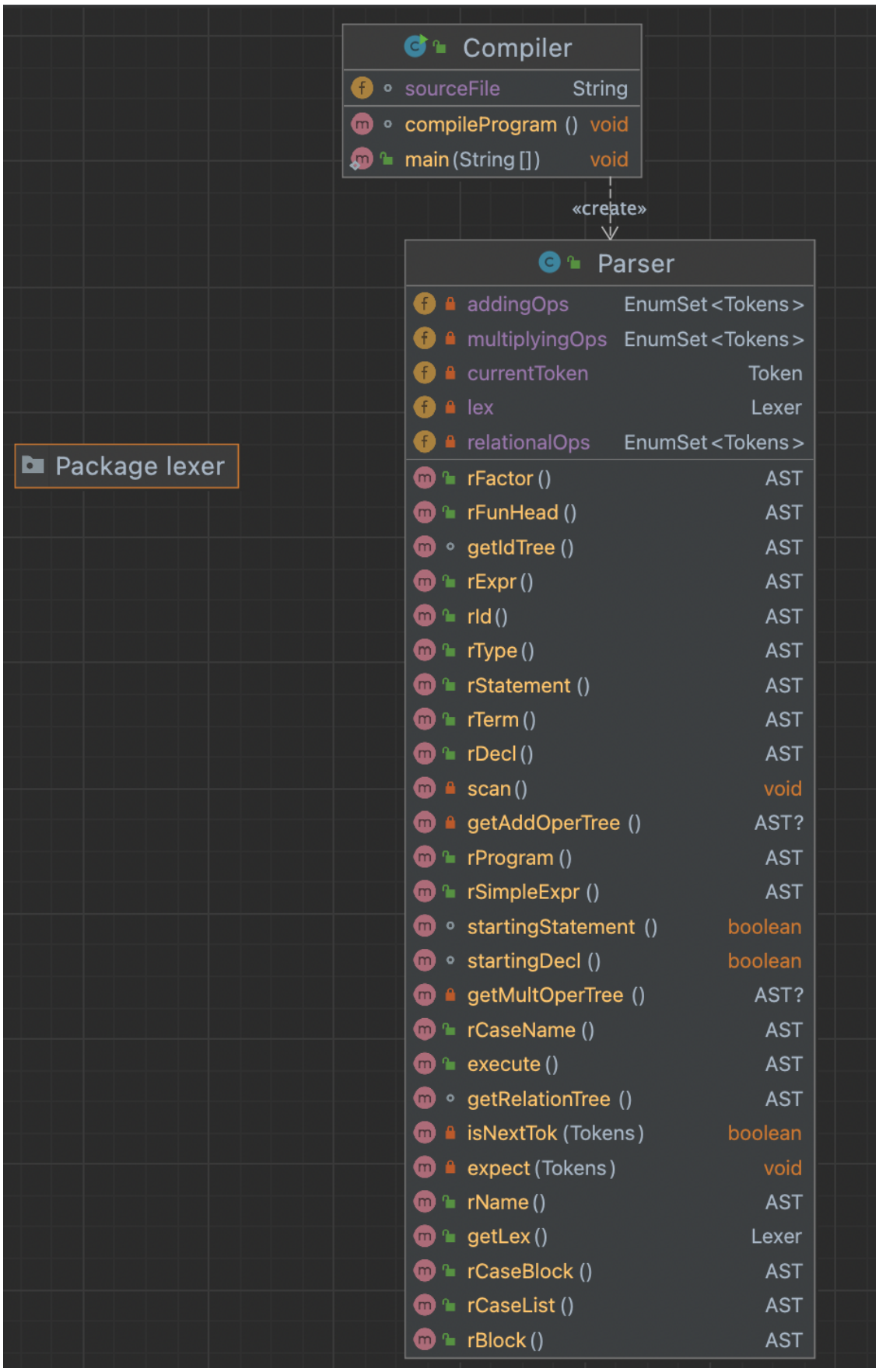
Constrain package



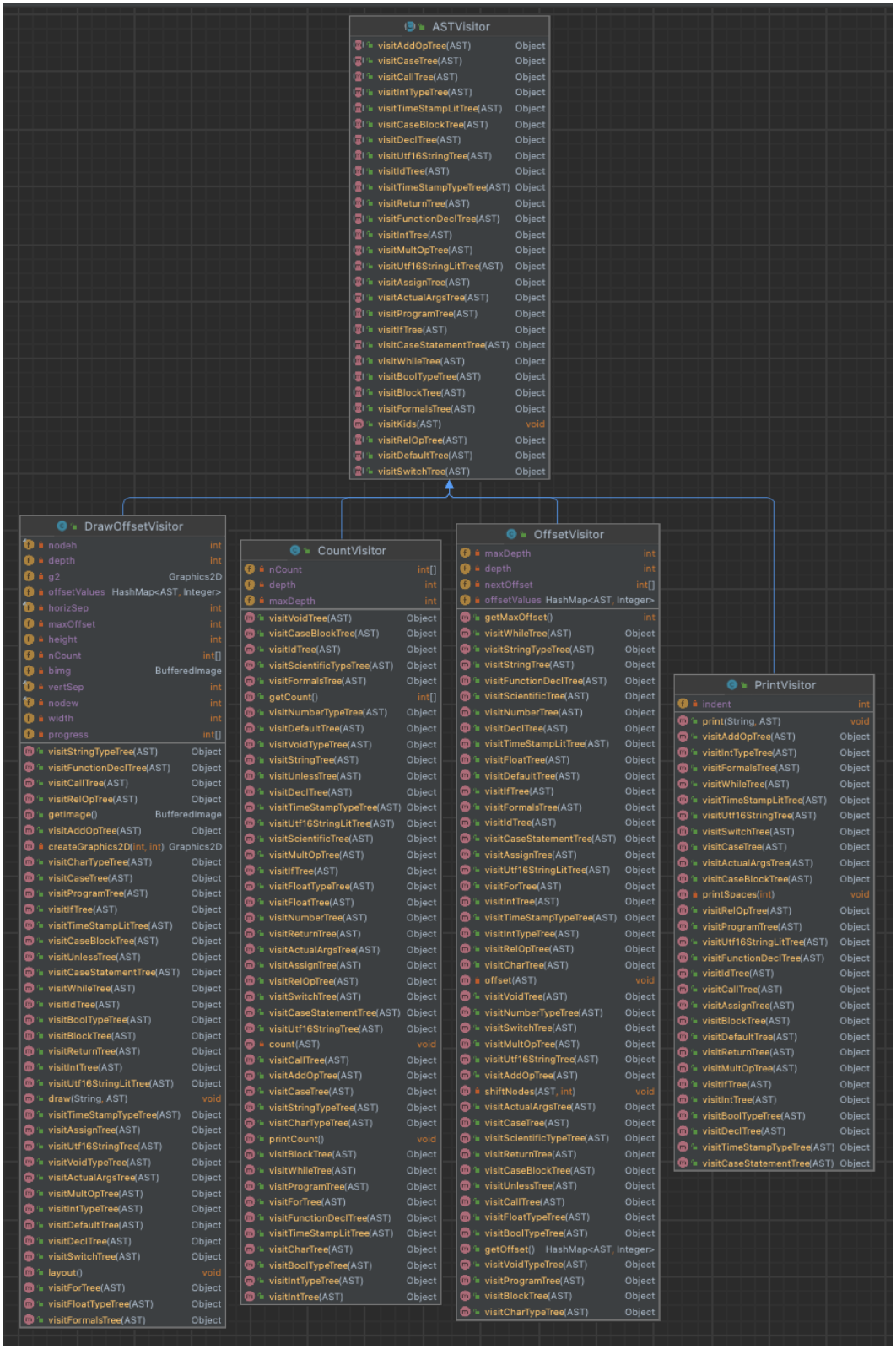
Lexer Package



Parser package



visitor package



Results and Conclusion

This project was a good refresher on data structures, binary search trees, recursion, and other programming concepts. While it was difficult to perform the required tasks for this assignment, it was rewarding to finally be able to understand how the parser algorithm works with AST classes and the last lexer project. I successfully completed all the required tasks for the parser and other corresponding classes.

Challenges

Similar to the last two assignments, beginning this assignment was very difficult, however, I wasn't super worried since I've felt the same way before with the lexer project and it turned out okay for me. One challenge I faced was tracing the code and understanding the flow of the program since there were many things happening at once and a multitude of semi-complex methods in the parser class, especially. Another challenge I can highlight was rendering the switch statement onto the tree output.

Future Work

With the parser algorithm, we can further improve the x language by adding more grammar such as a float token, an array structure, or a for loop.