Kshitiz Sareen
922230410
December 13, 2021

Assignment 5 Documentation

Github Repository

Link to Github Repository.

Project Introduction and Overview

The motive of this project was to build a debugger component for the x language compiler to be able to debug our programs.

Summary of Technical Work

Added additional bytecodes for processing.

Added a Function Environment Record class to keep track of function details like name, starting line, and ending line.

Updated the bytecode classes for the debugger.

Created a debugger code table to retrieve the bytecodes used by the debugger.

Added Command classes to process each command given by the user.

Added UI classes to give the appropriate output for each command.

Execution and Development Environment

I used the Visual Studio Code editor on my Mac to complete this assignment, and all my testing was done using the terminal.

Scope of Work

Task		Completed
Implemented the algorithm that helps the interpreter in processing byte codes.		✓
Tested the algorithm with various files that cover all possible test cases. The files that were used were:		✓
	sample_files/fatorial.x	✓
	sample_files/factorial.x.cod	✓
Added new byte code classes and made changes to existing bytecode classes to support debugging behaviour. The new bytecodes that were added were the LINE, FUNCTION and FORMAL. The existing bytecodes that were changed were CALL, RETURN, LIT and POP. Replicated the remains bytecodes in the debugger bytecodes package.		✓
Implemented the FunctionEnvironmentRecord class that is used to keep track of the functions state in the debugger. It keeps track of the function name, the start and end lines of a function and the current line number.		✓
Created the debugger code table that holds the bytecode classes required by the debugger.		✓
Implemented the debugger class that holds the stack of function environment records, and a vector of entries where each entry corresponds to the line number, source line and breakpoint of the source program. These data structures will be manipulated by the debugger commands, and the debugger bytecodes will be executed from the vm to call the methods of the debugger class.		✓
Added Debugger command classes that are implemented by the debugger. The debugger command classes process the commands from the user. The different types of commands implemented are:		✓
	set:- The set command is used to set a breakpoint for a line in the program file.	✓
	list:- The list command is used to list the breakpoints set with the debugger.	✓
	locals:- The locals command prints all the symbols in the current frame.	✓
	source:- The source command displays the source code of the current function.	✓
	step:- The step command executes a bytecode.	✓
	continue:- The continue command causes execution to continue.	✓
	exit:- The exit command halts execution.	✓

Implemented the DebuggerShell class to prompt the user for a command prompt.	✓
Added UI classes that display the appropriate output for each command given by the user.	✓
Implemented the debugger virtual machine class that encapsulates the methods of the debugger, and executes the byte codes, in order to manage the debugging session.	✓

Compilation Result

Using these instructions, I executed the interpreter:

```
> javac interpreter/debugger/commands/*.java
> javac interpreter/bytecode/*.java
> javac interpreter/bytecode/debuggercodes/*.java
> javac interpreter/Interpreter.java
```

They were no warnings and the program was executed.

Using these instructions, I compiled ran the interpreter in debugger mode:

```
java interpreter.Interpreter -d <base_filename>
```

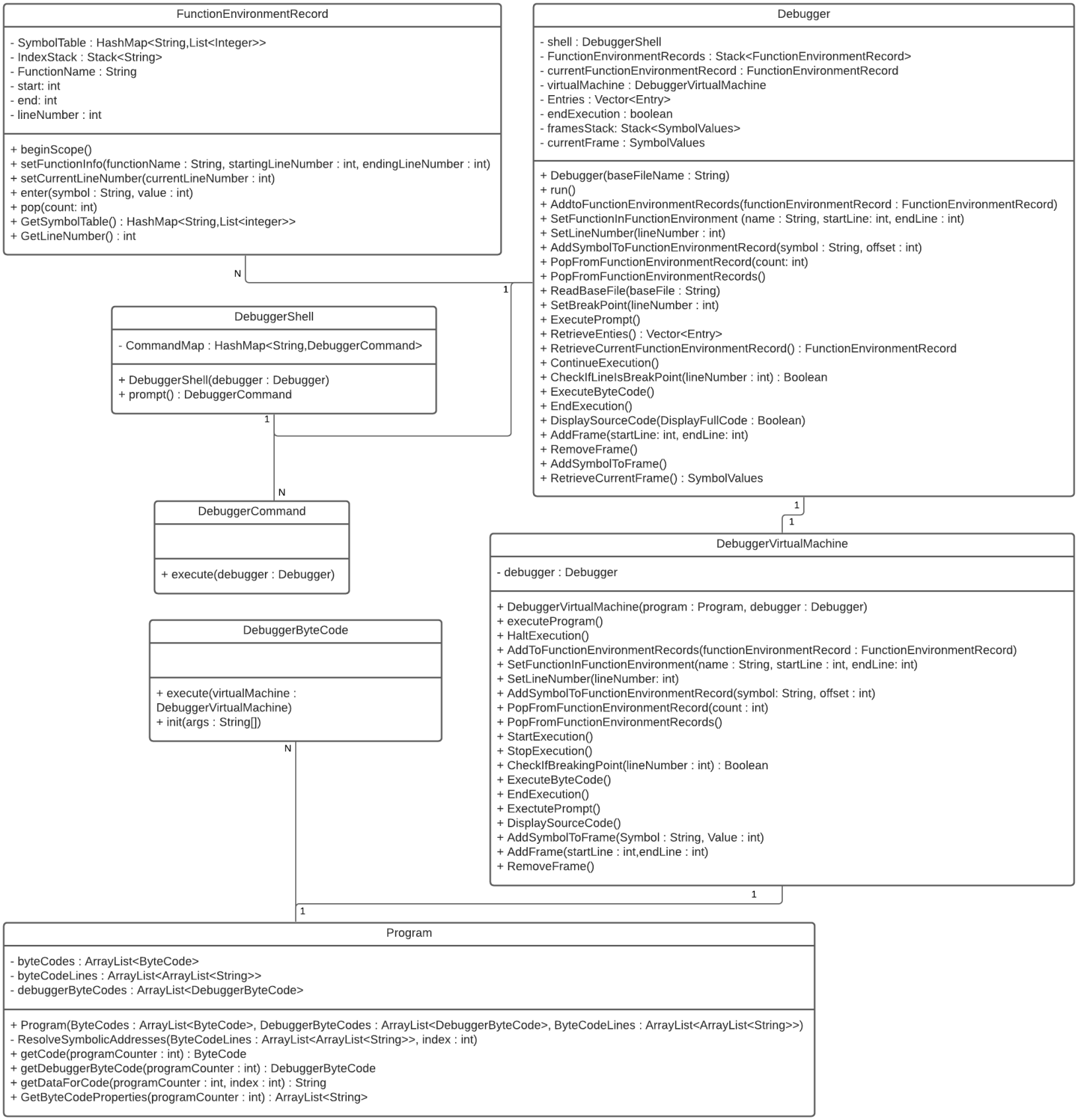
They were no warnings and the user prompt was executed

Assumptions

I assumed that all the base files and bytecode files were generated correctly, and they do not contain any errors.

Class Diagram

The following class diagram shows all the classes, along with their methods and properties. It shows the relation between the Debugger, DebuggerVirtualMachine, Program, FunctionEnvironmentRecord, DebuggerShell, Debugger Bytecode classes, and DebuggerCommand classes.



Implementation

Debugger

The Debugger class contains all the methods and properties that are used to interact with the Stack of Function Environment Records and the virtual machine. It receives a command from the Debugger shell and executes the command.

It contains methods that manipulate the vector of entries, and the current Function Record Environment, and the current function frame. These methods are called by the debugger class. It encapsulates all the methods of the SymbolValues class and FunctionEnvironmentRecord class, which are called by the virtual machine.

Debugger Virtual Machine

The DebuggerVirtualMachine inherits from the Virtual Machine class. It encapsulates all the methods of the Debugger class. These methods are called from the Debugger Bytecode classes.

Function Environment Record

The Function Environment Record keeps track of the symbols in the current function and its offsets. It keeps track of the function details, like function name, start line, and end line. It also keeps track of the current line number of execution.

A new Function Environment Record is added when the Call Bytecode is executed and removed on the Return Bytecode. It is added on the Call Bytecode as a new function is called.

Debugger Shell

The DebuggerShell class executes the prompt, which requests a command from the user. It then checks if the command is valid. If the command is valid it returns the respective class of the command to the debugger, or else it prompts the user for a new command.

Debugger Bytecode

The Debugger Bytecode classes inherit from the DebuggerBytecode class. Each class continues its normal behaviour, with a few additional steps to support debugging behaviours. The byte codes that require additional code for handling debugging are lit, pop, call, line, function, formal, store, load, and return.

The lit, store, formal, and load bytecodes, keep track of the symbols and their offsets and values. The call bytecode adds a new function environment record. The return bytecode pops a frame and function environment record. The line bytecode sets the current line of execution. The function byte code updates the function environment record and adds a new frame that keeps track of symbols and their values.

Debugger Command

The DebuggerCommand class executes its respective methods in the debugger. It gets the required data from the debugger class and sends it to the UI classes for output.

The Continue command continues execution until a breakpoint is encountered, the list command lists the breakpoints, the locals command prints the symbols along with their values in the current function, the set command sets a breakpoint, the source command displays the source code, and the step command executes one bytecode. The list command shows the available commands and the exit command ends the debugger.

Code Organisation

I moved the debugger bytecodes in the interpreter.bytecode.debuggerbytecodes package.
I moved the debugger commands in the interpreter.debugger.commands package.
I moved the UI classes in the interpreter.debugger.ui package.
This resulted in the proper structure of my project.

Results and Conclusion

This project is a good introduction to debuggers that helps us enhance our understanding of compilers and debugging.

Challenges

They were a lot of test cases and a lot of edge cases that were hard to detect. They were also a lot of classes to implement, and how to use those classes to build a good architecture was a challenge.

Future Work

We can extend the debugger component to handle more bytecodes and accept more commands.