Laith Hussin
921659619
Apr 17-2022

Assignment 5 Documentation

Github Repository

https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-5---debugger-laith57th.git

Project Introduction and Overview

Using a code skeleton provided for the interpreter, debugger and other corresponding classes, I was required to improve the interpreter by adding debugger capabilities. To do this, I implemented three new bytecodes: Formal, Line, and Function. Additionally, I created a new DebuggerVirtualMachine that carries out executing the new bytecodes. I also implemented DebuggerCodeTable, FunctionEnvironmentRecord, and a user interface to interact with the user commands.

Scope of Work

Task		**Completed**
Replace assignment 4 implementations inside the debugger repository		**X**
	Replaced VirtualMachine, ByteCodeLoader, CodeTable, Program, and RunTimeStack.java	**X**
	Insert implemented bytecode classes into the bytecode package.	**X**
Create the new bytecode files.		**X**
	Create LineCode, FunctionCode, and FormalCode.	**X**
	Create a debug code file for pop, return, call, and lit bytecodes.	**X**
	Place all new bytecode files inside a “debuggercodes” package inside bytecode package.	**X**
Implement FunctionEnvironmentRecord class		**X**
	Create all necessary accessors for the instance variables and other useful methods to manipulate the symbol table.	**X**
	Use the Table class implementation from the constrainer algorithm to create the symbol table.	**X**
	Test the provided main method to make sure everything is in working order.	**X**
Implement the new bytecodes required for debugging		**X**
	Using the FunctionEnvironmentRecord class, implement each new bytecodes to satisfy the needed behavior.	
	Implement all other bytecodes that require additional behavior.	**X**
Add all new debugger bytecodes to the debuggerCodeTable class		**X**

	Add all debugger codes using the initialize method inside the debuggerCodeTable class.	**X**
Implement Debugger class		**X**
	Call the super class from the debugger constructor to initialize the interpreter bytecodes.	**X**
	Initialize all debugger bytecodes using the debuggerCodeTable implementation.	**X**
	Implement the run method to initialize debuggerVirtualMachine, print the source program, then call the DebuggerShell class to prompt the user for a command.	**X**
Create command files		**X**
	Inside the debugger package, create a commands file that contains continue, list, locals, set, source, and step commands.	**X**
	All the commands must extend the debuggerCommand class and be passed as an instance when the command is entered from the UI.	**X**
Implement DebuggerVirtualMachine		**X**
	Create an arraylist of Entry objects that contain the line number, source line and whether the source line sits on a breakpoint.	**X**
	Implement all necessary accessors and command methods to carry out the correct behavior.	**X**
Test the debugger		**X**

	Using the provided factorial source file and bytecode file, I tested my program using the implemented commands:
	
	>javac interpreter/Interpreter.java
	> java interpreter.Interpreter -d sample_files/factorial
	1: program {boolean j int i
	2: int factorial(int n) {
	3: if (n < 2) then
	4: { return 1 }
	5: else
	6: {return n*factorial(n-1) }
	7: }
	8: while (1==1) {
	9: i = write(factorial(read()))
	10: }
	11: }
	
	Type ? for help
	>set
	
	Set a break point at line: >
		X

```
3

Type ? for help
>set

Set a break point at line: >
9

Type ? for help
>step

Type ? for help
>continue

Type ? for help
>locals
i: 0
j: 0

Type ? for help
>continue
Please input an integer: 3

Type ? for help
>source
  2:   int factorial(int n) {
* 3:       if (n < 2) then
  4:         { return 1 }
  5:       else
  6:         {return n*factorial(n-1) }
  7:   }

Type ? for help
>locals
n: 3

Type ? for help
>continue

Type ? for help
>locals
n: 2

Type ? for help
>continue

Type ? for help
>continue
6

Type ? for help
>exit
```

	The program successfully executed the correct behavior.	

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

The program ran as expected and no error messages were displayed

Assumptions

I assumed a correct and existing file name will be passed to the debugger.

Summary of technical work

To complete this project I implemented a strategy design pattern to dynamically choose the behavior executed for each scanned debugger bytecodes from the token stream. Also, I created an instance of each command that executes according to user input. Finally, I incorporated object oriented programming when creating an entry object for each source file and loading the information to each entry in the arraylist for easier access and better organization.

Implementation

New ByteCode subclasses

1. LineCode
    a. This debugger code sets the current line number to the value read from the bytecode file. This was somewhat straightforward to implement and test.
2. FunctionCode
    a. The function code takes a start, end, and function name instance variables from the token file and sets the function information inside the function environment record.
3. FormalCode
    a. The formal code takes a name and an offset and initializes them with the information read from the token stream. The execute method adds this information to the HashMap of the environment record on top of the functionEnvironmentRecord stack.

4. debugReturnCode
    a. The new behavior required for this bytecode is to pop the FunctionEnvironmentStack in the debugger virtual machine.
5. debugPopCode
    a. This bytecode reads a level integer then deletes that amount of items from the symbol table.
6. DebugLitCode
    a. This bytecode adds the literal value to the hash map on top of the environment record stack if an identifier is present in the token stream.
7. debugCallCode
    a. This bytecode simply pushes a new function environment record on top of the stack.

FunctionEnvironmentRecord class

To implement this class, I used the Table class algorithm from the constrainer. First, I created the symbol table as an instance of the Table class. Then I implemented all the necessary functions to carry out the correct behavior. Finally, I tested the methods using the provided main method which returned the desired output.

DebuggerCodeTable class

This class was fairly straightforward. I first added all the debugger bytecodes, as well as, the bytecodes that required additional behavior to the codeMap object. Next, I created a static method to access the hashMap and use in the bytecodeLoader class.

Debugger class

To implement this class's constructor, I called the super constructor and passed the baseFileName as well as a debug boolean to differentiate between modes. Next, I concatenated the correct extension to the baseFileName. Finally, I initialized the DebuggerCodeTable with the new bytecodes. For the run method, I created a new DebuggerVirtualMachine instance and passed the current program as well as the source file. Then I printed the source code to the screen followed by the prompt method inside the shell class.

DebuggerVirtualMachine class

This class was essentially the motherboard to my implementation. The main job of this class is to create the source file entries using an arraylist of an Entry object. This class also initializes the functionEnvironmentStack and loads the byteCodes from the file. The methods implemented in this class could be considered helper methods for all the command instances.

DebuggerShell Class

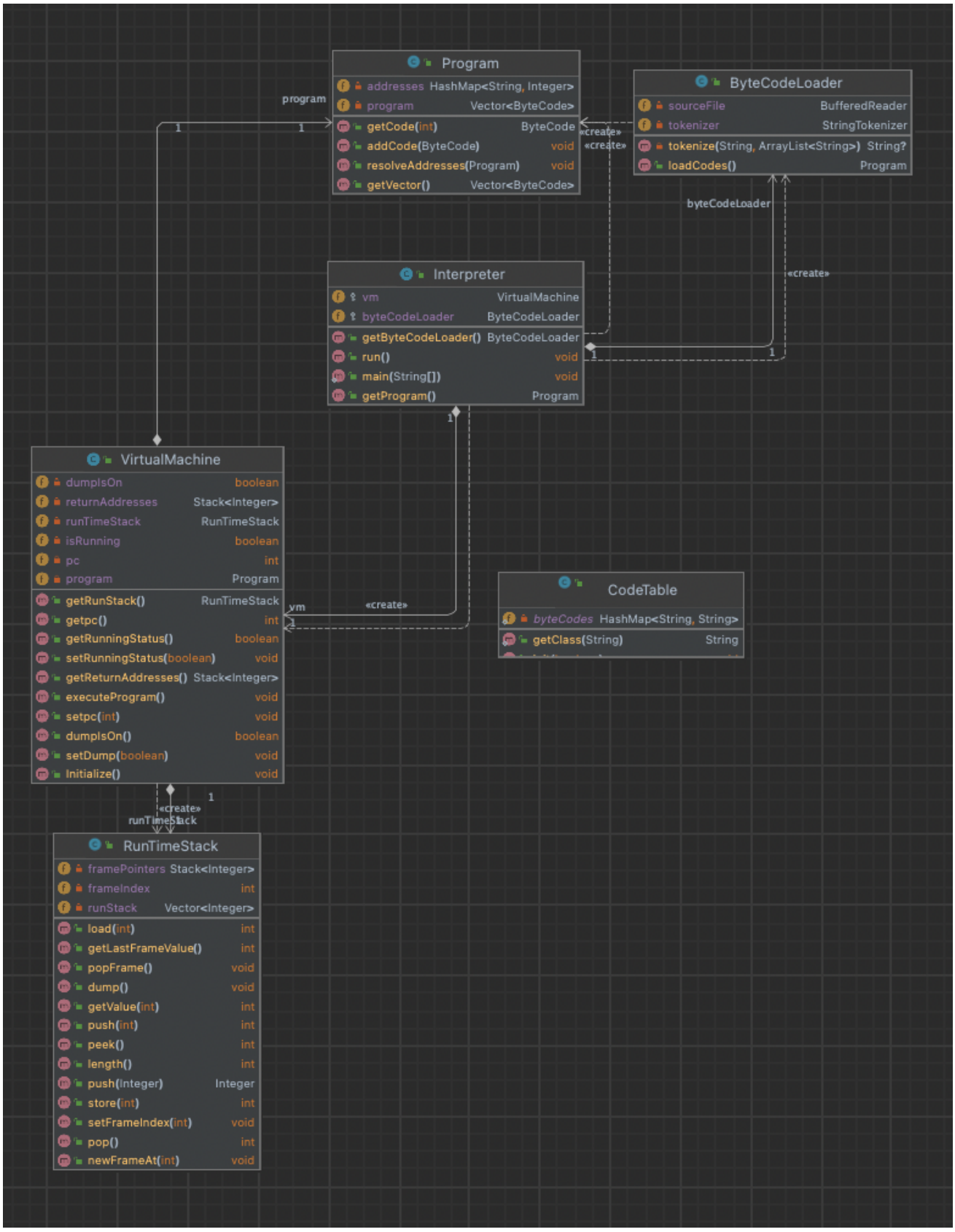
This class is used to interact with the user and carry out the entered commands. The prompt method is essentially a while loop that continues to ask the user for new commands until they decide to exit the program. The user input is checked through a switch statement that passes an instance of the entered command and executes the respective method for that command.

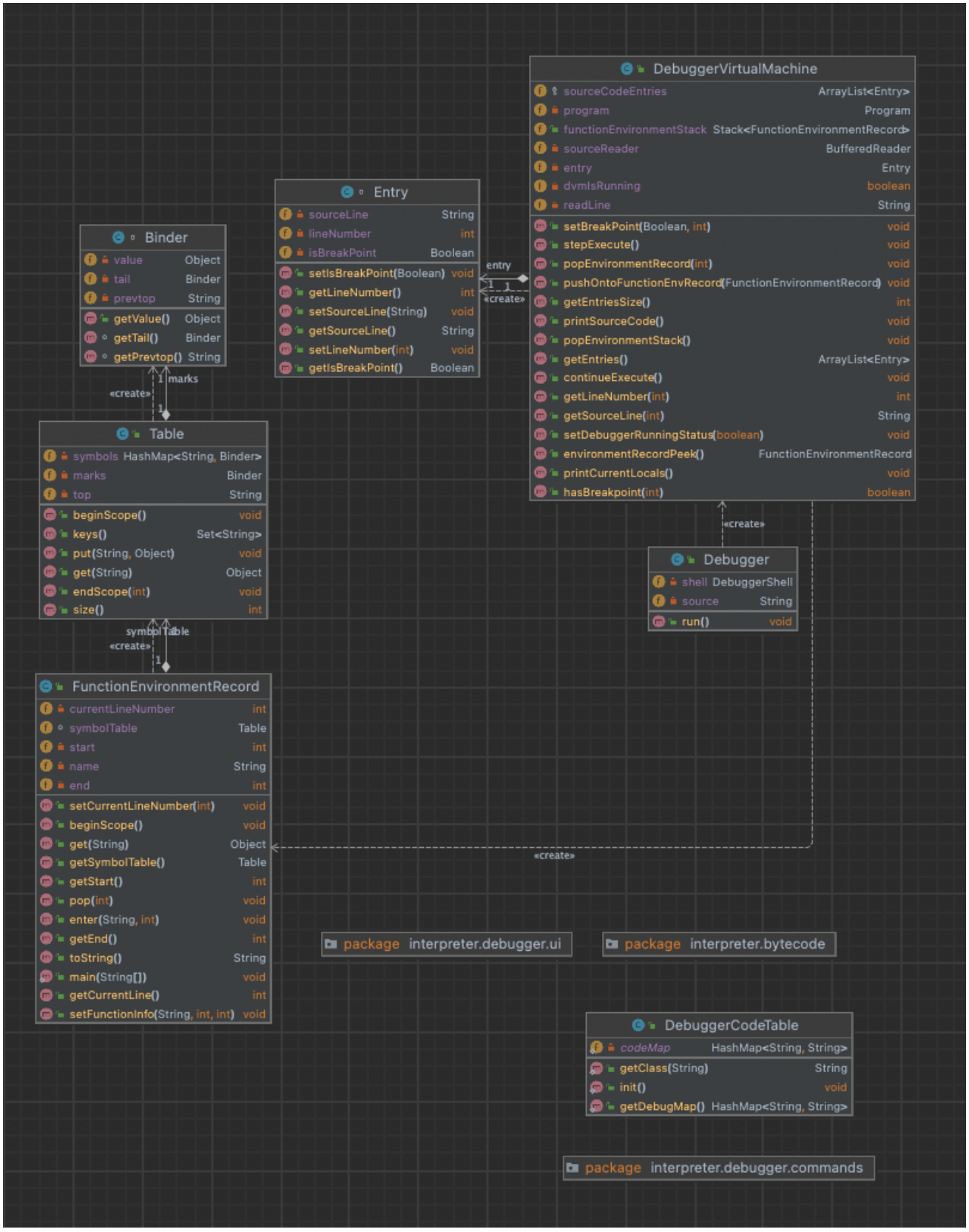
Code Organization

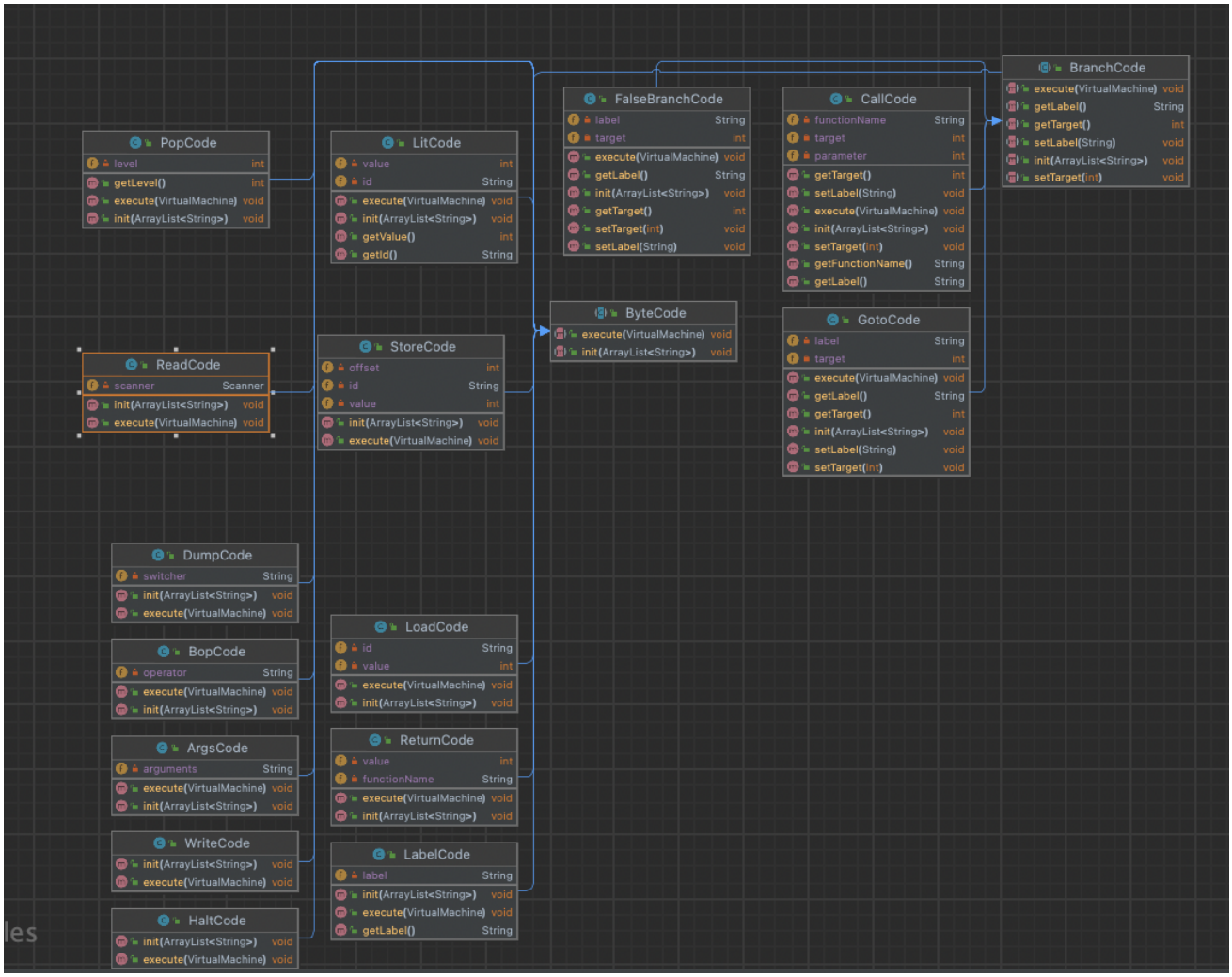
To better organize my code, I used single-function methods for each command. Also, I created an entry object that holds all source line information. Finally, I placed all command subclasses inside a

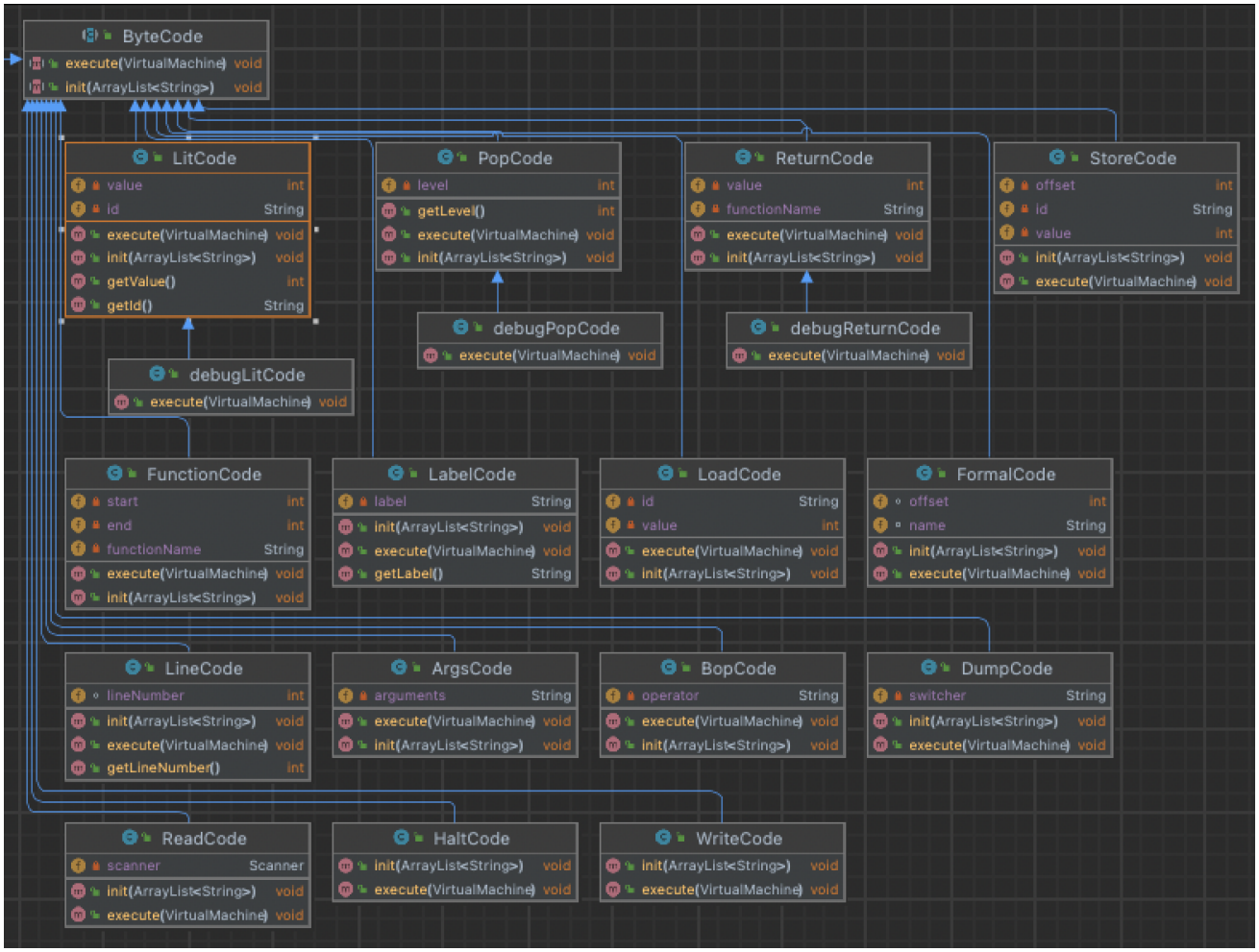
commands package inside the debugger package for easier access and debugging.

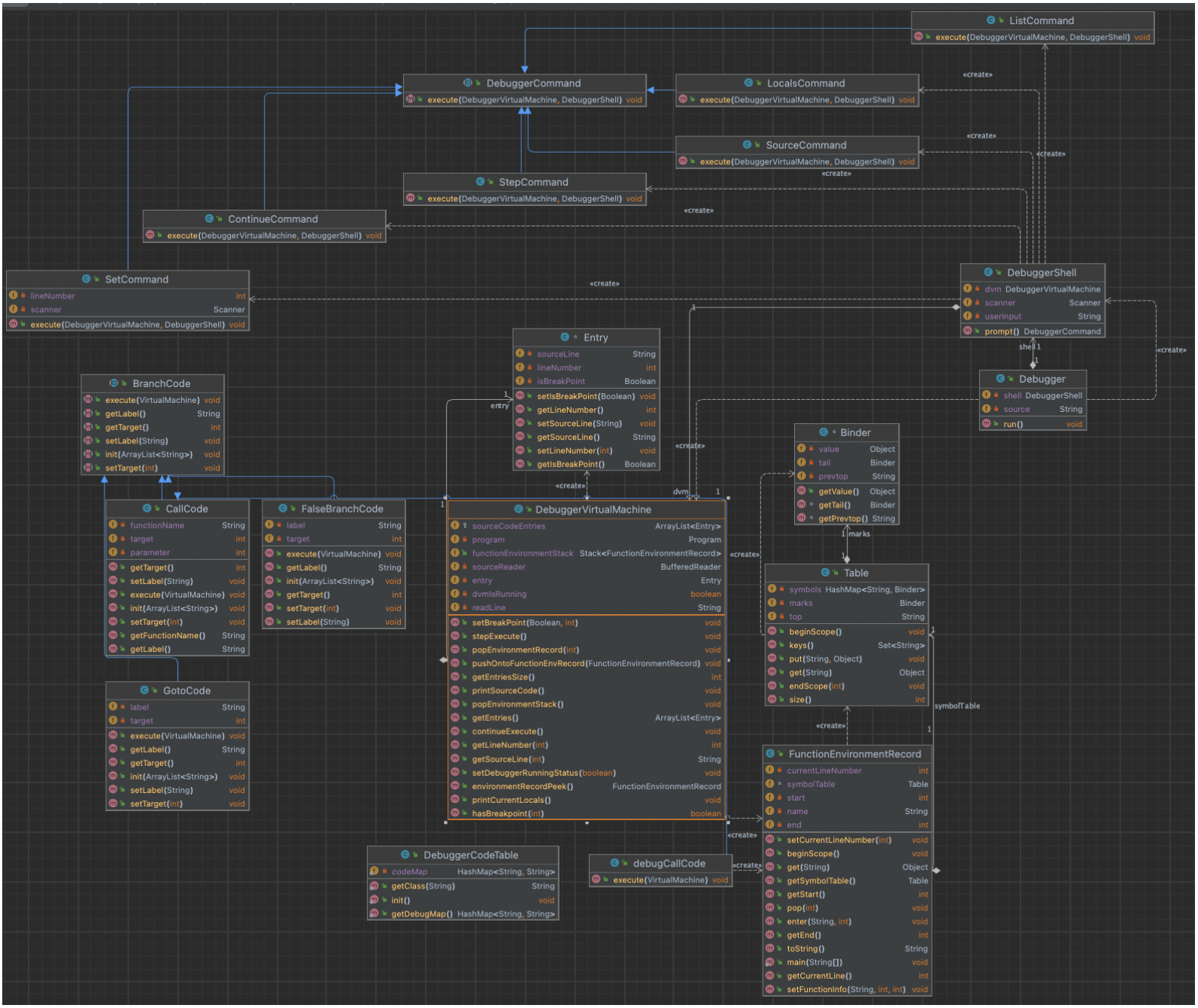
Class Diagram




DebuggerVirtualMachine
sourceCodeEntries ArrayList<Entry>
program Program
functionEnvironmentStack Stack<FunctionEnvironmentRecord>
sourceReader BufferedReader
entry Entry
dvmIsRunning boolean
readLine String
setBreakPoint(Boolean, int) void
stepExecute() void
popEnvironmentRecord(int) void
pushOntoFunctionEnvRecord(FunctionEnvironmentRecord) void
getEntriesSize() int
printSourceCode() void
popEnvironmentStack() void
getEntries() ArrayList<Entry>
continueExecute() void
getLineNumber(int) int
getSourceLine(int) String
setDebuggerRunningStatus(boolean) void
environmentRecordPeek() FunctionEnvironmentRecord
printCurrentLocals() void
hasBreakpoint(int) boolean
Entry
sourceLine String
lineNumber int
isBreakPoint Boolean
setIsBreakPoint(Boolean) void
getLineNumber() int
setSourceLine(String) void
getSourceLine() String
setLineNumber(int) void
getIsBreakPoint() Boolean
entry
1
1
«create»
Binder
value Object
tail Binder
prevtop String
getValue() Object
getTail() Binder
getPrevtop() String
1
marks
«create»
1
Table
symbols HashMap<String, Binder>
marks Binder
top String
beginScope() void
keys() Set<String>
put(String, Object) void
get(String) Object
endScope(int) void
size() int
symbolTable
«create»
1
FunctionEnvironmentRecord
currentLineNumber int
symbolTable Table
start int
name String
end int
setCurrentLineNumber(int) void
beginScope() void
get(String) Object
getSymbolTable() Table
getStart() int
pop(int) void
enter(String, int) void
getEnd() int
toString() String
main(String[]) void
getCurrentLine() int
setFunctionInfo(String, int, int) void
«create»
Debugger
shell DebuggerShell
source String
run() void
«create»
«create»
package interpreter.debugger.ui
package interpreter.bytecode
DebuggerCodeTable
codeMap HashMap<String, String>
getClass(String) String
init() void
getDebugMap() HashMap<String, String>
package interpreter.debugger.commands


PopCode
level int
getLevel() int
execute(VirtualMachine) void
init(ArrayList<String>) void
LitCode
value int
id String
execute(VirtualMachine) void
init(ArrayList<String>) void
getValue() int
getId() String
FalseBranchCode
label String
target int
execute(VirtualMachine) void
getLabel() String
init(ArrayList<String>) void
getTarget() int
setTarget(int) void
setLabel(String) void
CallCode
functionName String
target int
parameter int
getTarget() int
setLabel(String) void
execute(VirtualMachine) void
init(ArrayList<String>) void
setTarget(int) void
getFunctionName() String
getLabel() String
BranchCode
execute(VirtualMachine) void
getLabel() String
getTarget() int
setLabel(String) void
init(ArrayList<String>) void
setTarget(int) void
ReadCode
scanner Scanner
init(ArrayList<String>) void
execute(VirtualMachine) void
StoreCode
offset int
id String
value int
init(ArrayList<String>) void
execute(VirtualMachine) void
ByteCode
execute(VirtualMachine) void
init(ArrayList<String>) void
GotoCode
label String
target int
execute(VirtualMachine) void
getLabel() String
getTarget() int
init(ArrayList<String>) void
setLabel(String) void
setTarget(int) void
DumpCode
switcher String
init(ArrayList<String>) void
execute(VirtualMachine) void
BopCode
operator String
execute(VirtualMachine) void
init(ArrayList<String>) void
ArgsCode
arguments String
execute(VirtualMachine) void
init(ArrayList<String>) void
WriteCode
init(ArrayList<String>) void
execute(VirtualMachine) void
HaltCode
init(ArrayList<String>) void
execute(VirtualMachine) void
LoadCode
id String
value int
execute(VirtualMachine) void
init(ArrayList<String>) void
ReturnCode
value int
functionName String
execute(VirtualMachine) void
init(ArrayList<String>) void
LabelCode
label String
init(ArrayList<String>) void
execute(VirtualMachine) void
getLabel() String


ByteCode
execute(VirtualMachine) void
init(ArrayList<String>) void
LitCode
value int
id String
execute(VirtualMachine) void
init(ArrayList<String>) void
getValue() int
getId() String
PopCode
level int
getLevel() int
execute(VirtualMachine) void
init(ArrayList<String>) void
ReturnCode
value int
functionName String
execute(VirtualMachine) void
init(ArrayList<String>) void
StoreCode
offset int
id String
value int
init(ArrayList<String>) void
execute(VirtualMachine) void
debugPopCode
execute(VirtualMachine) void
debugReturnCode
execute(VirtualMachine) void
debugLitCode
execute(VirtualMachine) void
FunctionCode
start int
end int
functionName String
execute(VirtualMachine) void
init(ArrayList<String>) void
LabelCode
label String
init(ArrayList<String>) void
execute(VirtualMachine) void
getLabel() String
LoadCode
id String
value int
execute(VirtualMachine) void
init(ArrayList<String>) void
FormalCode
offset int
name String
init(ArrayList<String>) void
execute(VirtualMachine) void
LineCode
lineNumber int
init(ArrayList<String>) void
execute(VirtualMachine) void
getLineNumber() int
ArgsCode
arguments String
execute(VirtualMachine) void
init(ArrayList<String>) void
BopCode
operator String
execute(VirtualMachine) void
init(ArrayList<String>) void
DumpCode
switcher String
init(ArrayList<String>) void
execute(VirtualMachine) void
ReadCode
scanner Scanner
init(ArrayList<String>) void
execute(VirtualMachine) void
HaltCode
init(ArrayList<String>) void
execute(VirtualMachine) void
WriteCode
init(ArrayList<String>) void
execute(VirtualMachine) void


ListCommand
execute(DebuggerVirtualMachine, DebuggerShell) void
DebuggerCommand
execute(DebuggerVirtualMachine, DebuggerShell) void
LocalsCommand
execute(DebuggerVirtualMachine, DebuggerShell) void
SourceCommand
execute(DebuggerVirtualMachine, DebuggerShell) void
StepCommand
execute(DebuggerVirtualMachine, DebuggerShell) void
ContinueCommand
execute(DebuggerVirtualMachine, DebuggerShell) void
«create»
SetCommand
lineNumber int
scanner Scanner
execute(DebuggerVirtualMachine, DebuggerShell) void
DebuggerShell
dvm DebuggerVirtualMachine
scanner Scanner
userInput String
prompt() DebuggerCommand
shell
Debugger
shell DebuggerShell
source String
run() void
Entry
sourceLine String
lineNumber int
isBreakPoint Boolean
setIsBreakPoint(Boolean) void
getLineNumber() int
setSourceLine(String) void
getSourceLine() String
setLineNumber(int) void
getIsBreakPoint() Boolean
entry
BranchCode
execute(VirtualMachine) void
getLabel() String
getTarget() int
setLabel(String) void
init(ArrayList<String>) void
setTarget(int) void
CallCode
functionName String
target int
parameter int
getTarget() int
setLabel(String) void
execute(VirtualMachine) void
init(ArrayList<String>) void
setTarget(int) void
getFunctionName() String
getLabel() String
FalseBranchCode
label String
target int
execute(VirtualMachine) void
getLabel() String
init(ArrayList<String>) void
getTarget() int
setTarget(int) void
setLabel(String) void
GotoCode
label String
target int
execute(VirtualMachine) void
getLabel() String
getTarget() int
init(ArrayList<String>) void
setLabel(String) void
setTarget(int) void
DebuggerVirtualMachine
sourceCodeEntries ArrayList<Entry>
program Program
functionEnvironmentStack Stack<FunctionEnvironmentRecord>
sourceReader BufferedReader
entry Entry
dvmIsRunning boolean
readLine String
setBreakPoint(Boolean, int) void
stepExecute() void
popEnvironmentRecord(int) void
pushOntoFunctionEnvRecord(FunctionEnvironmentRecord) void
getEntriesSize() int
printSourceCode() void
popEnvironmentStack() void
getEntries() ArrayList<Entry>
continueExecute() void
getLineNumber(int) int
getSourceLine(int) String
setDebuggerRunningStatus(boolean) void
environmentRecordPeek() FunctionEnvironmentRecord
printCurrentLocals() void
hasBreakpoint(int) boolean
dvm
Binder
value Object
tail Binder
prevtop String
getValue() Object
getTail() Binder
getPrevtop() String
marks
Table
symbols HashMap<String, Binder>
marks Binder
top String
beginScope() void
keys() Set<String>
put(String, Object) void
get(String) Object
endScope(int) void
size() int
symbolTable
FunctionEnvironmentRecord
currentLineNumber int
symbolTable Table
start int
name String
end int
setCurrentLineNumber(int) void
beginScope() void
get(String) Object
getSymbolTable() Table
getStart() int
pop(int) void
enter(String, int) void
getEnd() int
toString() String
main(String[]) void
getCurrentLine() int
setFunctionInfo(String, int, int) void
DebuggerCodeTable
codeMap HashMap<String, String>
getClass(String) String
init() void
getDebugMap() HashMap<String, String>
debugCallCode
execute(VirtualMachine) void

Results and Conclusion

This project proved to be a good improvement on the previously implemented interpreter. I was able to implement all the features of the debugger successfully and satisfied all the required behavior. Overall, This was a good conclusion to this course that brought everything I learned together.

Challenges

This project was very complex and contained many moving parts. With this being the final project, I wasn't surprised at the complexity and began working on it with that expectation in mind. What really saved me in this project was the debugger. After being stuck on the implementation of some classes, I decided to learn how to use the visual studio code debugger which proved to be very useful and powerful.

Future Work

For the future, we can implement more logic to the x code that can handle more complex programs. We can also add more features to the debugger such as a command that restarts the program execution.