Duccio Rocca
ID = 922254031
May 18th, 2022

Assignment 5 Documentation

Github Repository

Assignment five repository.

Or

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

Project Introduction and Overview

This project required me to implement a Debugger starting from a skeleton code. It required me to implement some classes from scratch and to complete others starting from a template. I was allowed to import some classes from the previous assignment. Some of them required some modification to adjust for the new Debugger features.

The assignment was divided into two parts, the first part required me to modify some parts of the Interpreter so that be run in debugging mode. The second part required me to implement a User Interface that would allow the user to interact with the debugger, placing some breakpoints and printing some information related to the program's execution.

Execution and Development Environment

As IDE I used Visual Studio Code on my MacBook Pro and I tested in both IDE and terminal through the command line.

JDK is Openjdk, version "17.0.2" 2022-01-18. The OpenJDK Runtime Environment is build 17.0.2+8-86, and OpenJDK 64-Bit Server VM is build 17.0.2+8-86 (mixed mode, sharing).

Assumptions

I assumed that the bytecode programs used are generated correctly, and therefore should not contain any syntactical errors.

I also assumed that the source code file wasn't containing any syntactical errors.

Scope of Work

Task		Completed
Provide implementations for new byte codes discussed in class		X
	LINE	X
	FUNCTION	X
	FORMAL	X
The byte code classes already existing that needed to be modified		X
	CALL	X
	LIT	X
	POP	X
	RETURN	X
Implement a FunctionEnvironmentRecord class		X
	Symbol Table	X
	Function Name	X
	Start line	X
	End line	X
DebuggerCodeTable		X
Debugger Class		X
Vector of Entries		X
Entry Class		X
	lineNumber	X
	sourceLine	X
	isBreakpointLine	X
Stack of FunctionEnvironmentRecord		X
SourceCodeLoader		X
Strategy Pattern to implement DebuggersCommands		X
UI implementation and all the commands		X
	? Command	X
	Set Command	X
	Locals Command	X
	List Command	X
	Source Command	X
	Step Command	X
	Continue Command	X
DebuggerShell Class		X
DebuggerVirtualMachine		X
Replacing the classes from Assignment 4		X
	VirtualMachine	X
	ByteCodeLoader	X
	CodeTable	X
	Program	X
	RuntimeStack	X
DebuggerByteCodeLoader		X

Compilation Result

First, I compiled all the Commands in the ui package with the following command:

```
javac interpreter/debugger/ui/*.java
```

I compiled all the Bytecodes in the bytecode package with the following command:

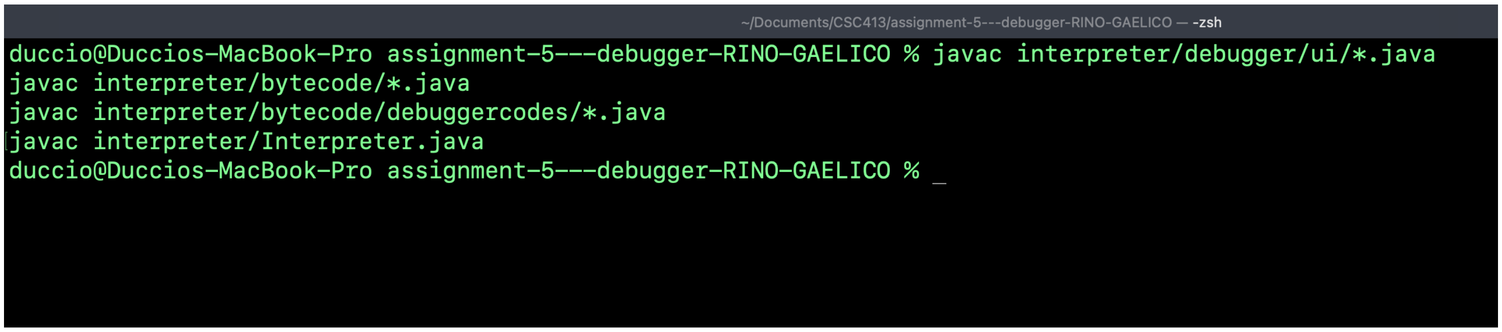
```
javac interpreter/bytecode/*.java
```

I compiled all the Debugger Bytecodes in the bytecode package with the following command:

```
javac interpreter/bytecode/debuggercodes/*.java javac
```

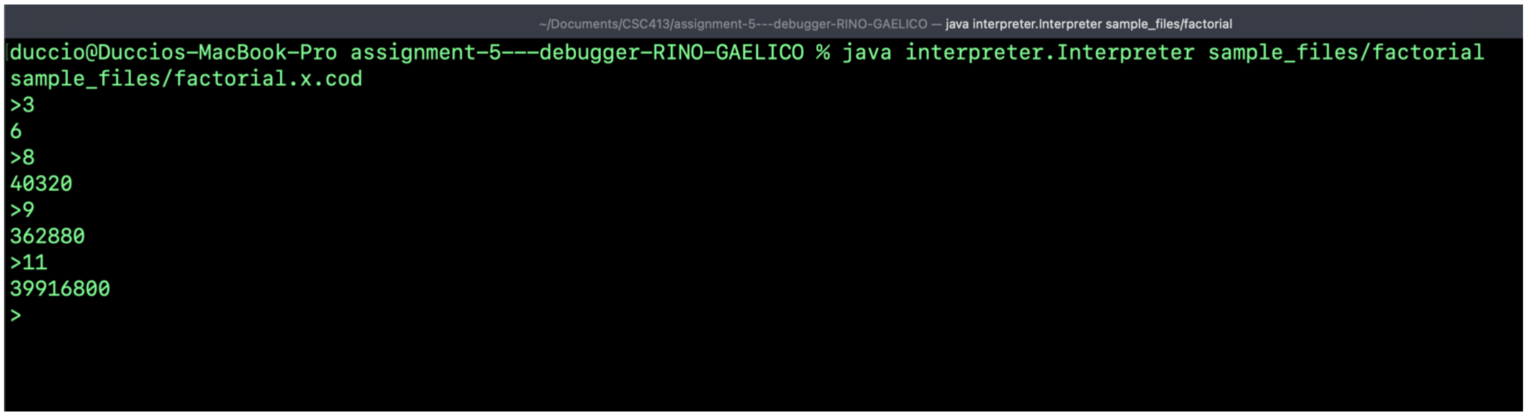
I compiled the Interpeter class with the following command:

```
javac interpreter/Interpreter.java
```



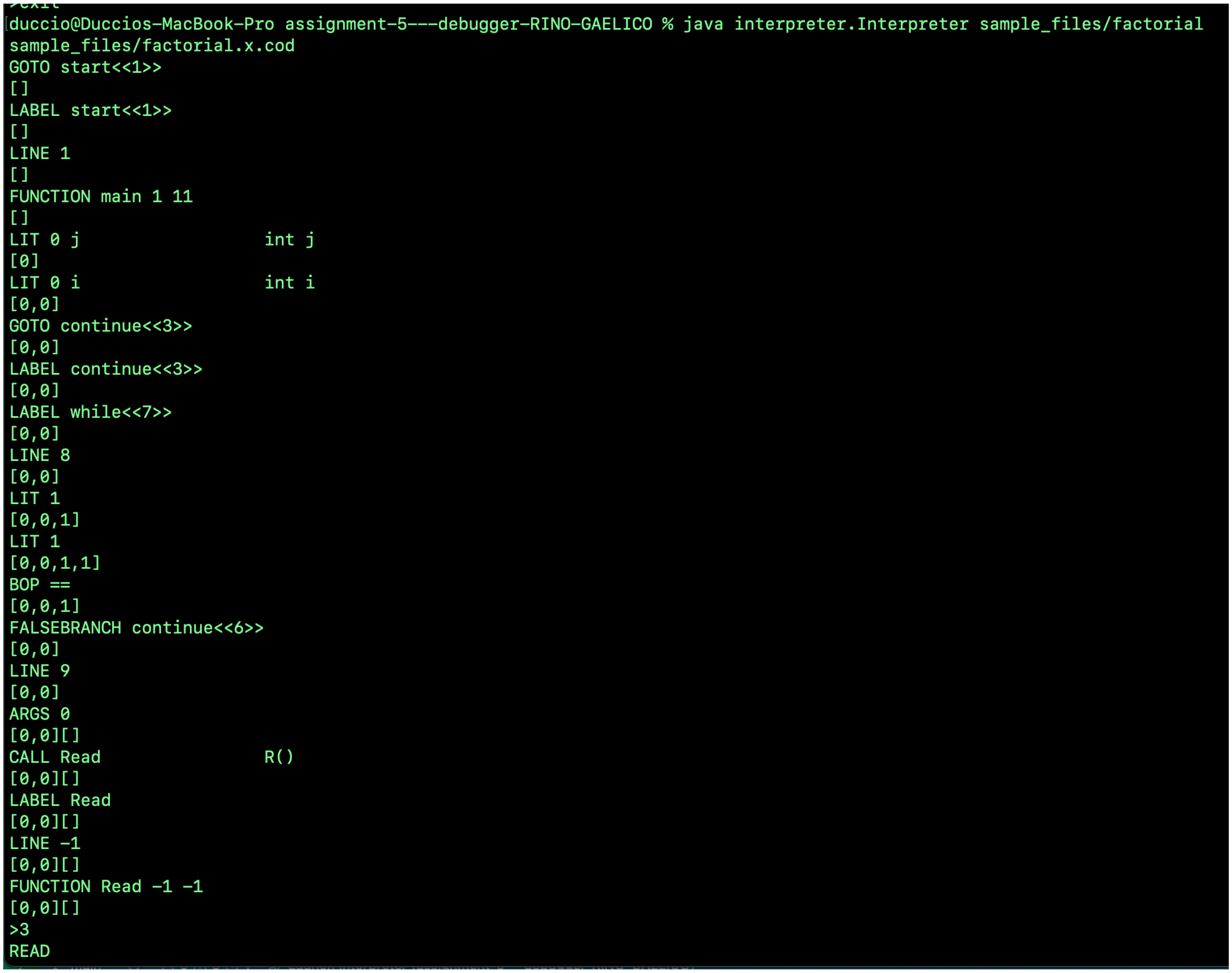
```
~/Documents/CSC413/assignment-5---debugger-RINO-GAELICO — -zsh
duccio@Duccios-MacBook-Pro assignment-5---debugger-RINO-GAELICO % javac interpreter/debugger/ui/*.java
javac interpreter/bytecode/*.java
javac interpreter/bytecode/debuggercodes/*.java
javac interpreter/Interpreter.java
duccio@Duccios-MacBook-Pro assignment-5---debugger-RINO-GAELICO % _
```

Finally I first ran without any debugging option activated:



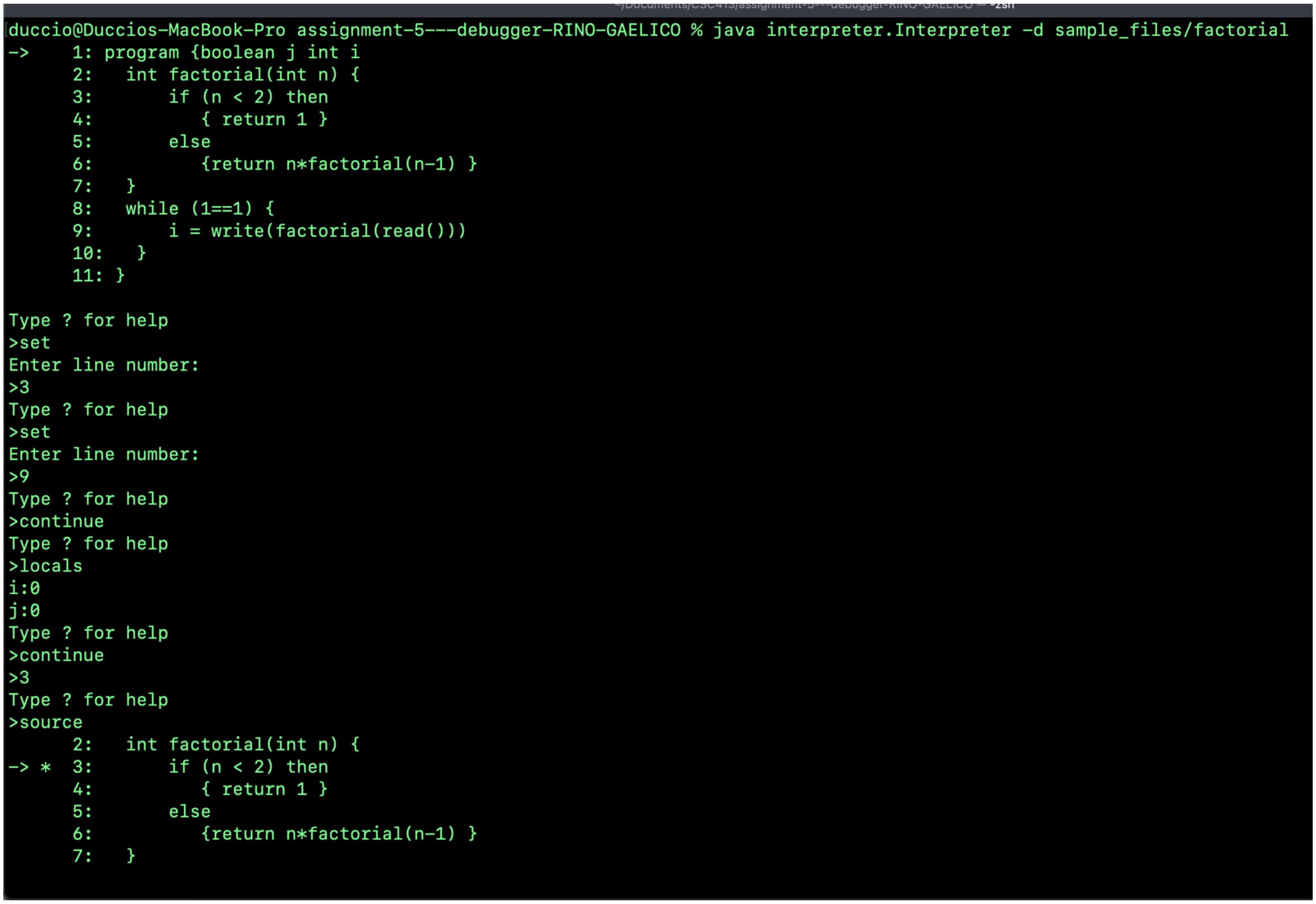
```
~/Documents/CSC413/assignment-5---debugger-RINO-GAELICO — java interpreter.Interpreter sample_files/factorial
duccio@Duccios-MacBook-Pro assignment-5---debugger-RINO-GAELICO % java interpreter.Interpreter sample_files/factorial
sample_files/factorial.x.cod
>3
6
>8
40320
>9
362880
>11
39916800
>
```

Then I ran with DUMP ON activated to make sure everything was ok:



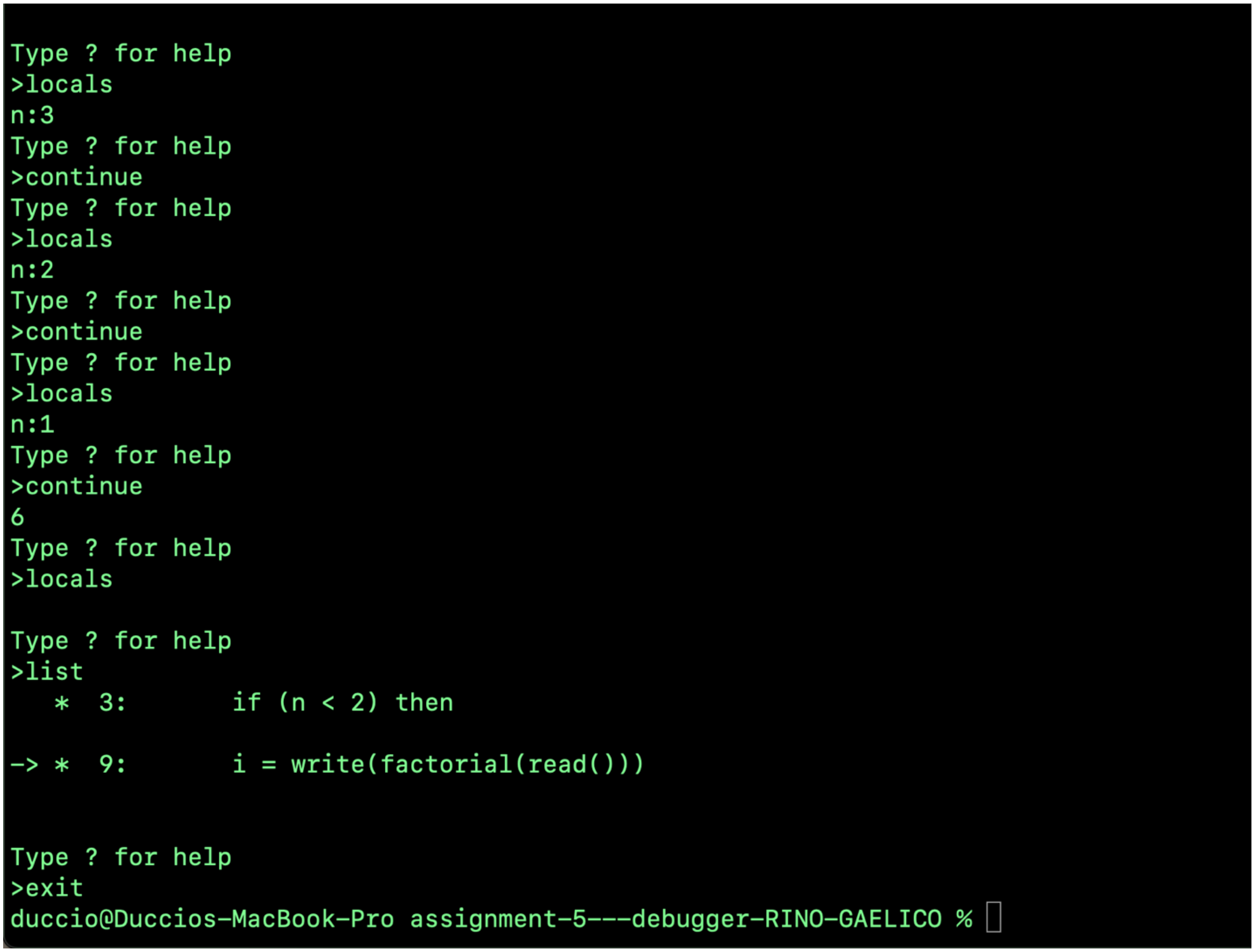
```
>exit
duccio@Duccios-MacBook-Pro assignment-5---debugger-RINO-GAELICO % java interpreter.Interpreter sample_files/factorial
sample_files/factorial.x.cod
GOTO start<<1>>
[]
LABEL start<<1>>
[]
LINE 1
[]
FUNCTION main 1 11
[]
LIT 0 j                  int j
[0]
LIT 0 i                  int i
[0,0]
GOTO continue<<3>>
[0,0]
LABEL continue<<3>>
[0,0]
LABEL while<<7>>
[0,0]
LINE 8
[0,0]
LIT 1
[0,0,1]
LIT 1
[0,0,1,1]
BOP ==
[0,0,1]
FALSEBRANCH continue<<6>>
[0,0]
LINE 9
[0,0]
ARGS 0
[0,0][]
CALL Read                R()
[0,0][]
LABEL Read
[0,0][]
LINE -1
[0,0][]
FUNCTION Read -1 -1
[0,0][]
>3
READ
```

I finally ran with debug mode activated:



```
duccio@Duccios-MacBook-Pro assignment-5---debugger-RINO-GAELICO % 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
Enter line number:
>3
Type ? for help
>set
Enter line number:
>9
Type ? for help
>continue
Type ? for help
>locals
i:0
j:0
Type ? for help
>continue
>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
>locals
n:1
Type ? for help
>continue
6
Type ? for help
>locals

Type ? for help
>list
   *  3:       if (n < 2) then

-> *  9:       i = write(factorial(read()))


Type ? for help
>exit
duccio@Duccios-MacBook-Pro assignment-5---debugger-RINO-GAELICO % 
```

Implementation

Bytecodes Debugger

The first requirement for this assignment was to implement the Bytecode classes. We were allowed to use some of the Bytecode classes from the previous assignment, but we also had to implement new ones and we had to modify some of the existing ones to adjust them to the Debugging features.
The Bytecode classes in the debuggercodes package are the classes that I implemented. They extend the “normal” version so that they are gonna be used only when the debug mode is activated. Since they are grandchildren of the Bytecode class they override the execute method.
In this method, they simply call the corresponding method in the DebuggerVirtualMachine class that provides a routine specific to the bytecode class visited. This design pattern is the same as the Visitor design pattern that we used in the Parser.

LINE, FORMAL, FUNCTION and others Bytecode Debuggers

Line is a bytecode class that updates the current source code line that we are executing. The value had to be updated in the FunctionEnvironmentRecord and a tracker variable inside the DVM that keep track of the current line of execution. This bytecode class did not represent a major challenge, but it was a crucial part of the program since it is this bytecode that allows us to know which line we are visiting.
Formal bytecode class has the task to upload the parameter and its value on the FunctionEnvironmentRecord. This class’ implementation was pretty straightforward, its only difficulty is represented by the interaction with the FunctionEnvironmentRecord, but since I had already implemented a method to update the record in the FER the rest of the implementation resulted fairly easy.
Function bytecode class is the one responsible for creating the FER and providing it with some important information, such as the function name, and starting line end line. This information will then be very useful when it comes to the debugging options.
The other existing bytecode classes that I had to modify are: Call, Return, LIT and POP.
Call and Return mostly interact with the FER stack because they are responsible for creating a new function and returning from a function. On the other hand, Lit and Pop classes interact mostly with the runtime stack.

FunctionEnvironmentRecord and supporting classes

The FunctionEnvironmentRecord was one of the most difficult parts of the whole project. First of all, I did not study in detail how the Constrainer works, so I did not have a clear idea of how the symbol table worked. I had to watch the lecture again and study the slides.
After I grasped the idea behind the table and how the Binder class is structured I simply imported some parts of the Constrainer code into the new assignment. I still had to make some adjustments since it required some additional features.
The FER has a crucial role in the debugger project because it allows us to keep track of the current function and its parameters. One of the tricky aspects is that within a function there could be more than one scope.

Furthermore, the FER keeps track of the current line of execution, which represents a very important piece of information when it comes to debugging and breakpoints.

DebuggerCodeTable Class

The DebuggerCodeTable class had a relatively easy implementation since it mostly behaved as the regular CodeTable. Like the latter, it has a static method that initializes a HashMap that would associate each new Bytecode with its proper className.
Its purpose is to provide a map that could be used by the ByteCodeLoader to obtain the correct Class name for each Bytecode so that it could use a reflective technique to create an instance of that particular Bbytecode Class.
The method get() first searches for that bytecode inside the regular CodeTable's map and only if it doesn't find it, it will then go and retrieve it from its map.
The implementation of this class represented one of the simplest requirements for this assignment.

SourceCodeLoader Class

It took me a fair amount of time before I understood that I needed this class to load the source code file. I did not understand at first how to simultaneously work with two different file types. I thought it was the responsibility of the BytecodeLoader class to load also the source code file, but then I realized that the best solution was to give this responsibility to a separate class that would also contain an instance of a class representing the whole file in the form of an array of lines. I decided to call this class SourceFileEntry.
The implementation of the SourceCodeLoader class was pretty straightforward, it simply emulated the work done by the Bytecode Loader but it loads a source code file instead.

SourceFileEntry and Entry Classes

The SourceFileEntry class's implementation was fairly simple, its job is to store an array of Entries, and each Entry represents the source code's line.
It allows for interaction with the single entry and retrieves important information such as line number and the existence of breakpoints associated with the line.
One of the most complicated aspects of this class's implementation is the toString() method. This method has the delicate responsibility of correctly representing the breakpoints and the current line of execution. For this method, I used a StringBuilder class and an Iterator, for each line I verify if a breakpoint has been set and if the line under scrutiny is the current line of execution.

DebuggerVirtualMachine class

This class took away a lot of my time since it is a class with more code in it than any other class. At the same time, its implementation is pretty straightforward.
Its responsibility is to execute byte codes and to maintain the state of the runtime stack and the FER stack. It also acts as a bridge between the debugger and the byte codes.
As I mentioned earlier, following the idea of a Visitor design pattern I implemented some methods in this class that is related to the bytecodes that the DVM visits.
The heart of the class is definitely the execute method which allows the execution of each bytecode and also establishes if a breakpoint has been encountered. Once a breakpoint has been encountered the DVM interrupts the execution and passes the control back to the Debugger.

Debugger class

This class is a child class of the Interpreter, therefore the Debugger class is an Inrterpreter. Its job is to run the program and to terminate it once the user has sends the appropriate command. The debugger also calls the DebuggerShell class and uses it to execute the single commands. This class is definitely very important in the overall design but it is not complicated and its implementation was pretty much done for us. I only added minor changes to it.

UI and Command Classes

One of the most difficult parts of the project was to correctly implement each of these commands.
For their implementation we followed the Strategy design pattern, using an abstract class that provides an execute method that every single command has to implement and also a static method. The static method allows us to take advantage of the polymorphism and the dynamic binding since we can let the runtime context determine the specific command that will be created.
The first command is the question mark command it simply displays available commands. This command was pretty simple to implement, I used a stream starting from the commands map inside the DebuggerCommand class.
The set command records that a breakpoint has been set for a specific line. Its implementation is pretty straightforward, I added a try and catch block so that I can validate the user input.
The list command was one of the easiest. It simply lists all breakpoints currently set with the debugger. I just used a for-each block to traverse the Vector of Entries. I created a string with a StringBuilder object appending each line that resulted to have a breakpoint set.
For the source command, I had to use a for loop that would traverse the source code from the starting to the ending point provided by the FER.
Step and Continue are pretty similar since both causes the DVM to execute, in the case of continuing it uses the execute() method, while for step I had to implement a brand new method that simply runs one bytecode at a time.
Locals were one of the most difficult because they were required to retrieve different pieces of information from different objects. The FER provided the name of the variables that had to be displayed and the RuntimeStack provided the actual values corresponding to each variable. I

implemented a new method in the RuntimeStack to retrieve the correct values from the current frame.
Finally, exit has the responsibility to interrupt the loop that executes the program in the debugger.

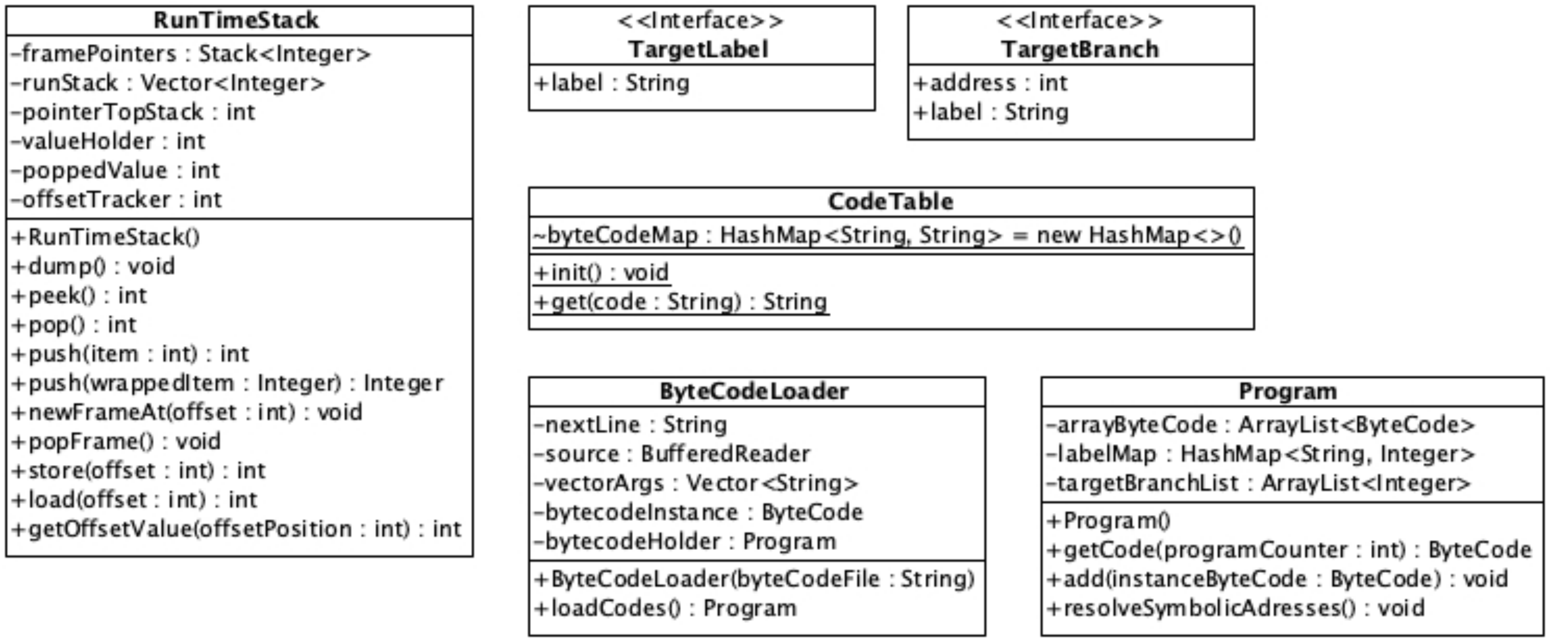
Code Organization

The classes were already organized into packages, I added the new Bytecodes for debugging purposes inside the debuggercodes package into the Bytecode package and create a new folder for documentation. I also imported the Bytecode and other classes as per instructions. I added some command classes to the commands package inside the Ui package.

Class Diagram

The class diagrams shown below represent all the classes in this project.

Here are the classes directly contained by the interpreter package that have no relationship with other classes:



```
RunTimeStack
-framePointers : Stack<Integer>
-runStack : Vector<Integer>
-pointerTopStack : int
-valueHolder : int
-poppedValue : int
-offsetTracker : int
+RunTimeStack()
+dump() : void
+peek() : int
+pop() : int
+push(item : int) : int
+push(wrappedItem : Integer) : Integer
+newFrameAt(offset : int) : void
+popFrame() : void
+store(offset : int) : int
+load(offset : int) : int
+getOffsetValue(offsetPosition : int) : int
```

```
<<Interface>>
TargetLabel
+label : String
```

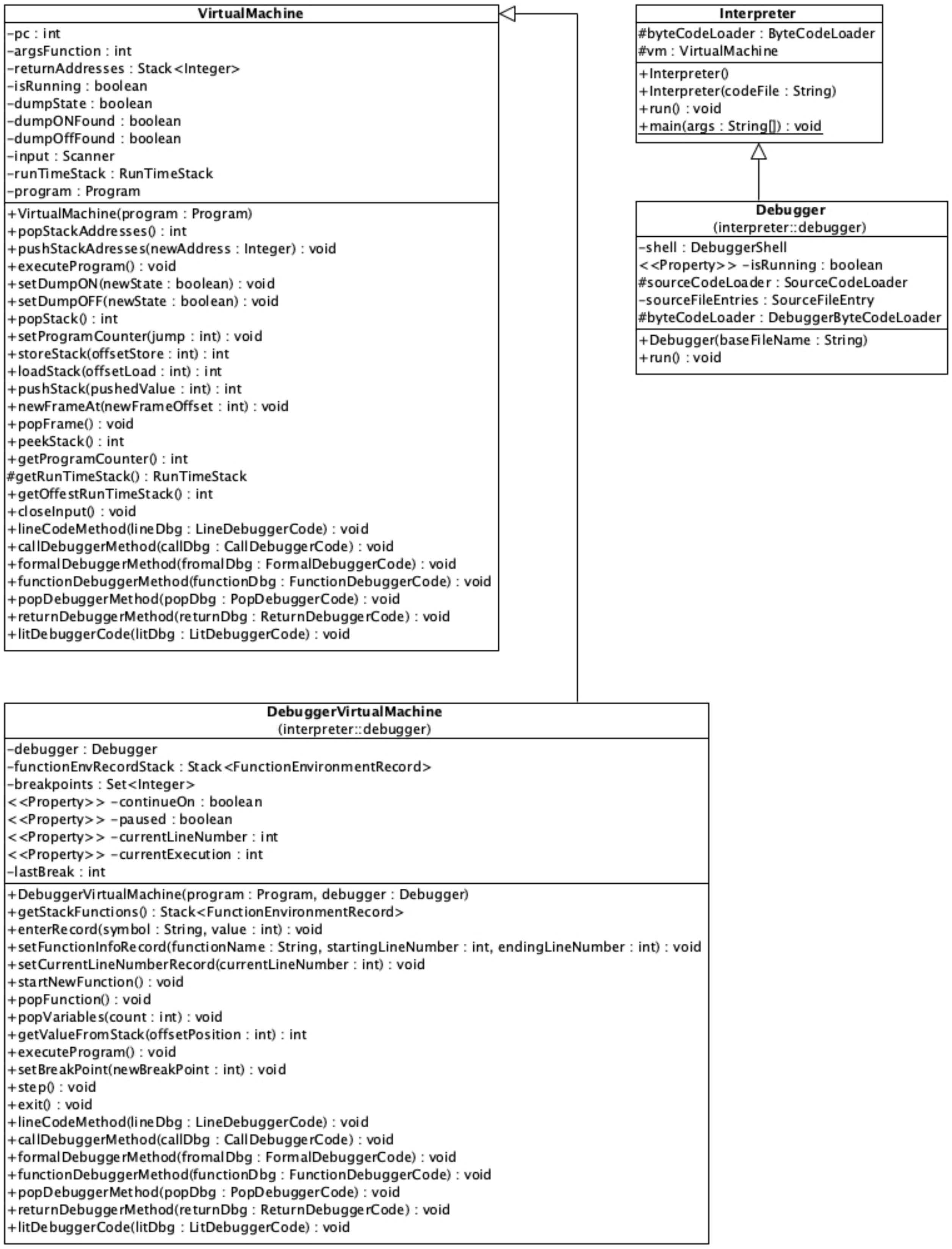
```
<<Interface>>
TargetBranch
+address : int
+label : String
```

```
CodeTable
~byteCodeMap : HashMap<String, String> = new HashMap<>()
+init() : void
+get(code : String) : String
```

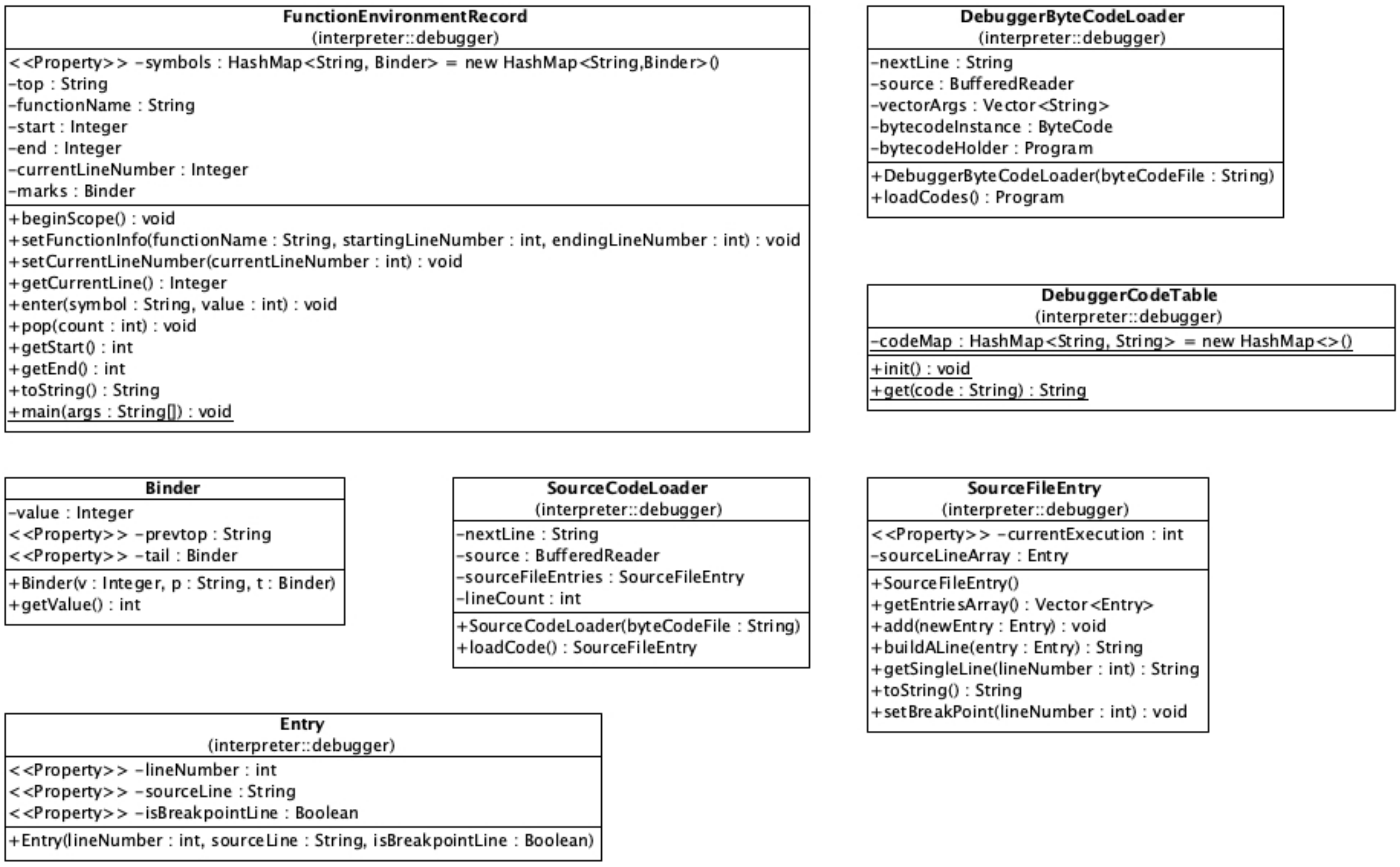
```
ByteCodeLoader
-nextLine : String
-source : BufferedReader
-vectorArgs : Vector<String>
-bytecodeInstance : ByteCode
-bytecodeHolder : Program
+ByteCodeLoader(byteCodeFile : String)
+loadCodes() : Program
```

```
Program
-arrayByteCode : ArrayList<ByteCode>
-labelMap : HashMap<String, Integer>
-targetBranchList : ArrayList<Integer>
+Program()
+getCode(programCounter : int) : ByteCode
+add(instanceByteCode : ByteCode) : void
+resolveSymbolicAdresses() : void
```

Here is the diagram for the classes contained in the Interpreter package that have a relationship of parenthood:


VirtualMachine
-pc : int
-argsFunction : int
-returnAddresses : Stack<Integer>
-isRunning : boolean
-dumpState : boolean
-dumpONFound : boolean
-dumpOffFound : boolean
-input : Scanner
-runTimeStack : RunTimeStack
-program : Program
+VirtualMachine(program : Program)
+popStackAddresses() : int
+pushStackAdresses(newAddress : Integer) : void
+executeProgram() : void
+setDumpON(newState : boolean) : void
+setDumpOFF(newState : boolean) : void
+popStack() : int
+setProgramCounter(jump : int) : void
+storeStack(offsetStore : int) : int
+loadStack(offsetLoad : int) : int
+pushStack(pushedValue : int) : int
+newFrameAt(newFrameOffset : int) : void
+popFrame() : void
+peekStack() : int
+getProgramCounter() : int
#getRunTimeStack() : RunTimeStack
+getOffestRunTimeStack() : int
+closeInput() : void
+lineCodeMethod(lineDbg : LineDebuggerCode) : void
+callDebuggerMethod(callDbg : CallDebuggerCode) : void
+formalDebuggerMethod(fromalDbg : FormalDebuggerCode) : void
+functionDebuggerMethod(functionDbg : FunctionDebuggerCode) : void
+popDebuggerMethod(popDbg : PopDebuggerCode) : void
+returnDebuggerMethod(returnDbg : ReturnDebuggerCode) : void
+litDebuggerCode(litDbg : LitDebuggerCode) : void
Interpreter
#byteCodeLoader : ByteCodeLoader
#vm : VirtualMachine
+Interpreter()
+Interpreter(codeFile : String)
+run() : void
+main(args : String[]) : void
Debugger
(interpreter::debugger)
-shell : DebuggerShell
<<Property>> -isRunning : boolean
#sourceCodeLoader : SourceCodeLoader
-sourceFileEntries : SourceFileEntry
#byteCodeLoader : DebuggerByteCodeLoader
+Debugger(baseFileName : String)
+run() : void
DebuggerVirtualMachine
(interpreter::debugger)
-debugger : Debugger
-functionEnvRecordStack : Stack<FunctionEnvironmentRecord>
-breakpoints : Set<Integer>
<<Property>> -continueOn : boolean
<<Property>> -paused : boolean
<<Property>> -currentLineNumber : int
<<Property>> -currentExecution : int
-lastBreak : int
+DebuggerVirtualMachine(program : Program, debugger : Debugger)
+getStackFunctions() : Stack<FunctionEnvironmentRecord>
+enterRecord(symbol : String, value : int) : void
+setFunctionInfoRecord(functionName : String, startingLineNumber : int, endingLineNumber : int) : void
+setCurrentLineNumberRecord(currentLineNumber : int) : void
+startNewFunction() : void
+popFunction() : void
+popVariables(count : int) : void
+getValueFromStack(offsetPosition : int) : int
+executeProgram() : void
+setBreakPoint(newBreakPoint : int) : void
+step() : void
+exit() : void
+lineCodeMethod(lineDbg : LineDebuggerCode) : void
+callDebuggerMethod(callDbg : CallDebuggerCode) : void
+formalDebuggerMethod(fromalDbg : FormalDebuggerCode) : void
+functionDebuggerMethod(functionDbg : FunctionDebuggerCode) : void
+popDebuggerMethod(popDbg : PopDebuggerCode) : void
+returnDebuggerMethod(returnDbg : ReturnDebuggerCode) : void
+litDebuggerCode(litDbg : LitDebuggerCode) : void

Here are the other classes that are contained in the Debugger package that don't have relationship of parenthood:



```
FunctionEnvironmentRecord
(interpreter::debugger)
--------------------------------------------------
<<Property>> -symbols : HashMap<String, Binder> = new HashMap<String,Binder>()
-top : String
-functionName : String
-start : Integer
-end : Integer
-currentLineNumber : Integer
-marks : Binder
--------------------------------------------------
+beginScope() : void
+setFunctionInfo(functionName : String, startingLineNumber : int, endingLineNumber : int) : void
+setCurrentLineNumber(currentLineNumber : int) : void
+getCurrentLine() : Integer
+enter(symbol : String, value : int) : void
+pop(count : int) : void
+getStart() : int
+getEnd() : int
+toString() : String
+main(args : String[]) : void
```

```
DebuggerByteCodeLoader
(interpreter::debugger)
--------------------------------------------------
-nextLine : String
-source : BufferedReader
-vectorArgs : Vector<String>
-bytecodeInstance : ByteCode
-bytecodeHolder : Program
--------------------------------------------------
+DebuggerByteCodeLoader(byteCodeFile : String)
+loadCodes() : Program
```

```
DebuggerCodeTable
(interpreter::debugger)
--------------------------------------------------
-codeMap : HashMap<String, String> = new HashMap<>()
--------------------------------------------------
+init() : void
+get(code : String) : String
```

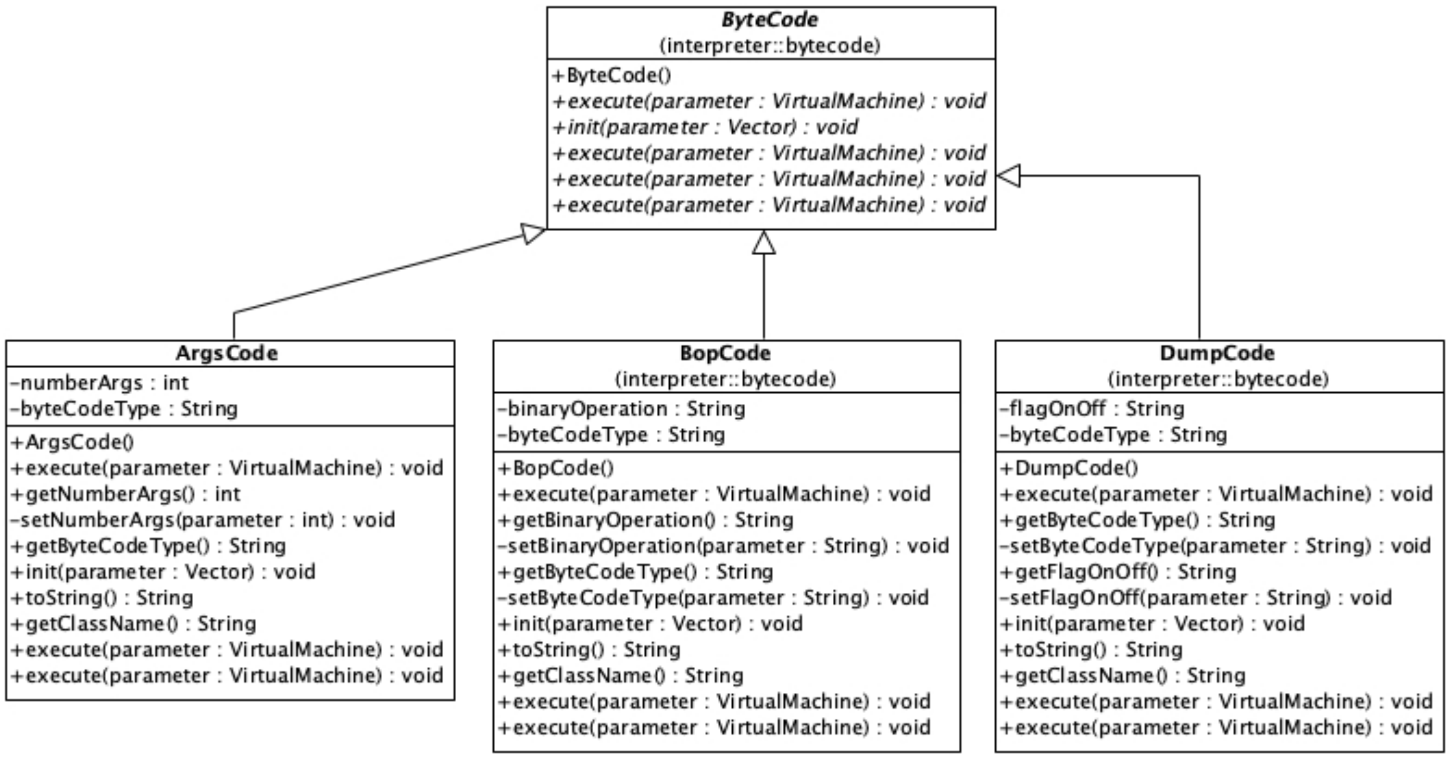
```
Binder
--------------------------------------------------
-value : Integer
<<Property>> -prevtop : String
<<Property>> -tail : Binder
--------------------------------------------------
+Binder(v : Integer, p : String, t : Binder)
+getValue() : int
```

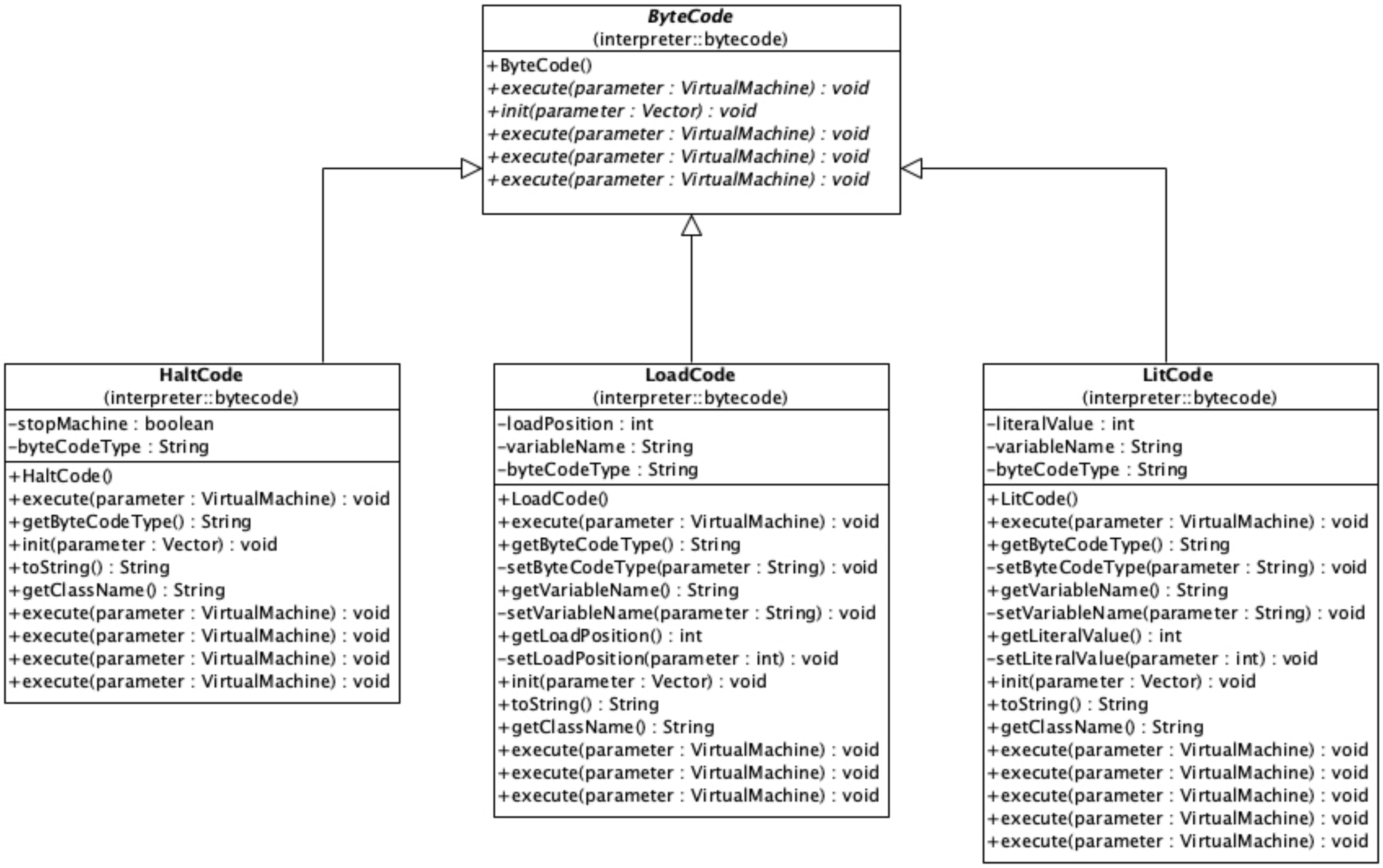
```
SourceCodeLoader
(interpreter::debugger)
--------------------------------------------------
-nextLine : String
-source : BufferedReader
-sourceFileEntries : SourceFileEntry
-lineCount : int
--------------------------------------------------
+SourceCodeLoader(byteCodeFile : String)
+loadCode() : SourceFileEntry
```

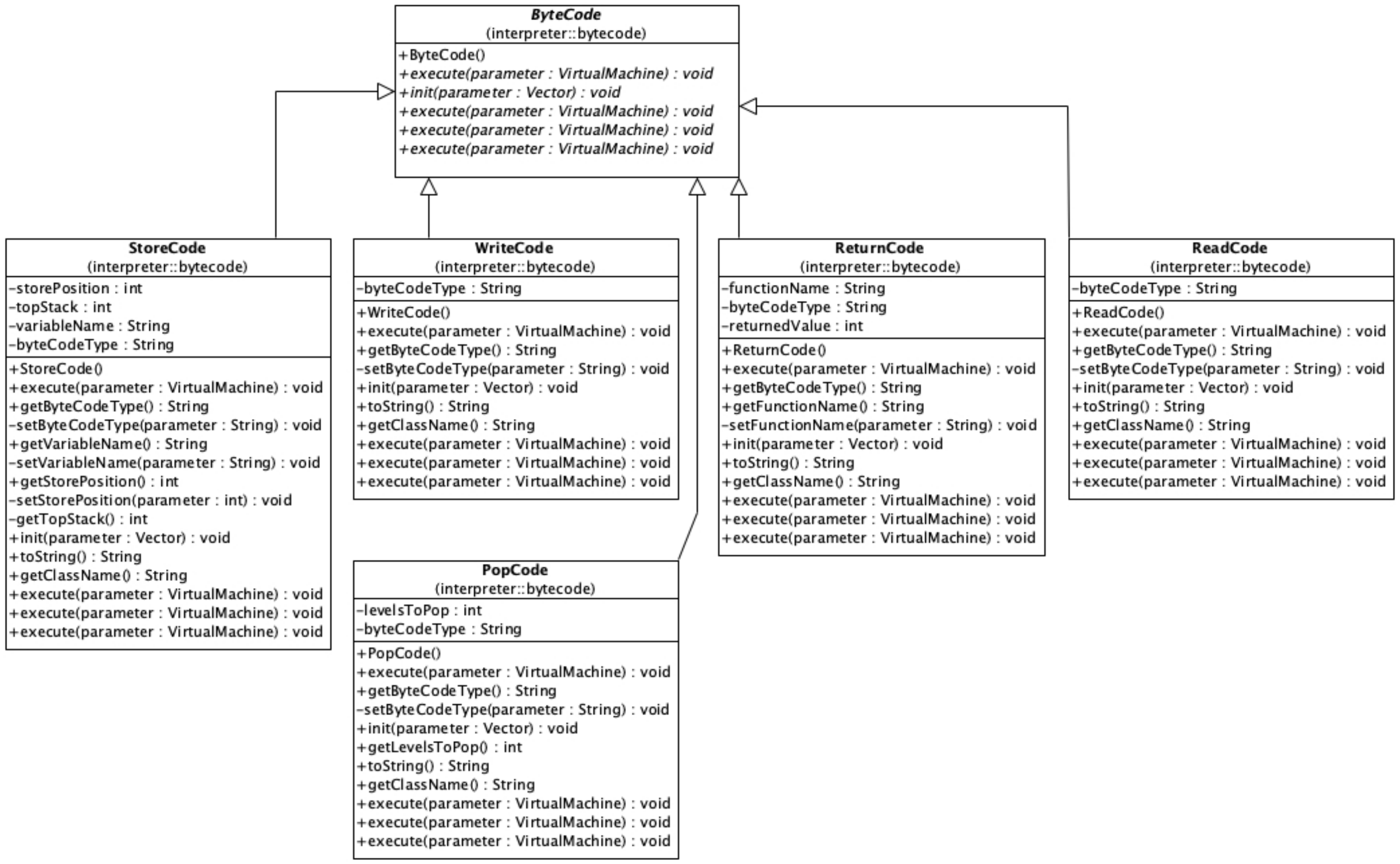
```
SourceFileEntry
(interpreter::debugger)
--------------------------------------------------
<<Property>> -currentExecution : int
-sourceLineArray : Entry
--------------------------------------------------
+SourceFileEntry()
+getEntriesArray() : Vector<Entry>
+add(newEntry : Entry) : void
+buildALine(entry : Entry) : String
+getSingleLine(lineNumber : int) : String
+toString() : String
+setBreakPoint(lineNumber : int) : void
```

```
Entry
(interpreter::debugger)
--------------------------------------------------
<<Property>> -lineNumber : int
<<Property>> -sourceLine : String
<<Property>> -isBreakpointLine : Boolean
--------------------------------------------------
+Entry(lineNumber : int, sourceLine : String, isBreakpointLine : Boolean)
```

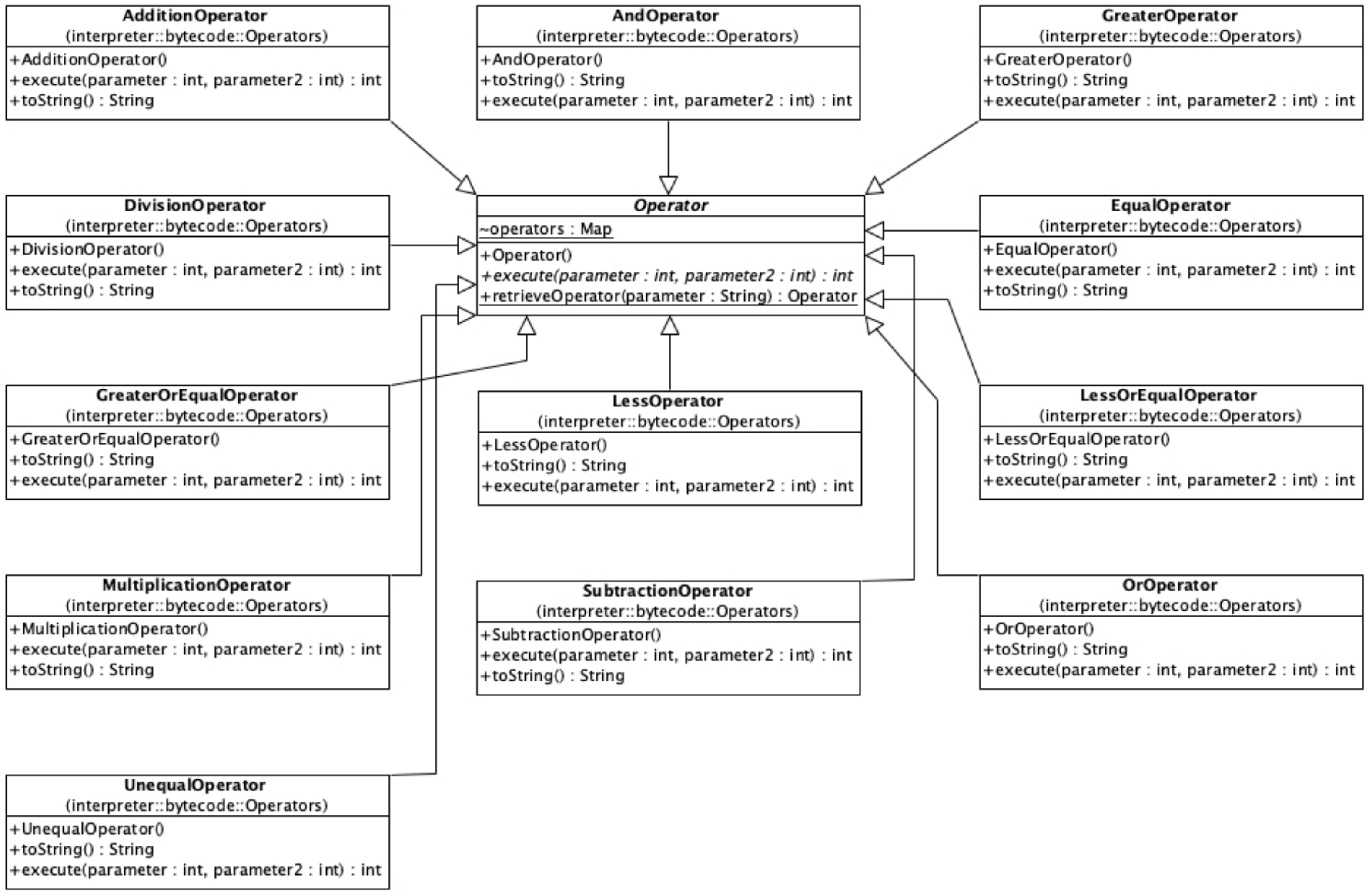
Here is a diagram for the remaining Bytecodes classes from the previous assignment:


ByteCode
(interpreter::bytecode)
+ByteCode()
+execute(parameter : VirtualMachine) : void
+init(parameter : Vector) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
ArgsCode
-numberArgs : int
-byteCodeType : String
+ArgsCode()
+execute(parameter : VirtualMachine) : void
+getNumberArgs() : int
-setNumberArgs(parameter : int) : void
+getByteCodeType() : String
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
BopCode
(interpreter::bytecode)
-binaryOperation : String
-byteCodeType : String
+BopCode()
+execute(parameter : VirtualMachine) : void
+getBinaryOperation() : String
-setBinaryOperation(parameter : String) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
DumpCode
(interpreter::bytecode)
-flagOnOff : String
-byteCodeType : String
+DumpCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getFlagOnOff() : String
-setFlagOnOff(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void

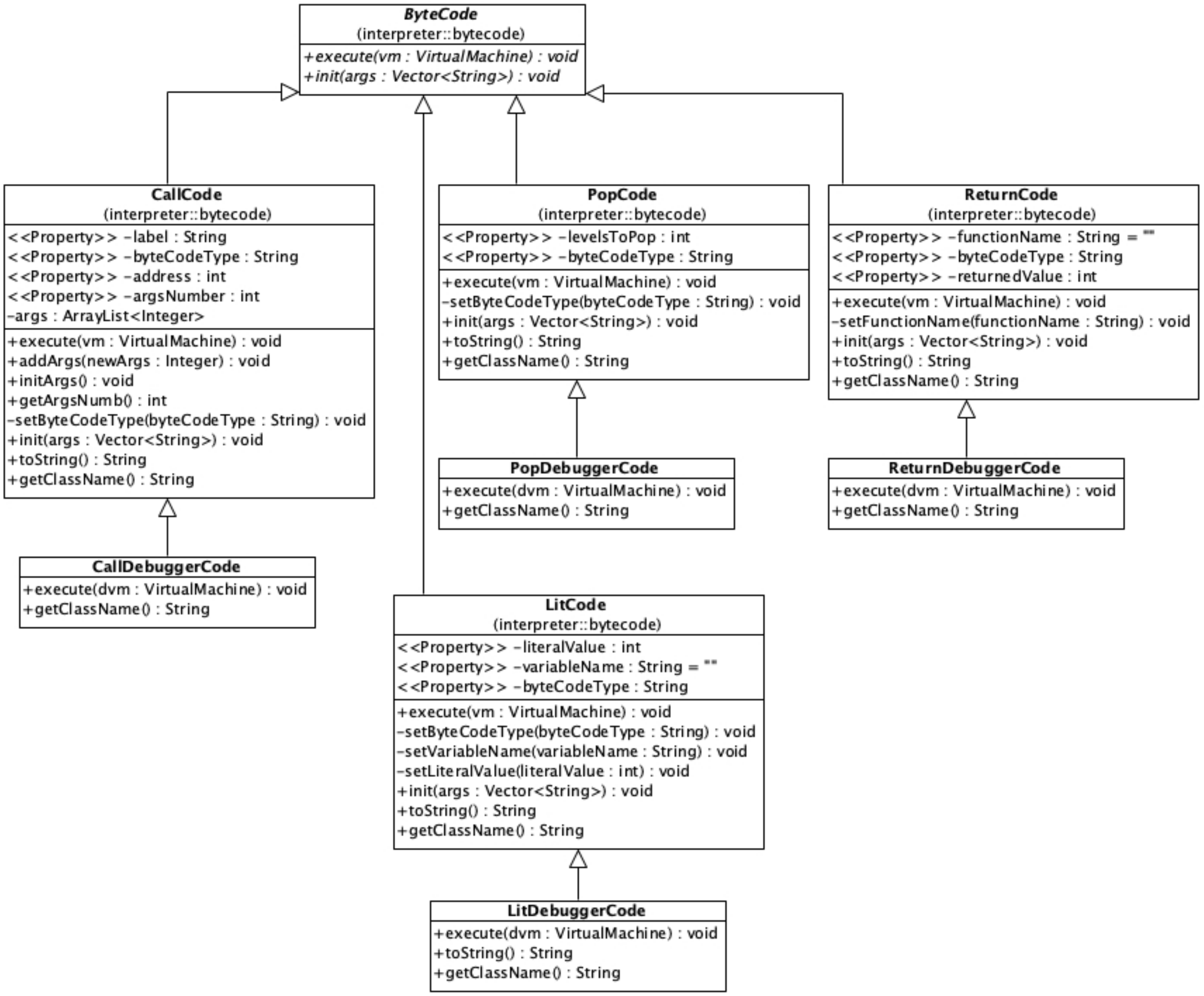

ByteCode
(interpreter::bytecode)
+ByteCode()
+execute(parameter : VirtualMachine) : void
+init(parameter : Vector) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
HaltCode
(interpreter::bytecode)
-stopMachine : boolean
-byteCodeType : String
+HaltCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
LoadCode
(interpreter::bytecode)
-loadPosition : int
-variableName : String
-byteCodeType : String
+LoadCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getVariableName() : String
-setVariableName(parameter : String) : void
+getLoadPosition() : int
-setLoadPosition(parameter : int) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
LitCode
(interpreter::bytecode)
-literalValue : int
-variableName : String
-byteCodeType : String
+LitCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getVariableName() : String
-setVariableName(parameter : String) : void
+getLiteralValue() : int
-setLiteralValue(parameter : int) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void


ByteCode
(interpreter::bytecode)
+ByteCode()
+execute(parameter : VirtualMachine) : void
+init(parameter : Vector) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
StoreCode
(interpreter::bytecode)
-storePosition : int
-topStack : int
-variableName : String
-byteCodeType : String
+StoreCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getVariableName() : String
-setVariableName(parameter : String) : void
+getStorePosition() : int
-setStorePosition(parameter : int) : void
-getTopStack() : int
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
WriteCode
(interpreter::bytecode)
-byteCodeType : String
+WriteCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
ReturnCode
(interpreter::bytecode)
-functionName : String
-byteCodeType : String
-returnedValue : int
+ReturnCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
+getFunctionName() : String
-setFunctionName(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
ReadCode
(interpreter::bytecode)
-byteCodeType : String
+ReadCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
PopCode
(interpreter::bytecode)
-levelsToPop : int
-byteCodeType : String
+PopCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+getLevelsToPop() : int
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void

Here is a diagram for Operator Classes inside Operators package:

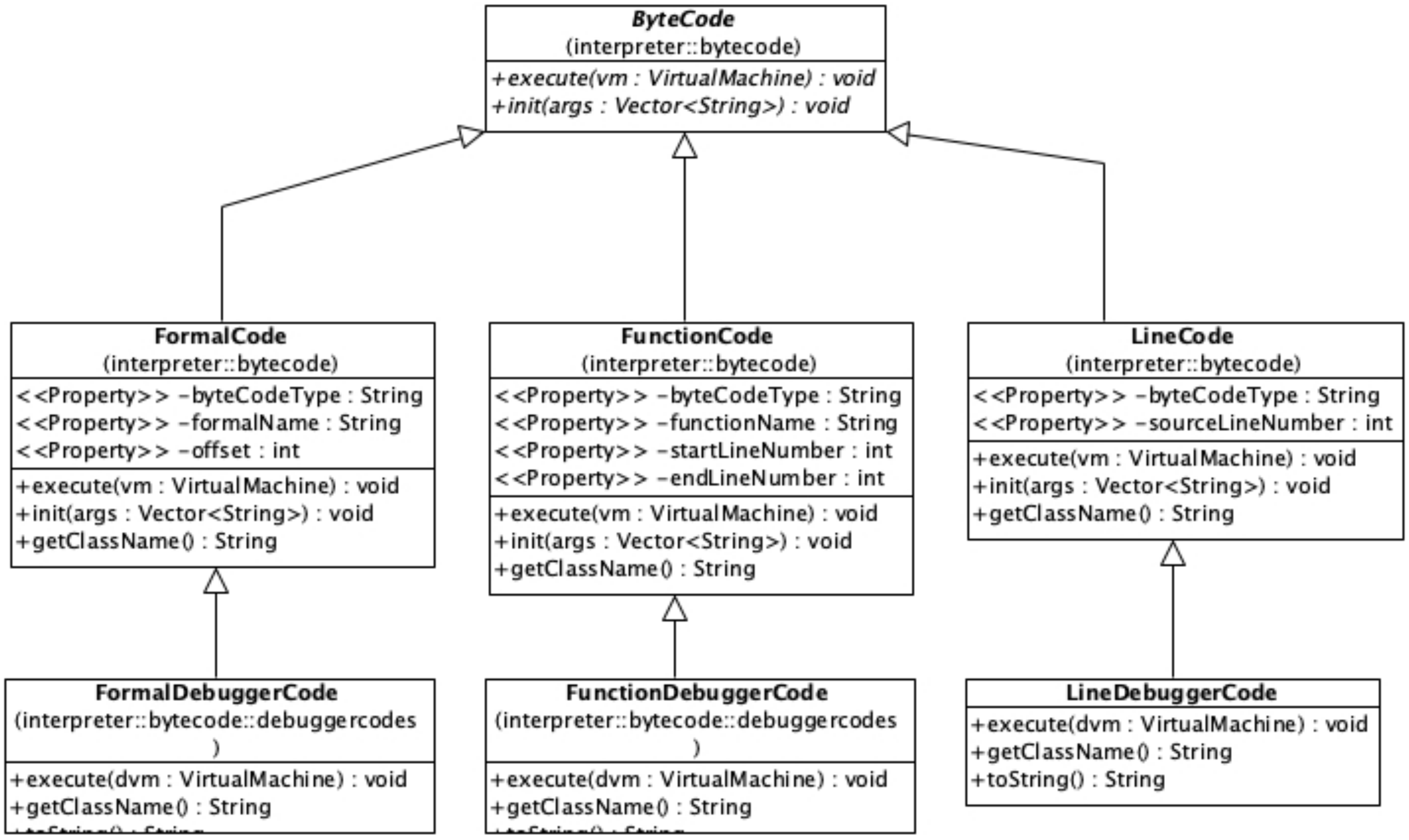

AdditionOperator
(interpreter::bytecode::Operators)
+AdditionOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
AndOperator
(interpreter::bytecode::Operators)
+AndOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
GreaterOperator
(interpreter::bytecode::Operators)
+GreaterOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
DivisionOperator
(interpreter::bytecode::Operators)
+DivisionOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
Operator
~operators : Map
+Operator()
+execute(parameter : int, parameter2 : int) : int
+retrieveOperator(parameter : String) : Operator
EqualOperator
(interpreter::bytecode::Operators)
+EqualOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
GreaterOrEqualOperator
(interpreter::bytecode::Operators)
+GreaterOrEqualOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
LessOperator
(interpreter::bytecode::Operators)
+LessOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
LessOrEqualOperator
(interpreter::bytecode::Operators)
+LessOrEqualOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
MultiplicationOperator
(interpreter::bytecode::Operators)
+MultiplicationOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
SubtractionOperator
(interpreter::bytecode::Operators)
+SubtractionOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
OrOperator
(interpreter::bytecode::Operators)
+OrOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
UnequalOperator
(interpreter::bytecode::Operators)
+UnequalOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int

Here are the modified Bytecode Classes:



ByteCode
(interpreter::bytecode)
+execute(vm : VirtualMachine) : void
+init(args : Vector<String>) : void
CallCode
(interpreter::bytecode)
<<Property>> -label : String
<<Property>> -byteCodeType : String
<<Property>> -address : int
<<Property>> -argsNumber : int
-args : ArrayList<Integer>
+execute(vm : VirtualMachine) : void
+addArgs(newArgs : Integer) : void
+initArgs() : void
+getArgsNumb() : int
-setByteCodeType(byteCodeType : String) : void
+init(args : Vector<String>) : void
+toString() : String
+getClassName() : String
CallDebuggerCode
+execute(dvm : VirtualMachine) : void
+getClassName() : String
PopCode
(interpreter::bytecode)
<<Property>> -levelsToPop : int
<<Property>> -byteCodeType : String
+execute(vm : VirtualMachine) : void
-setByteCodeType(byteCodeType : String) : void
+init(args : Vector<String>) : void
+toString() : String
+getClassName() : String
PopDebuggerCode
+execute(dvm : VirtualMachine) : void
+getClassName() : String
ReturnCode
(interpreter::bytecode)
<<Property>> -functionName : String = ""
<<Property>> -byteCodeType : String
<<Property>> -returnedValue : int
+execute(vm : VirtualMachine) : void
-setFunctionName(functionName : String) : void
+init(args : Vector<String>) : void
+toString() : String
+getClassName() : String
ReturnDebuggerCode
+execute(dvm : VirtualMachine) : void
+getClassName() : String
LitCode
(interpreter::bytecode)
<<Property>> -literalValue : int
<<Property>> -variableName : String = ""
<<Property>> -byteCodeType : String
+execute(vm : VirtualMachine) : void
-setByteCodeType(byteCodeType : String) : void
-setVariableName(variableName : String) : void
-setLiteralValue(literalValue : int) : void
+init(args : Vector<String>) : void
+toString() : String
+getClassName() : String
LitDebuggerCode
+execute(dvm : VirtualMachine) : void
+toString() : String
+getClassName() : String

Here are the new Bytecode Classes:



ByteCode
(interpreter::bytecode)
+execute(vm : VirtualMachine) : void
+init(args : Vector<String>) : void
FormalCode
(interpreter::bytecode)
<<Property>> -byteCodeType : String
<<Property>> -formalName : String
<<Property>> -offset : int
+execute(vm : VirtualMachine) : void
+init(args : Vector<String>) : void
+getClassName() : String
FunctionCode
(interpreter::bytecode)
<<Property>> -byteCodeType : String
<<Property>> -functionName : String
<<Property>> -startLineNumber : int
<<Property>> -endLineNumber : int
+execute(vm : VirtualMachine) : void
+init(args : Vector<String>) : void
+getClassName() : String
LineCode
(interpreter::bytecode)
<<Property>> -byteCodeType : String
<<Property>> -sourceLineNumber : int
+execute(vm : VirtualMachine) : void
+init(args : Vector<String>) : void
+getClassName() : String
FormalDebuggerCode
(interpreter::bytecode::debuggercodes
)
+execute(dvm : VirtualMachine) : void
+getClassName() : String
FunctionDebuggerCode
(interpreter::bytecode::debuggercodes
)
+execute(dvm : VirtualMachine) : void
+getClassName() : String
LineDebuggerCode
+execute(dvm : VirtualMachine) : void
+getClassName() : String
+toString() : String

Results and Conclusion

This project was very interesting and I can imagine a practical implication for this kind of software. It was also stimulating to build on the previous assignment, which in some way made the assignment easier to understand in its implications and developing context. On the other hand, it represented a real challenge. It was not easy to think of a tool that can run almost in parallel with the regular interpreter. We only worked with one type of file until this project and the idea of keeping track of a Bytecode file and a source code file made my head spin. I finally grasped the underlying idea and I was able to terminate the project with satisfactory results.

Challenges

This assignment presented many challenges. First of all the idea of working with two types of files simultaneously represented a challenge in itself. We only worked with one type of file at a time in the previous projects, either a bytecode file (like in the Interpreter) or a source code file

(like in the Parser). I had a hard time understanding this idea, but once I got it right, it felt much easier to design and implement the classes that I needed to build the program.
Another challenging part was represented by the FunctionRecordEnvironment. I did not understand very well the idea of the Symbol table and the Binder class used in the constrainer. I had to rewatch the lectures on the Constrainer many times and study the slides to be able to grasp the idea behind it. Also, the concept of many scopes within the same function was hard to grasp. I am used to Java and there is no such thing as different scopes within the same method.
Finally, a challenge was represented by the UI part of the assignment. I did not know how to make the classes communicate at first. Then I figured that I probably should have passed some parameters into the execute method of the Command classes. That step changed completely my perspective and I was finally able to access all the fields and methods that I needed to correctly display and manipulate debugging information.

Future Work

I would like to implement more features for the Debugger. For example, we have a Set command for breakpoints but we don’t have a Remove command to be able to remove them. I am also fascinated by the idea of having a feature that allows you to go back one step. I know generally, a debugger doesn’t have this feature because it is probably too hard (impossible?) to implement, but I would like to better understand what are the obstacles to its implementation and try to imagine a way to implement it.

Summary of Technical Work

Category	Description		Notes
Code Quality		✓ Code is clean, well formatted (appropriate white space and indentation)	
		✓ Classes, methods, and variables are meaningfully named (no comments exist to explain functionality - the identifiers serve that purpose)	
		✓ Methods are small and serve a single purpose	
		✓ Code is well organized into a meaningful file structure	
Documentation		✓ A PDF is submitted that contains:	
		✓ Full name/Student ID	
		✓ A link to the github repository	
	Overview	✓ Project introduction	
	Overview	✓ Summary of technical work	
	Overview	✓ Execution and development environment described	
	Overview	✓ Scope of work described (including what work was completed)	
		✓ Command line instructions to compile and execute	
		✓ Assumptions made	
	Implementation discussion	✓ Class diagram with hierarchy	
	Implementation discussion	✓ Implementation decisions	
	Implementation discussion	✓ Code organization	
		✓ Results/Conclusions	
		✓ Formatting	
Requirement 1		✓	
Requirement 2		✓	
Requirement 3		✓	