CSC 413 Project Documentation

Fall 2018

Grant Kennedy

913056513

413.02

https://github.com/csc413-02-fa18/csc413-p2-grantwkenn.git

Table of Contents

1 Introduction 3

1.1 Project Overview 3

1.2 Technical Overview 3

1.3 Summary of Work Completed 3

2 Development Environment 3

3 How to Build/Import this Project 3

4 How to Run this Project 4

5 Assumption Made 4

6 Implementation Discussion 5

6.1 Class Diagram 5

7 Project Reflection 5

8 Project Conclusion/Results 6

1 Introduction

1.1 Project Overview

This project implements an interpreter used to run a program loaded from a file containing “X” language compiled bytecodes.

1.2 Technical Overview

The program takes a file containing “X” language bytecodes as its argument. First, the ByteCodeLoader class parses from this file the corresponding bytecodes and stores them in the Array List program. Then the bytecode loader calls the program’s resolveAddress() function to inspect each bytecode and resolve the address to which the byte code is referring if any exists. From there, the Interpreter creates a Virtual Machine to move through the program and execute each bytecode in order. The Virtual Machine manages data structures including an object of the runTimeStack class: runStack to maintain the state of the program. Each bytecode operates by requesting the Virtual Machine to make changes to its structures and alter the state of the program.

1.3 Summary of Work Completed

Implemented the ByteCodeLoader, Program, RunTimeStack, and VirtualMachine classes in the interpreter package to work alongside the provided Interpreter and CodeTable classes in order to run a program from a given file.

Implemented each ByteCode class to carry out the necessary operations on the program.

Added additional abstract classes AddressCode and IntArgumentCode to improve the bytecode inheritance hierarchy.

2 Development Environment

This project was developed with Java version: 1.8.0_161 in the Eclipse IDE for Java Developers with Build Version: Photon Release (4.8.0). Eclipse was installed on a PC with Windows 10.

3 How to Build/Import this Project

Download this program using the following GitHub link:

https://github.com/csc413-02-fa18/csc413-p2-grantwkenn.git

The project was imported into Eclipse from the file system by importing the master folder titled “csc413-p2-grantwkenn”. The project was built automatically throughout development in the Eclipse IDE. With “Build Automatically” disabled, the project can be built with “Ctrl+B” or from the “Project” dropdown menu.

NOTE: The Eclipse IDE used required that the contents of the interpreter package be in their own folder (csc413-p2-grantwkenn/interpreter/interpreter) alongside the bytecode package folder. Other IDEs may identify the default package folder (csc413-p2-grantwkenn/interpreter) as the interpreter package. This change used during development is not reflecting in this finished product, to maintain the intended format of file system. In other words, the items in the interpreter package

are found in csc413-p2-grantwkenn/interpreter, but some IDEs may require them to be contained in their own folder: csc413-p2-grantwkenn/interpreter/interpreter.

4 How to Run this Project

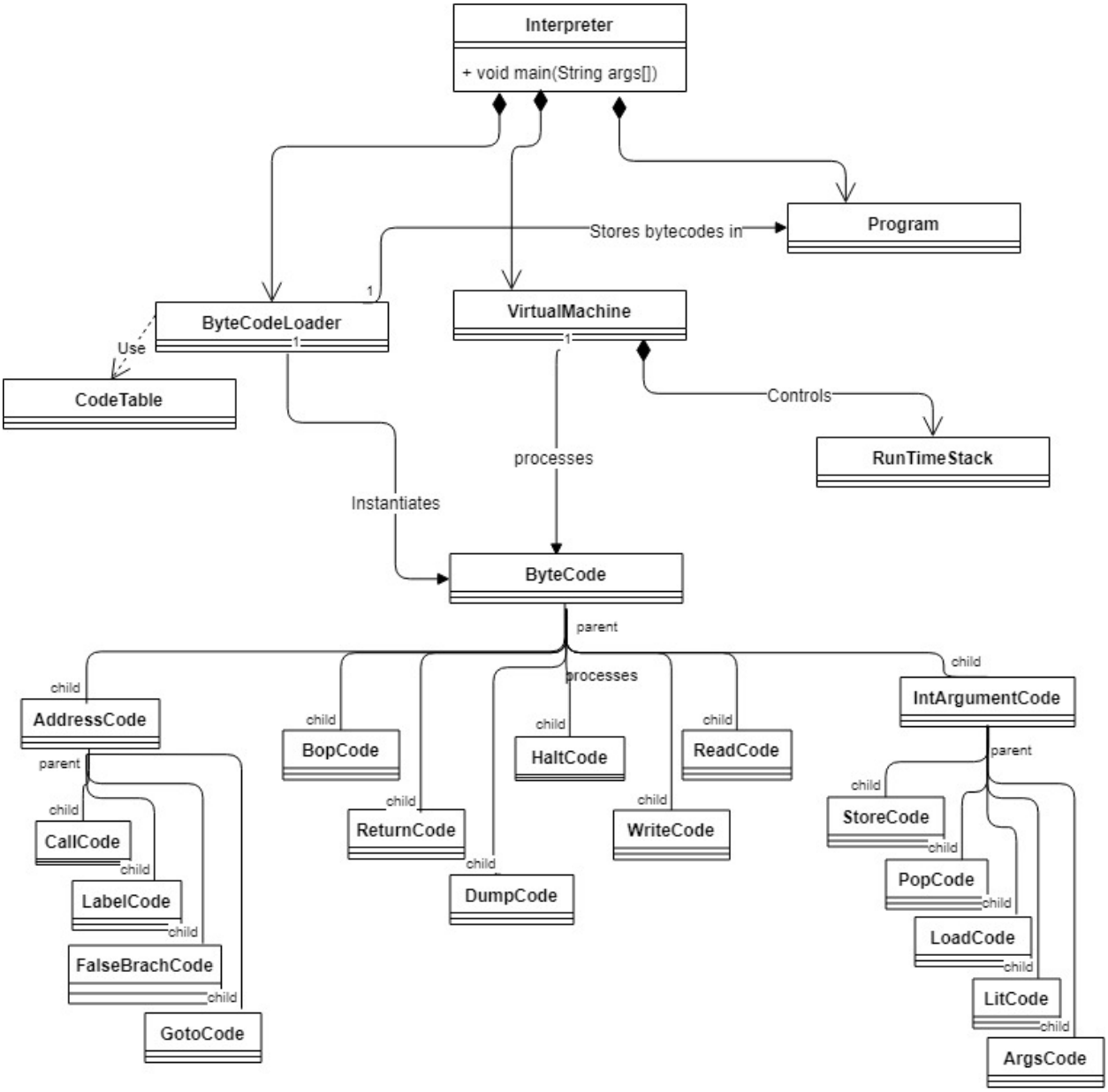
Project was run from within the Eclipse IDE by pressing Ctrl+F11 or from the “Run” dropdown menu. Program requires a “.X.cod” file as an argument, this was set in the Run Configurations in the IDE. Sample “.X.cod” files are located in the main folder: csc413-p2-grantwkenn

5 Assumption Made

This program assumes that the bytecodes represent the correct compilation of “X” language source code. However, the RunTimeStack class protects the program state from operations which would cause Stack related exceptions. In other words, the program is designed not to crash even if the source code contains errors.

6 Implementation Discussion

6.1 Class Diagram



7 Project Reflection

This project was implemented with the expected results. One of the most expensive and complex tasks was the dump() function within the RunTimeStack. This function is meant to simply print the contents of the stack, but it requires altering several of the Run Time Stack data structures, and then putting them back together to print the program state from left to right, between every single execution of a bytecode. Some consideration was made to the best implementation for the bytecode inheritance hierarchy. There may be a more efficient and cleaner looking way for each bytecode to inherit from ByteCode to inherit the methods and variables they each need. But this

implementation worked well by separating the codes which would need an address resolved, and having them inherit that variable from the abstract class AddressCode. Also, the IntArgumentCode allowed the separation of bytecodes which did not need to store and int argument, so that they did not have to wastefully inherit that variable.

8 Project Conclusion/Results

This program successfully implemented an “X” code interpreter. The included test files “factorial.x.cod” and “fib.x.cod” can be run to obtain their expected results. The program is stable for very large inputs such as 12 factorial.