CSC 413 Project Documentation

Fall 2018

Brian Lai

916818167

CSC413.03

<https://github.com/csc415-03-spring2019/csc413-p2-blai30>

Table of Contents

[1 Introduction 3](#_Toc522827688)

[1.1 Project Overview 3](#_Toc522827689)

[1.2 Technical Overview 3](#_Toc522827690)

[1.3 Summary of Work Completed 3](#_Toc522827691)

[2 Development Environment 3](#_Toc522827692)

[3 How to Build/Import your Project 3](#_Toc522827693)

[4 How to Run your Project 3](#_Toc522827694)

[5 Assumption Made 3](#_Toc522827695)

[6 Implementation Discussion 3](#_Toc522827696)

[6.1 Class Diagram 3](#_Toc522827697)

[7 Project Reflection 3](#_Toc522827698)

[8 Project Conclusion/Results 3](#_Toc522827699)

# Introduction

## Project Overview

This program reads a file and performs specific functions based on the text that is on each line in the file. The user is asked to type in an integer value at some point running the program and the program will read that value into a stack. The program manages a stack of values that are separated by functions and works its way through each of them in order to calculate a result.

## Technical Overview

This program will be able to read a source code file that was written in the mock language X. The file will contain byte codes and arguments for various byte codes. The program interprets the file and asks for a nonnegative integer input from the user then calculates the result based on the input.

In order for the program to work, each byte code represents a specific function, if it has one, and the program will parse each byte code to identify how they will function. The program manages a stack and frames to store values that are generated by the various byte codes. These values are then further manipulated to calculate the result.

## Summary of Work Completed

Created an abstract class called ByteCode and created fifteen subclasses that extends ByteCode. Created an interface class called AddressLabel which will be implemented by a select few of the ByteCode subclasses. Implemented the ByteCodeLoader class to parse the program argument file into ByteCode objects and their arguments, if they have any, that correspond to their class name. These objects are stored into an ArrayList that is passed to a Program object. Implemented the Program class to resolve the addresses of ByteCode objects that implement the AddressLabel interface. Doing this will tell these objects what labels to point to when the ByteCode in the program executes. Implemented the RunTimeStack class with various methods, similar to those of Java’s Stack class, that will assist in managing the runtime stack. These methods will be used by the VirtualMachine class which will be used by the ByteCode classes. Implemented the VirtualMachine class with various methods that the ByteCodes will use in order to manage the runtime stack. Most of these methods serve as wrappers for the RunTimeStack class methods in order to preserve encapsulation. The ByteCode classes are not allowed to call methods from the RunTimeStack class directly.

Each ByteCode class except for Dump is capable of dumping itself to the console if the dump flag in the VirtualMachine object is set to true. If the program is being dumped, the runtime stack and its frames will also be dumped.

# Development Environment

Java Version: JDK 11.0.2

IDE: IntelliJ IDEA Ultimate 2018.3.3

# How to Build/Import your Project

1. Open IntelliJ IDEA
2. Click import project
3. Navigate to the “csc413-p2-blai30” directory and click OK
4. Make sure “Create project from existing sources” is selected and click NEXT
5. Proceed to click NEXT until import is finished
6. Build the project using IntelliJ IDEA IDE by navigating to Build > Build Project from the menu bar

# How to Run your Project

1. While the project is open in IntelliJ, navigate to Run > Edit Configurations… from the menu bar
2. Click the “+” button in the top left corner of the Run/Debug Configurations window and select Application
3. In the “Name:” text field, type in “factorial”
4. In the “Main class:” text field, type in “interpreter.Interpreter”
5. In the “Program arguments:” text field, type in “factorial.x.cod”
6. Repeat steps 3-5 with “fib” for the name and “fib.x.cod” for the program arguments
7. Click APPLY then OK
8. In the top right corner of the IntelliJ IDEA window, click the dropdown menu and select either “fib” or “factorial”
9. Click the green play button next to the dropdown menu (or press Shift+F10) to run the program with the selected arguments

# Assumption Made

# Implementation Discussion

## Class Diagram

# Project Reflection

# Project Conclusion/Results