CSC 413 Project Documentation

Fall 2019

Moataz Affan

916231373

413.3

https://github.com/Moataz-A

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

The interpreter is a program to read and convert a made up language called X and its bytecode through a virtual machine that can interpret to the language Java. It’s basically given cake ingredients and it will make the cake itself. To test our interpreter, we have sample codes with the extension x.cod to be fed into the program and see if it can calculate recursive functions such as Fibonacci sequence and factorial for n where n is any integer. Object oriented programming is critical for our project where we need to use every element of it.

## Technical Overview

OOP principles are used to build, test and launch this project that interprets X language which is a simplified version of Java. We have classes for basic assembly-like operations, in those classes we take bytcodes and split them into commands such as (jump to, pop, call, store, halt etc.) every bytecode needs functions for loading, initializing, and executing. Here we must encapsulate abstract classes under the ByteCode class and give them limited accessibility. The subclasses are considered to be lower in level than others in the interpreter. We are also making a Virtual Machine class from scratch that executes the bytecode then iterate through all the lines. Also, the Program class takes all the code received from the file into an ArrayList and the ByteCodeLoader class loads the byte code into the object from that class. Given classes are Interpreter and CodeTable which are already implemented and Interpreter has the main method that takes a file as an argument to first start the program and run the stack as a whole.

## Summary of Work Completed

Started by making the ByteCodeLoader followed by subclass ByteCode and then the other subclasses while going back and forth between Program and RunTimeStack. VirtualMachine was last and it took most of my time because my approach was to make a huge while (isRunning) block that has tens of if statements to invoke the write methods and code. It should run without any exceptions unless there’s a test file that I haven’t received since my debugging is based on 3 test files only.

# Development Environment

Intellij Community 2018.3.2

JDK 11.0.2

# How to Build/Import your Project

IMPORT:

1. Open IDE
2. Import project
3. Locate directory (interpreter)
4. “Create project from existing sources”
5. Next until done
6. Inside IDE find Build then Build Project

BUILD:

1. File
2. Project Structure
3. Add jar files
4. Module menu choose main
5. “Extract to target JAR”
6. Build Artifacts

# How to Run your Project

Double shift in the IDE to open up the search box

Search for edit configuration and click it

Go to program arguments and put the file name that you want (fib.x.cod)

Click on run while in interpreter class and run .main()

You will be asked to enter an integer and then press enter

The result for that Fibonacci sequence number would popup (warning: numbers above 30 would take a long time)

# Assumption Made

This is a difficult project to make assumptions for since we are diving into bytecode and can’t guarantee the client’s input, or the file being read. I have thrown exceptions here and there but also the program will keep prompting the user for an integer until it’s finally entered. If we are working with other files to be read by the program then I’m assuming that there will be a few exceptions since I can’t ensure that the bytecode statements are correct.

# Implementation Discussion

OOP concepts are clear in this program. From inheritance to composition and encapsulation not to forget abstraction. All were necessary for this project to work efficiently.

## Class Diagram

A screenshot of a video game

Description automatically generated

# Project Reflection

This is definitely the most difficult project I’ve ever worked on even though the tank game said to be harder/equal but at least you can visualize and see what’s going on with every step and I bet it’s fun to debug. While this is really vague and abstract, I couldn’t wrap my head around a lot of concepts and it took me countless hours. Short time to research some concepts and catch up on Java even with the extension period due to how massive and complicated this is. Reading the PDF itself was a project by itself and if I’m going to be honest I definitely did not enjoy this project it was torture.

# Project Conclusion/Results

From the mockup X language we take its bytecode and interpret that into Java. Looking at my code after finishing the project it was ugly and still is, I didn’t want to optimize or enhance readability due to time constraints and the fear of destroying the whole thing and starting from scratch. As long as OOP concepts were followed, I’m happy with the result since all tests were positive and ran efficiently I even left the computer over night to see if the program would crash with an input of 1000 for the Fibonacci sequence and it was still running when I woke up which to me is a positive sign that it’s functional.