CSC 413 Project Documentation

Fall 2019

Maria Seljak

915736307

Section 01

https://github.com/csc413-01-fall2019/csc413-p2-lseljak92

Table of Contents

1 Introduction

1.1 Project Overview

This project implements an interpreter for the mock `language X`. The main goal of the interpreter consists on processing byte codes that are created from source code files that have the extension x. This project also implements a Virtual Machine, which works together with the interpreter to run a program written in the Language X.

1.2 Technical Overview

When approaching this project, I first focused on the abstract expressions. Before developing any of the byte codes, it was crucial to declare an abstract class that would define all common operations to the specific codes. Each one of the byte codes extends the abstract `ByteCode` class, which defines the two main abstract methods: `execute(VirtualMachine vm)` and `init(ArrayList<String> args)`. Then, I also supplied new methods and data fields depending on the specific byte code.

Encapsulation was also a key concept when working on The Interpreter. It doesn't only allow the program's reuse, but it also increases its reliability. In the project's implementation, each byte code interacts with the `runtime stack` data only through the virtual machine's methods. They cannot directly access instance fields from any of the other program's classes other than their own.

The `ByteCodeLoader` extends the `Object` class. Its main functionality consists on reading one line of source code at a time. For doing so, I used the `StringTokenizer` class from the `java.util` library. This class allows the program to break a string into tokens. This makes each token accessible and easier to compare to another specific string. In order to get the source code, I utilize the `readline()` method. In order to avoid exceptions, I applied `try` and `catch` methods.

The `Virtual Machine` is used to execute the program. In other words, it is the main controller of the project: all operations need to go through this class, otherwise they can't be executed. In this class, the runtime stack is defined, as well as the program, the program counter, the Stack that stores all the byte codes and a Boolean flag that sets if the program is running or not. Its functionality also reinforces the idea of encapsulation.

1.3 Summary of Work Completed

I was able to complete the `ByteCode` abstract class as well as the implementation of each distinct byte code. In the `ByteCodeLoader` class, I completed the `loadCodes()` function implementation. The `RunTimeStack`, `VirtualMachine` and `CodeTable` classes are also fully implemented, except for the dump function in the `RunTimeStack`. Due to not managing my time accordingly, I was not able to implement the `resolveAddrs()` function in the Program class, which makes the project not work appropriately. Even though I wasn't able to complete the overall requirements by the deadline, I'm planning on fixing the program's implementation in order to have a functional project

2 Development Environment

This project has been developed using the IntelliJ IDE on JAVA version 8 update 221.

3 How to Build/Import The Interpreter

In order to build/import the Interpreter project, first clone or download the repository to your local machine. For guidance on cloning, this link provides a thorough explanation of the steps. When opening IntelliJ or your preferred IDE select “import project” and choose the root of the repository as the source. It is important not to use the interpreter folder as the root since this will cause the project not to load properly.

4 How to Run the Interpreter

For running the Interpreter search for the “Run program” icon on the top menu of your IDE. If using IntelliJ, next to that icon there’s a dropdown menu where any file can be chosen to be executed. Select the Interpreter file and then select “Edit Configurations”. A new widow will pop up where it can be established the program arguments. You can either choose `factorial.x.cod` or `fib.x.cod`. However, since the project is not fully complete this will not execute properly.

5 Assumption Made

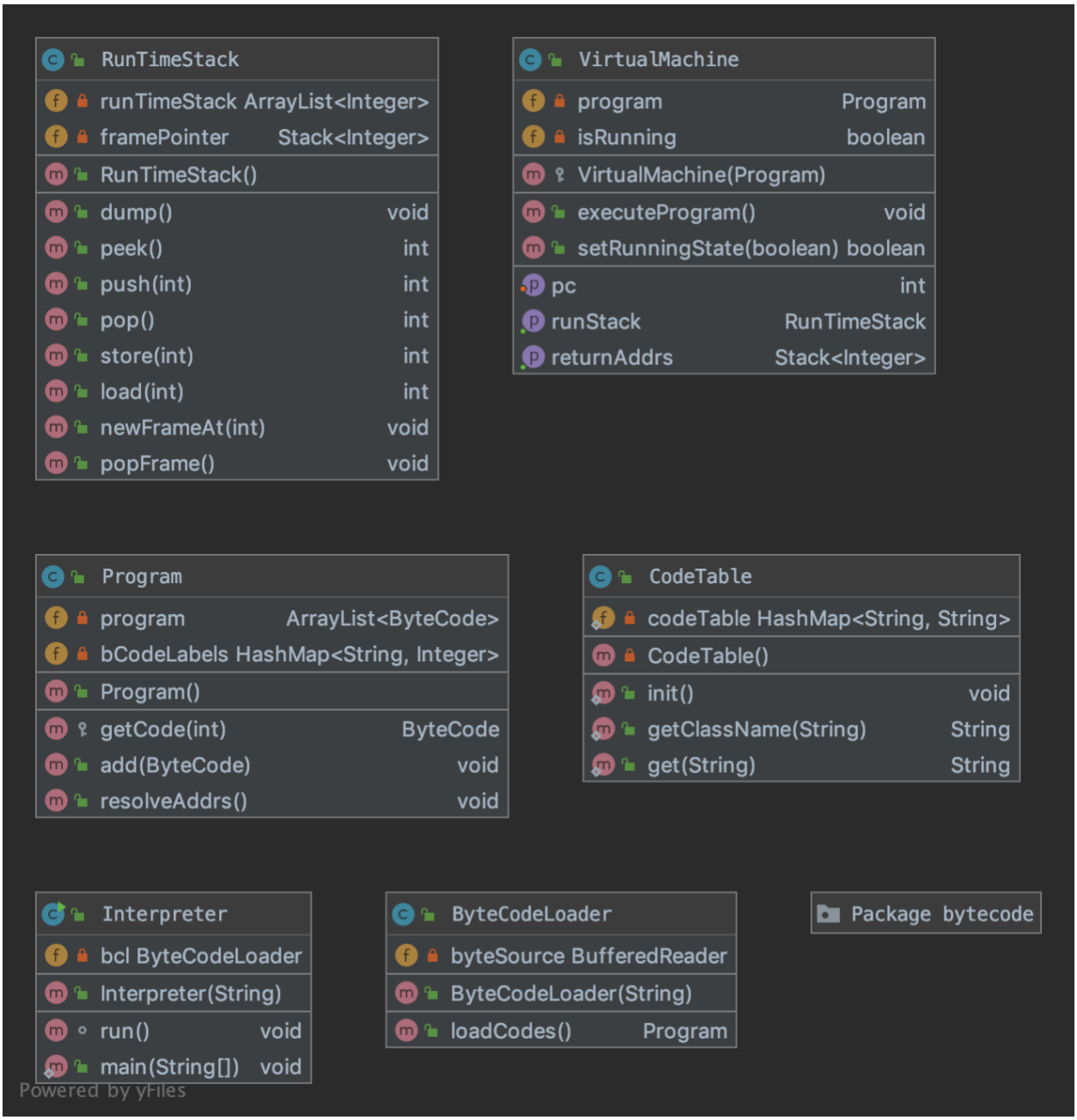
My first (wrong) assumption was that I had sufficient time to implement the assignment. I did not manage my time accordingly which led to not being able to finish the requirements. This undoubtedly serves as a lesson to start managing my time better and planning ahead of time on the implementations of the program.

This assignment was a challenging project which let me “think outside the box”. It let me understand better the importance of encapsulation and abstraction in Object Oriented programming, and it also let me get creative when thinking of the different methods and instance variables. While working on the project, I found the book “Core Java Fundamentals” very useful when looking into possible solutions to solve specific problems.

6 Implementation Discussion

Many of my decisions were based on the book “Design Patterns, Elements of Reusable Object-Oriented Software”. One of the patterns described in this book actually refers to an Interpreter. The applicability of this design consists on having a language to interpret whose statements/labels can be represented as “abstract syntax trees”. This was applied in the bytecode package. There is one main byte code class that sets the blueprint for the classes that extend it. It defines the methods common to all the nodes in that tree. The use of abstraction makes it possible to save space and possibly time when running the program. Some of the benefits of approaching the project using abstraction and encapsulation is that it makes the program easy to change, maintain and reuse.

6.1 Class Diagram



7 Project Reflection

This project was a great way to learn and implement the concepts of encapsulation and abstraction. In order to improve it, I'm planning to complete the missing functions in the program in order to experience the actual expected functionality.

8 Project Conclusion/Results

Even though I was not able to fully complete the assignment, I believe that I learned a lot along the way. One of the main mistakes I made was not managing my time appropriately. However, even though I was not able to meet the deadline with all the requirements, I'm planning to complete the missing methods and classes to fully complete the challenge.