CSC 413 Interpreter Documentation

Fall 2024

Dale Rivera

922792075

Section 1

[GitHub Repository Link]

Table of Contents

1 Introduction

1.1 Project Overview

This project is an interpreter for the mock language *X,* a simplified version of Java.

1.2 Technical Overview

This interpreter processes bytecode instructions, maintaining a runtime stack for function calls, variable storage, and intermediate computations. It interprets and executes instructions in the .cod files step by step.

1.3 Summary of Work Completed

I was able to complete the assignment after re-reading the instructions and re-watching the recorded lectures many times. I was confused as to what the virtual machine is supposed to do but after the explanation: "the virtual machine acts as a bouncer between the byte codes and the runtime stack," I understood what I was supposed to do to complete the project. Most errors I got when I ran the .cod files initially were empty stack exceptions but I was able to figure out that it had something to do with the frame pointer.

2 Development Environment

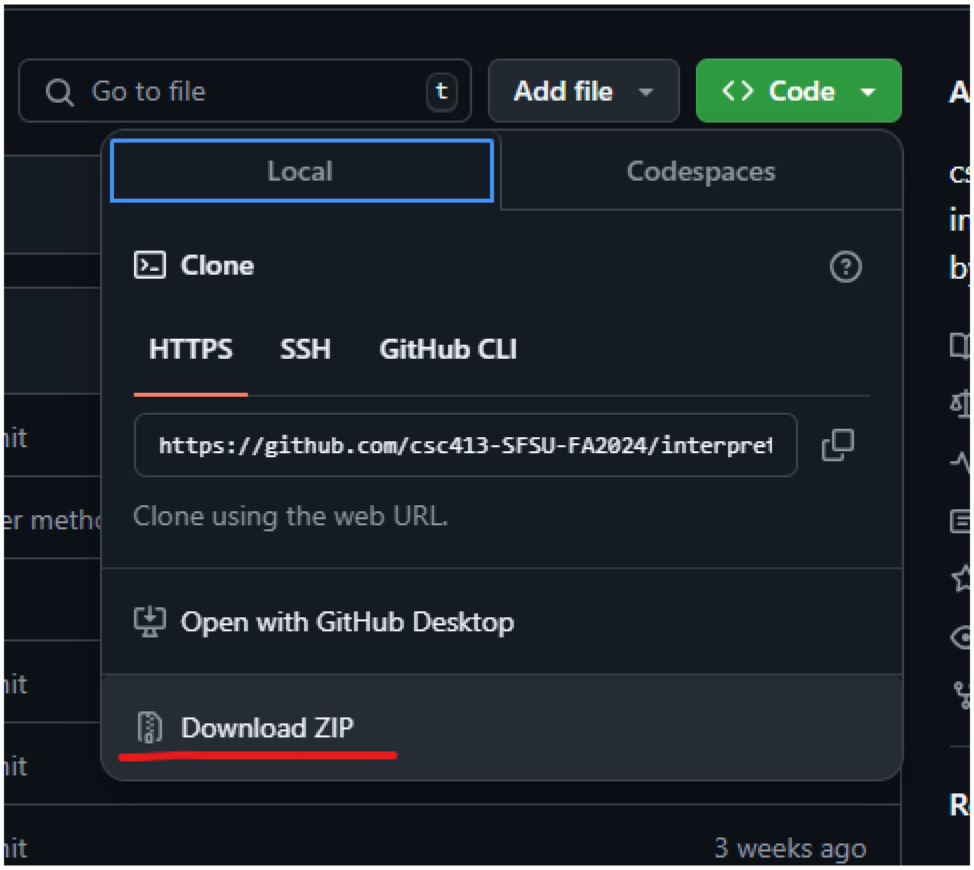
Java version: "22.0.1" 2024-04-16

IDE: IntelliJ IDEA 2024.2.0.2 (Ultimate Edition)

3 How to Build/Import your Project

This instruction is tested for Windows only.

1. Download the zip file from the repository by clicking the green “code” button.

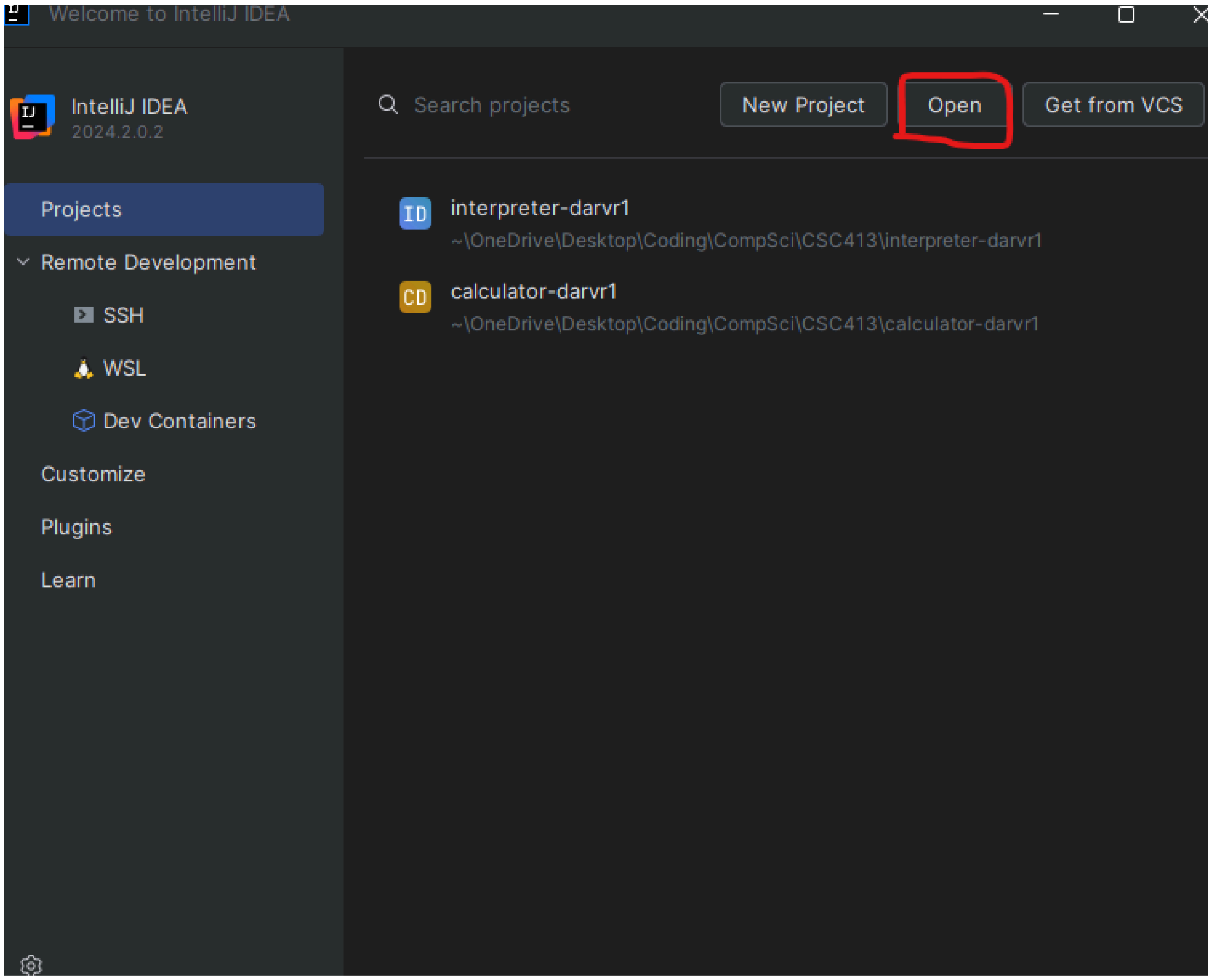


```
https://github.com/csc413-SFSU-FA2024/interpret
```

Clone using the web URL.

2. Extract the zip file however you prefer.
3. Open IntelliJ.

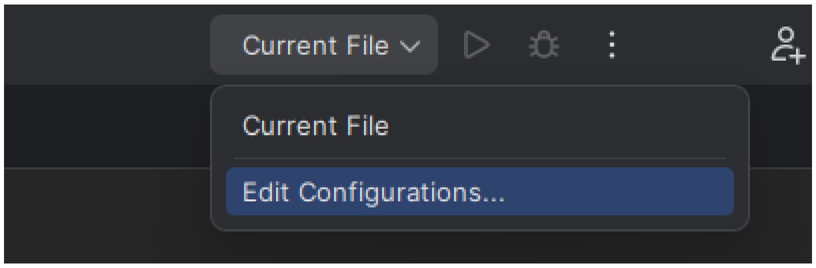
4. Open the (unzipped) folder through intelliJ by hitting the “open” button then find and select the project folder.



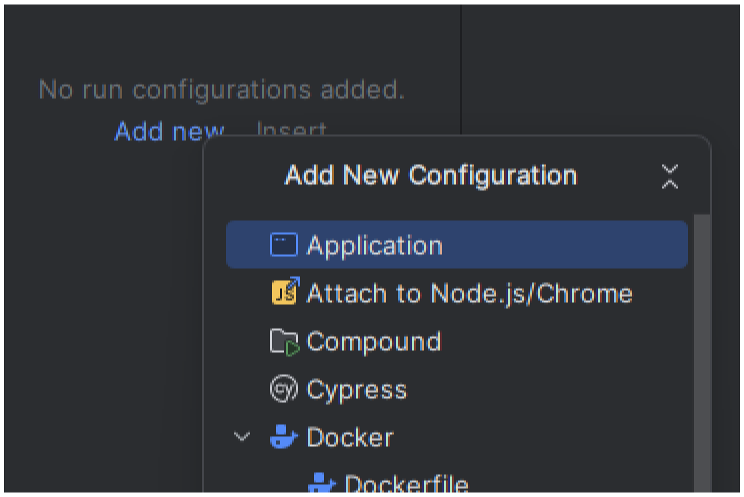
interpreter-darvr1
~\OneDrive\Desktop\Coding\CompSci\CSC413\interpreter-darvr1
calculator-darvr1
~\OneDrive\Desktop\Coding\CompSci\CSC413\calculator-darvr1

4 How to Run your Project

1. After you successfully opened the project in IntelliJ. Open the “Current File” dropdown at the top right then click “edit configuration.”



2. Add new “application”



3. After creating a new configuration, make sure that the text in the red underline **IS EXACTLY**: interpreter.Interpreter

   The text in the green underline is the **EXACT** name of the *.cod* file you want to run. The supported files are:

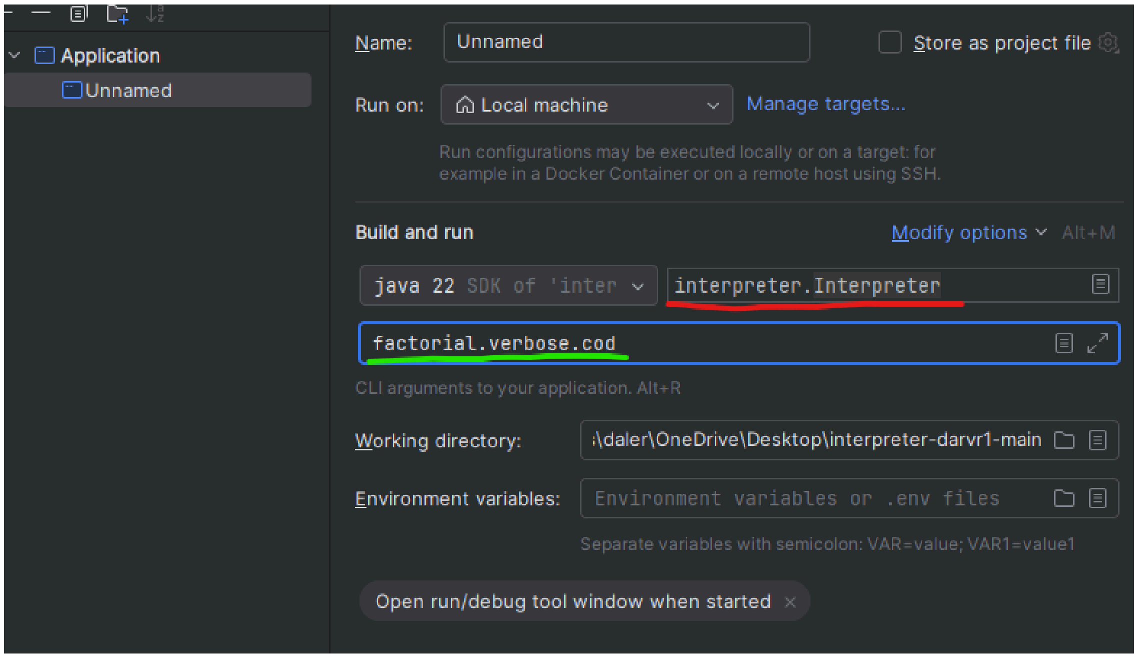
   factorial.verbose.cod

   factorial.x.cod

   fib.x.cod

   functionArgsTest.cod

   In this image, it will run factorial.verbose.cod



Application
Unnamed

Name: Unnamed
Store as project file

Run on: Local machine
Manage targets...

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run
Modify options Alt+M

java 22 SDK of 'inter
interpreter.Interpreter

factorial.verbose.cod

CLI arguments to your application. Alt+R

Working directory: :\daler\OneDrive\Desktop\interpreter-darvr1-main

Environment variables: Environment variables or .env files

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started

5 Assumption Made

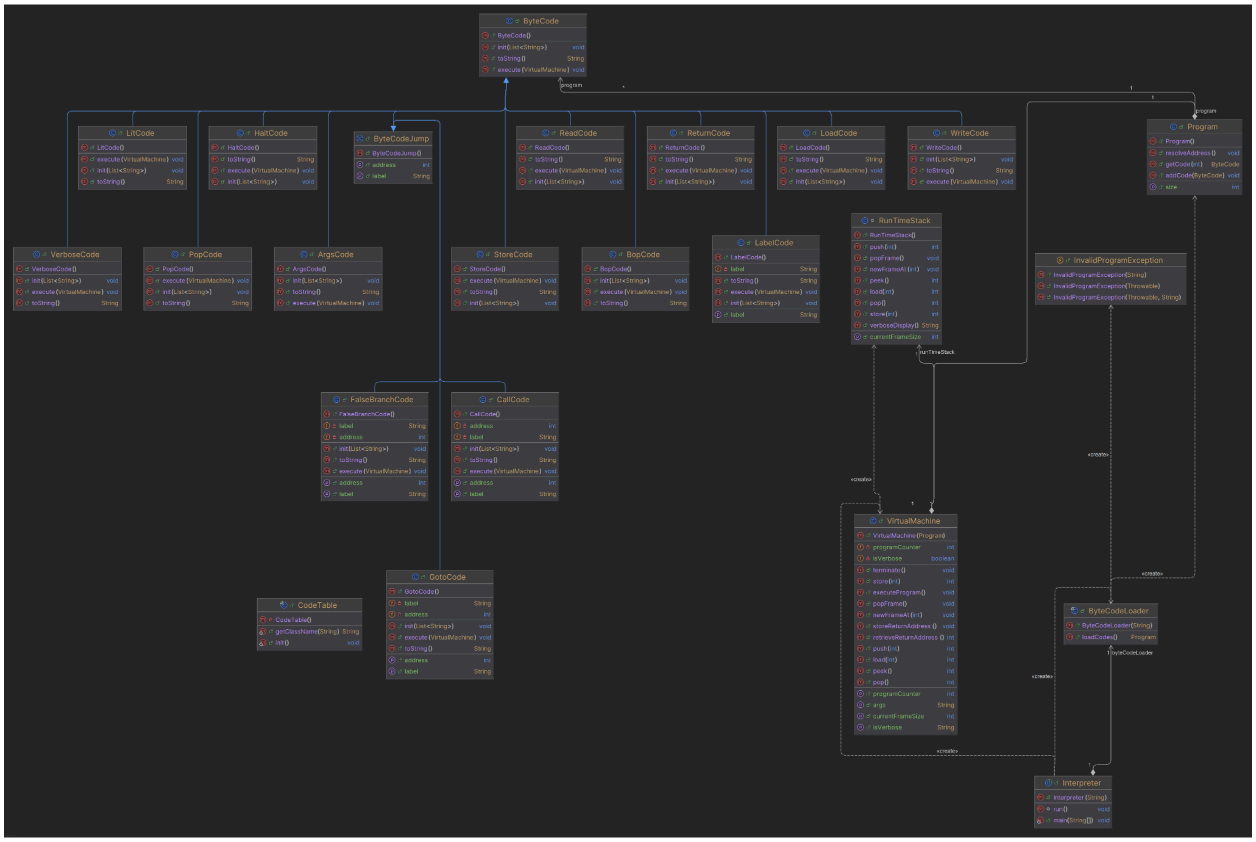
1. The *x* codes are working correctly.
2. ARGS byte code is always called before CALL

6 Implementation Discussion

I decided to create an interface for all the byte codes. I also noticed that I was repeating a lot of the same code in `resolveAddress()` in the Program class. This was because 3 of the byte codes have the ability to jump to labels. This led me to create another interface for the jump byte codes. It was only for three byte codes but it makes the `resolveAddress()` method cleaner.

6.1 Class Diagram

Image is found in the documentation folder.



7 Project Reflection

This project helped me understand how interpreters work, specifically in handling stack-based operations and function calls. The biggest obstacle I faced (again) was procrastination. We were given about 3 weeks to work on this but I only really started working when there were 5 days left. I also needed a lot of time understanding how the virtual machine and the runtime stack worked but thankfully the recorded lecture helped a lot.

8 Project Conclusion/Results

This project was tedious and difficult but I learned a lot from it. It was overwhelming and confusing at the beginning but gradually got easier and more enjoyable. I was too focused on the big picture and kept overthinking every single detail for every byte code but after only focusing on one byte code at a time, it was manageable. The interpreter successfully executes the given .cod files and matches the sample output given by the professor. It helped me understand low-level execution details and managing data flow through the runtime stack.