

CSC 413 Project 2 The Interpreter Documentation

*Calvin Tam
917902523*

*CSC413.03
Fall 2019*

<https://github.com/csc413-03-fall2019/csc413-p2-ctam4>

0 Table of Contents

Table of Contents	2
Introduction	3
Project Overview	3
Technical Overview	3
Summary of Work Completed	3
Development Environment	3
How to Build/Import Your Project	4
How to Run Your Project	14
Assumption Made	16
Implementation Discussion	17
Design Choices	17
Class Diagram	17
Project Reflection	19
Project Conclusion/Results	20
Test Case 1	20
Test Case 2	21
Test Case 3	23
Test Case 4	26

1 Introduction

1.1 Project Overview

This project is to implement an interpreter for the mock language X. The interpreter is responsible for processing byte codes that are created from the source code files with the extension `x`. The interpreter and the VirtualMachine will work together to run a program written in Language X. The two sample programs are a recursive version of computing the *nth* Fibonacci number and recursively finding the factorial of a number. These files have the extension `x.cod`.

1.2 Technical Overview

The project has been based on the template made available by the professor. There are several incomplete classes provided: 1) `ByteCodeLoader`; 2) `CodeTable`; 3) `Program`; 4) `RunTimeStack`; and 5) `VirtualMachine`. Besides, `Interperpreter` class has been provided, and code will not be modified. There is also an empty folder named `bytecode` provided. It contains `ByteCode` and `SymbolicByteCode` classes and its subclasses for different ByteCodes. These are the basic structure of the project.

1.3 Summary of Work Completed

I have created an abstract class `ByteCode` and a subclass of `ByteCode` class that supports handling symbolic address and index named `SymbolicByteCode` classes. All of the following subclasses are subclass of either `ByteCode` or `SymbolicByteCode`: 1) `ArgsCode`; 2) `BopCode`; 3) `CallCode`; 4) `DumpCode`; 5) `FalseBranchCode`; 6) `GotoCode`; 7) `HaltCode`; 8) `LabelCode`; 9) `LitCode`; 10) `LoadCode`; 11) `PopCode`; 12) `ReadCode`; 13) `ReturnCode`; 14) `StoreCode`; and 15) `WriteCode`. They are the foundation of the project.

Moreover, I have completed the implementation of the incomplete classes provided: 1) `ByteCodeLoader`; 2) `CodeTable`; 3) `Program`; 4) `RunTimeStack`; and 5) `VirtualMachine`.

Also, dumping has been implemented per the requirement. It will print out the `ByteCode` and its argument(s) on the first line and `runStack` partitioned by frame pointer on the second line.

2 Development Environment

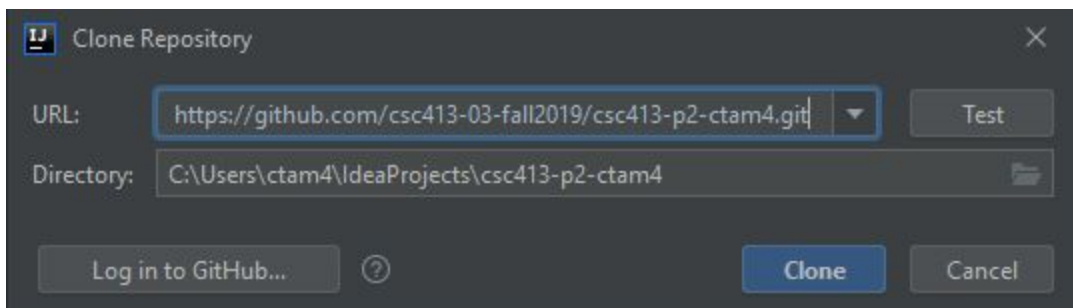
This program has successfully compiled on OpenJDK 11.0.4 in command line and inside IDE IntelliJ IDEA 2019.2.2.

3 How to Build/Import Your Project

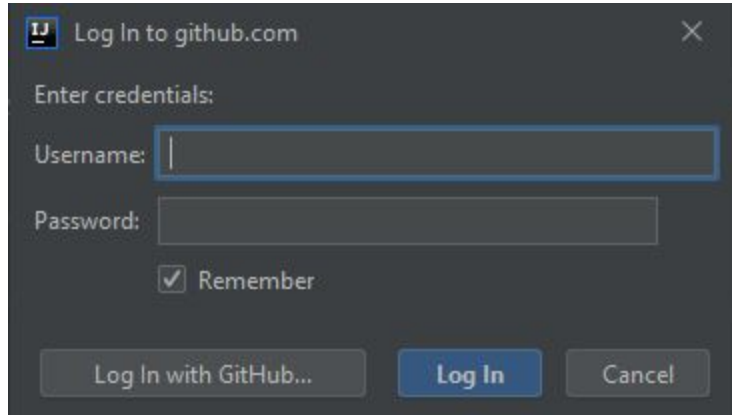
Step 1: Open IntelliJ IDEA, select "Check out from Version Control" and select "Git".



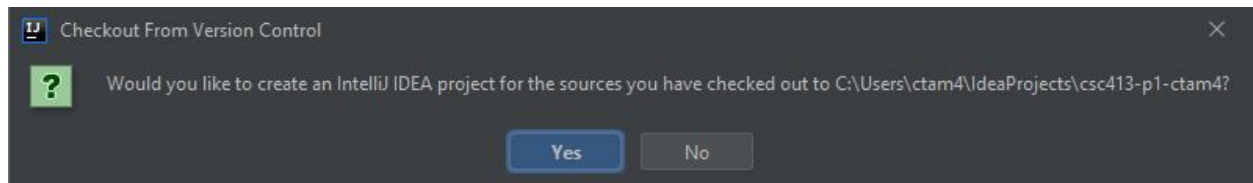
Step 2: Enter "<https://github.com/csc413-03-fall2019/csc413-p2-ctam4.git>" for URL, select directory path at your preference, and select "Clone".



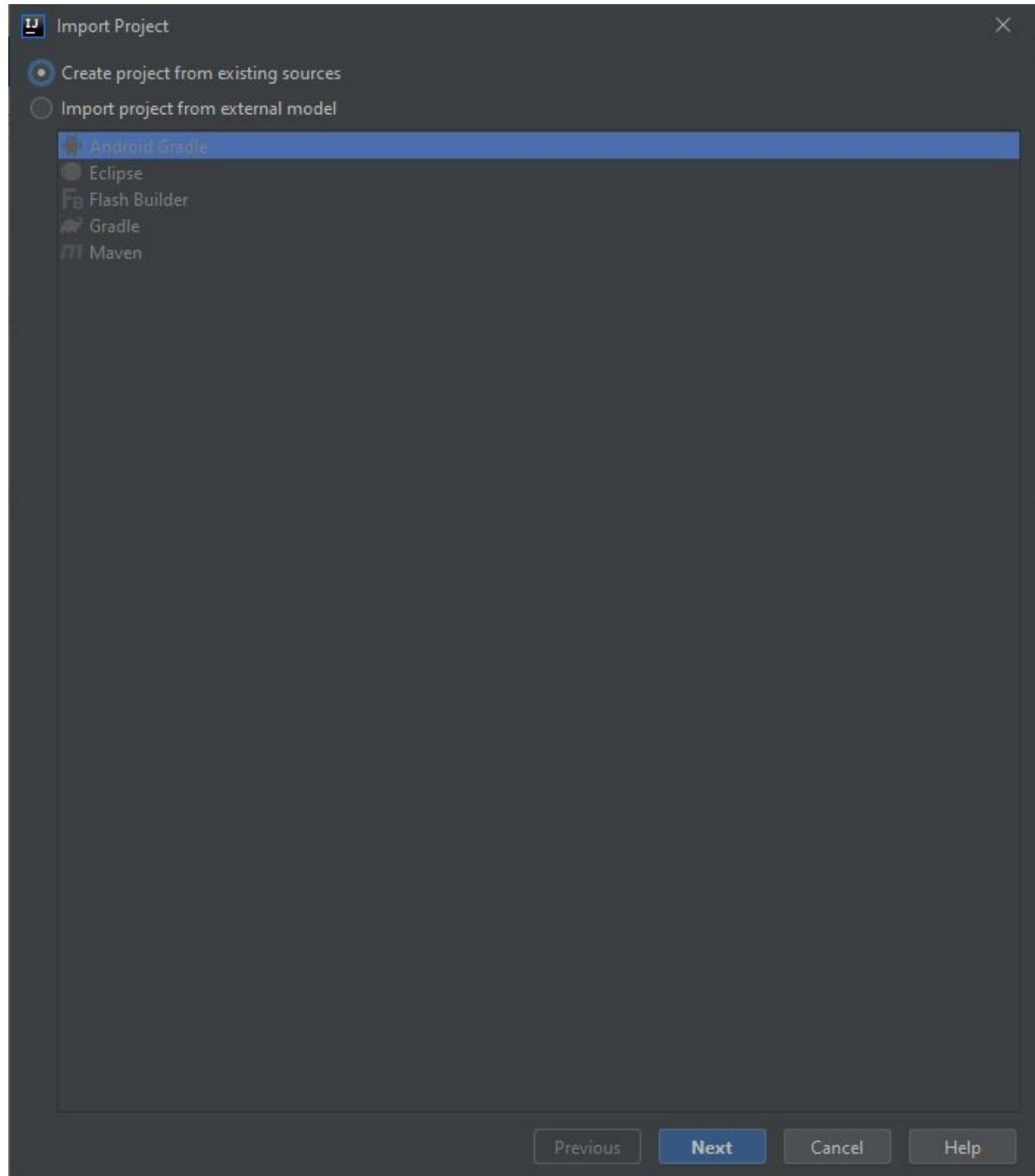
Step 3: Enter your GitHub username and password and select “Log In”.

A screenshot of the 'Log In to github.com' dialog box in IntelliJ IDEA. The dialog has a dark gray background and a title bar with the IntelliJ logo and a close button. It contains a label 'Enter credentials:' followed by two input fields: 'Username:' and 'Password:'. The 'Username:' field is highlighted with a blue border. Below the password field is a checked checkbox labeled 'Remember'. At the bottom, there are three buttons: 'Log In with GitHub...', 'Log In' (highlighted in blue), and 'Cancel'.

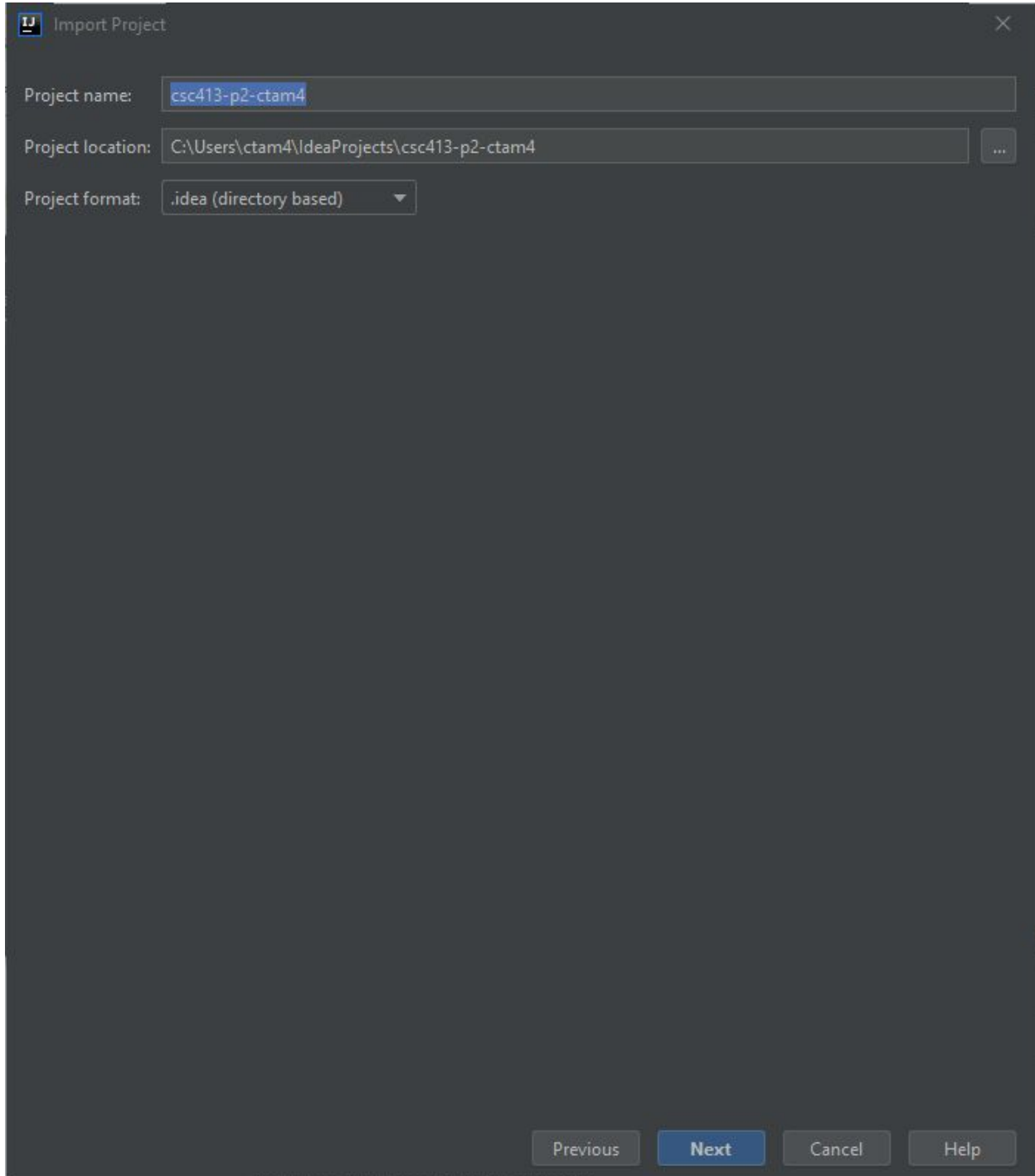
Step 4: Select “Yes”.

A screenshot of the 'Checkout From Version Control' dialog box in IntelliJ IDEA. The dialog has a dark gray background and a title bar with the IntelliJ logo and a close button. It features a green question mark icon on the left. The main text asks: 'Would you like to create an IntelliJ IDEA project for the sources you have checked out to C:\Users\ctam4\IdeaProjects\csc413-p1-ctam4?'. At the bottom, there are two buttons: 'Yes' (highlighted in blue) and 'No'.

Step 5: Select “Next”.



Step 6: Select “Next”.

The image shows a dark-themed 'Import Project' dialog box. At the top left is the IntelliJ logo and the title 'Import Project'. At the top right is a close button (X). The dialog contains three input fields: 'Project name:' with the text 'csc413-p2-ctam4', 'Project location:' with the path 'C:\Users\ctam4\IdeaProjects\csc413-p2-ctam4' and a browse button (...), and 'Project format:' with a dropdown menu showing '.idea (directory based)'. At the bottom right are four buttons: 'Previous', 'Next' (highlighted in blue), 'Cancel', and 'Help'.

Import Project

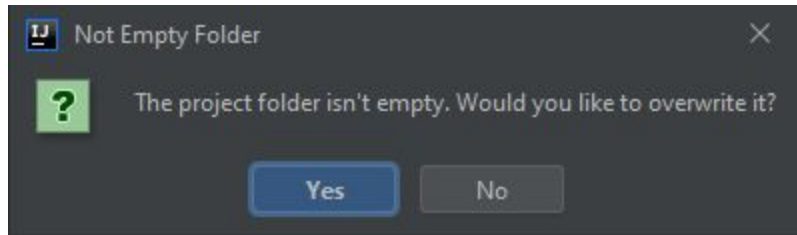
Project name: csc413-p2-ctam4

Project location: C:\Users\ctam4\IdeaProjects\csc413-p2-ctam4 ...

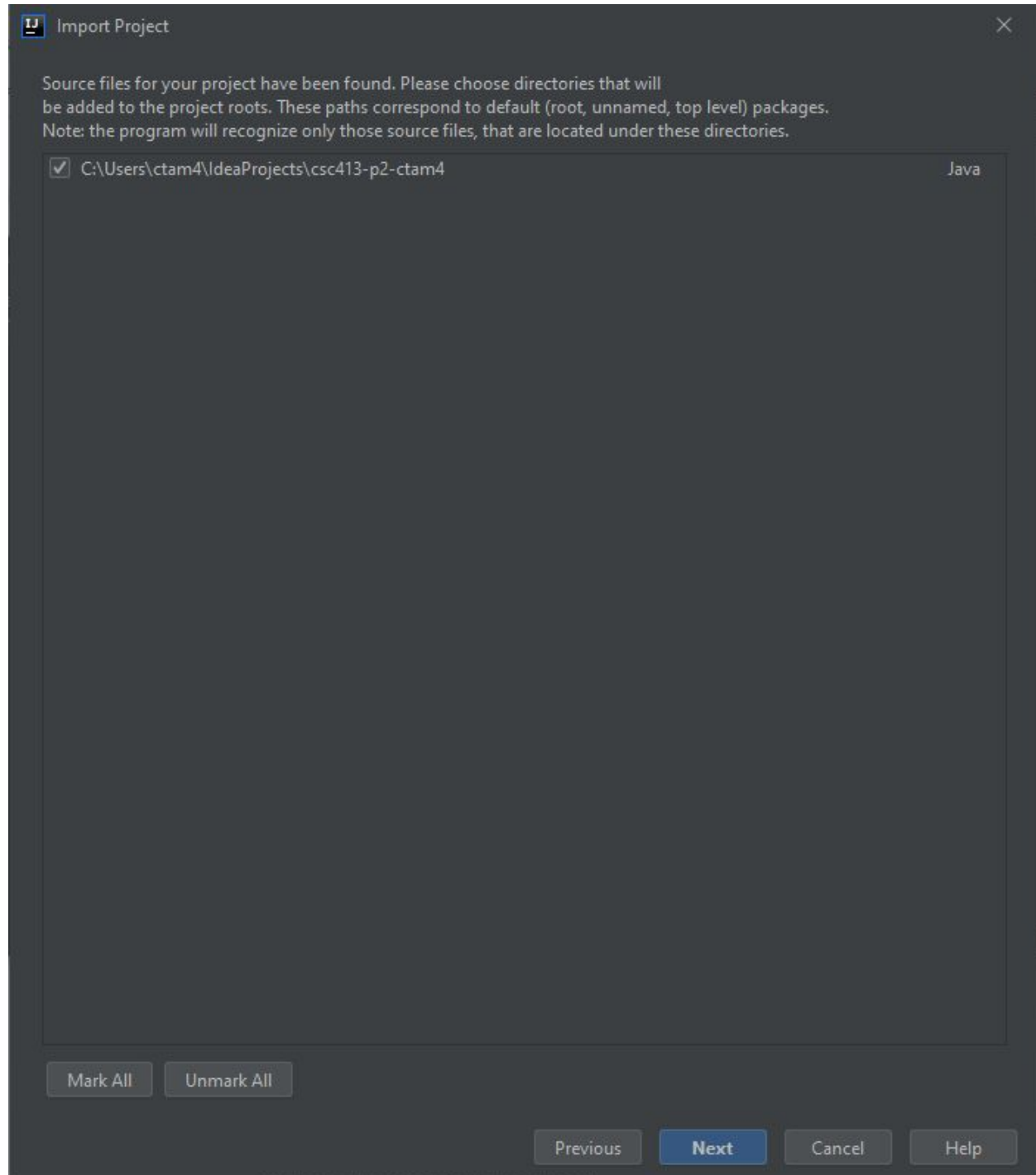
Project format: .idea (directory based) ▼

Previous Next Cancel Help

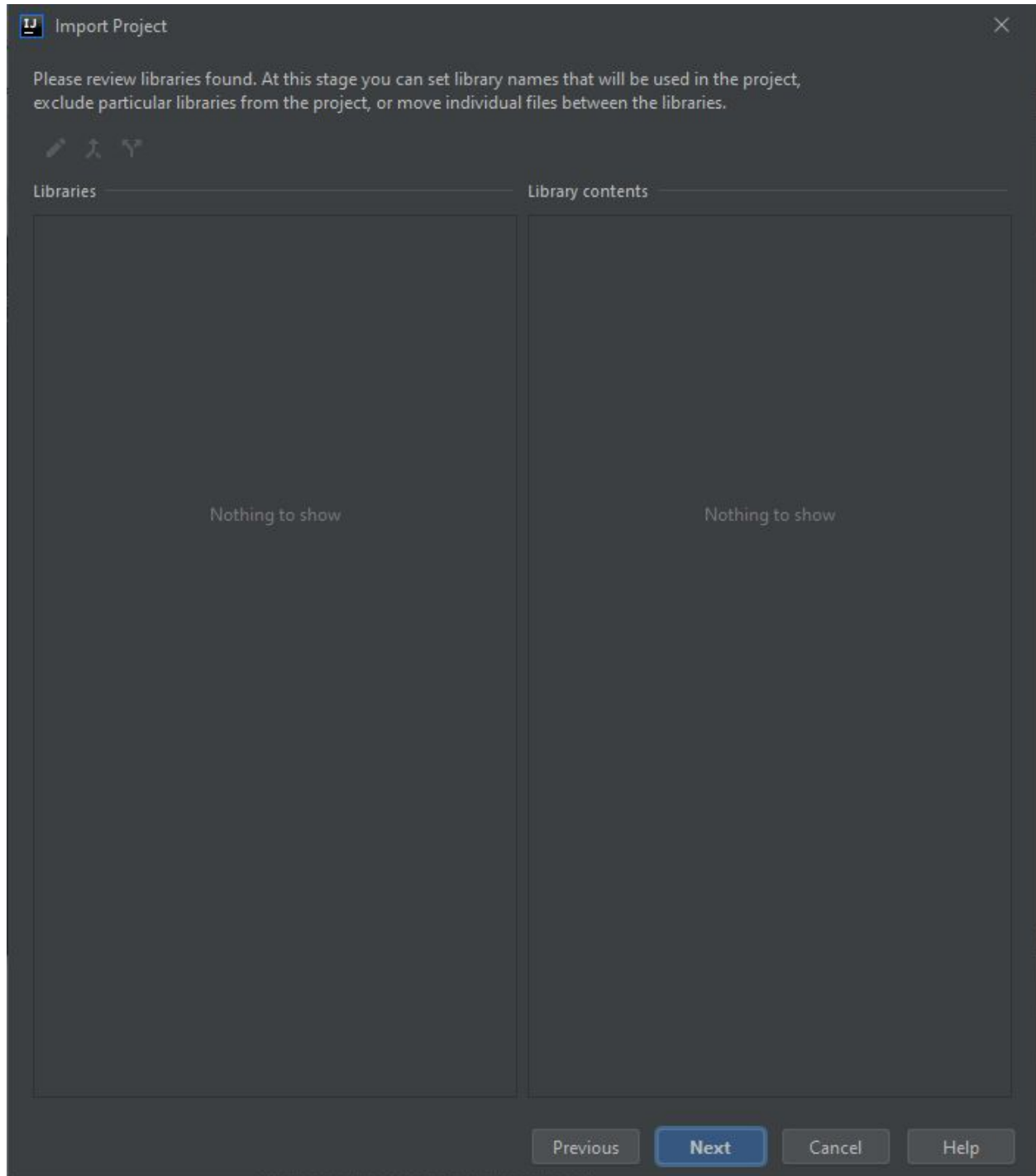
Step 7: Select "Yes".



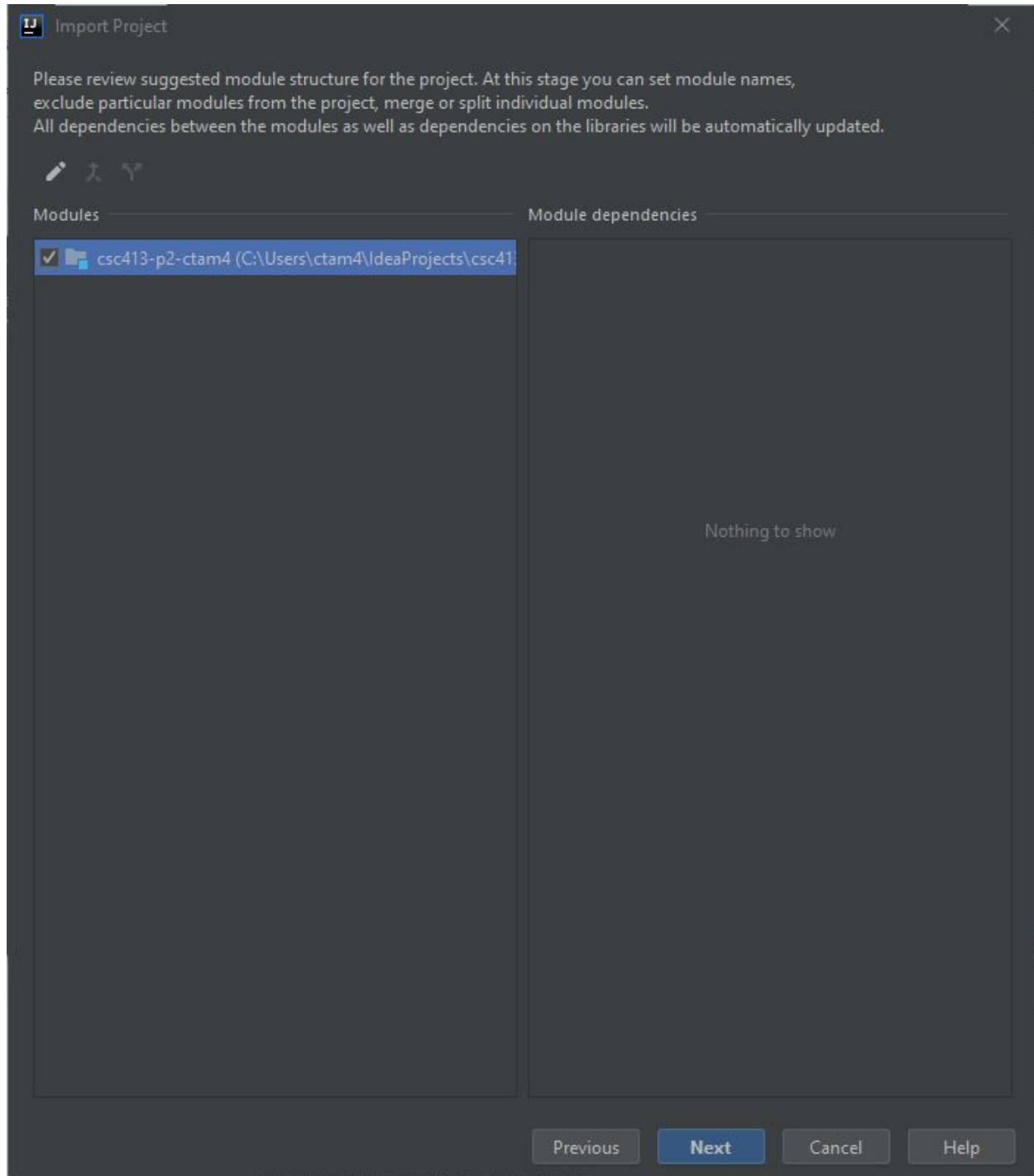
Step 8: Select “Next”.



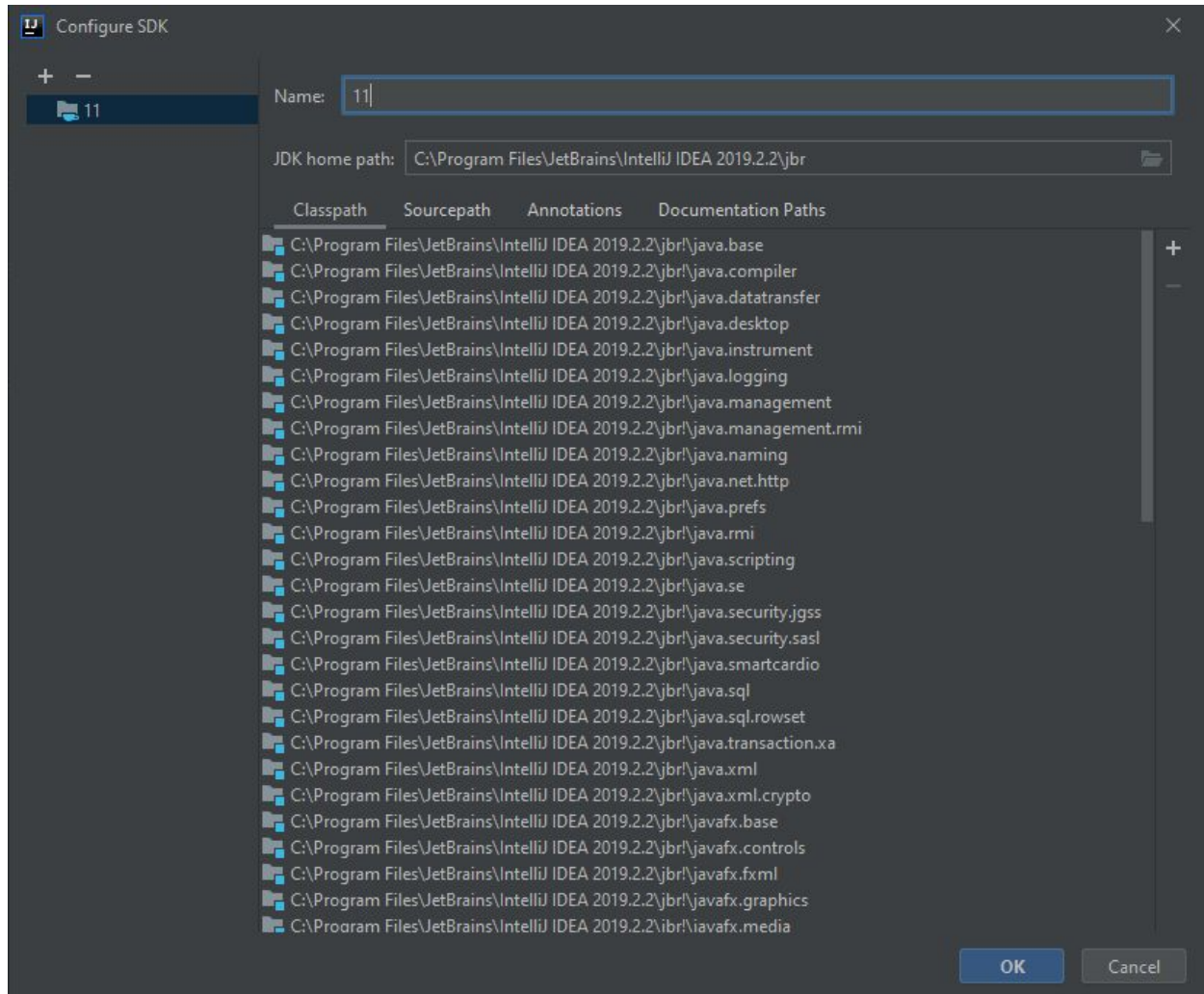
Step 9: Select “Next”.



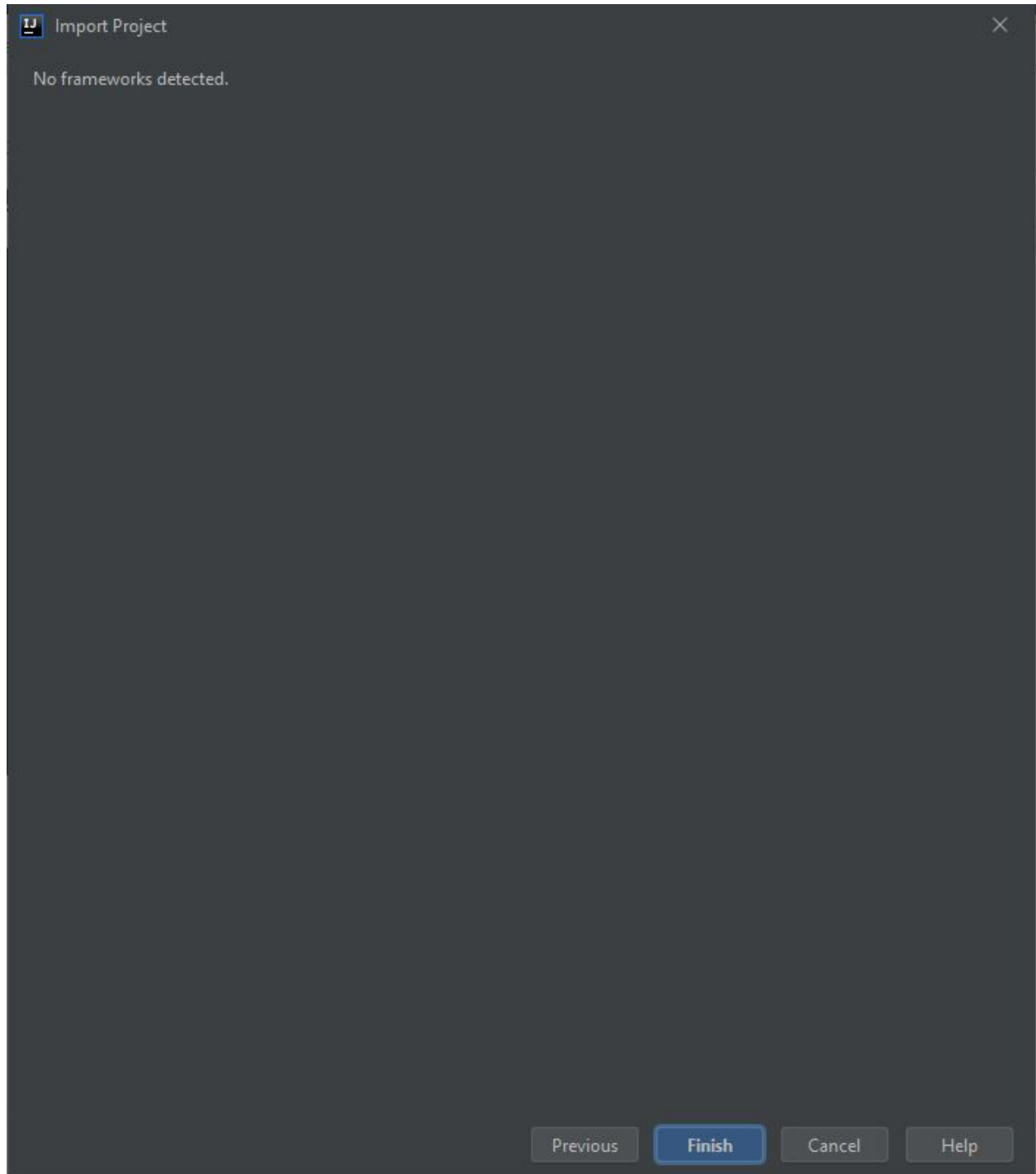
Step 10: Select “Next”.



Step 11: Select “Next”.

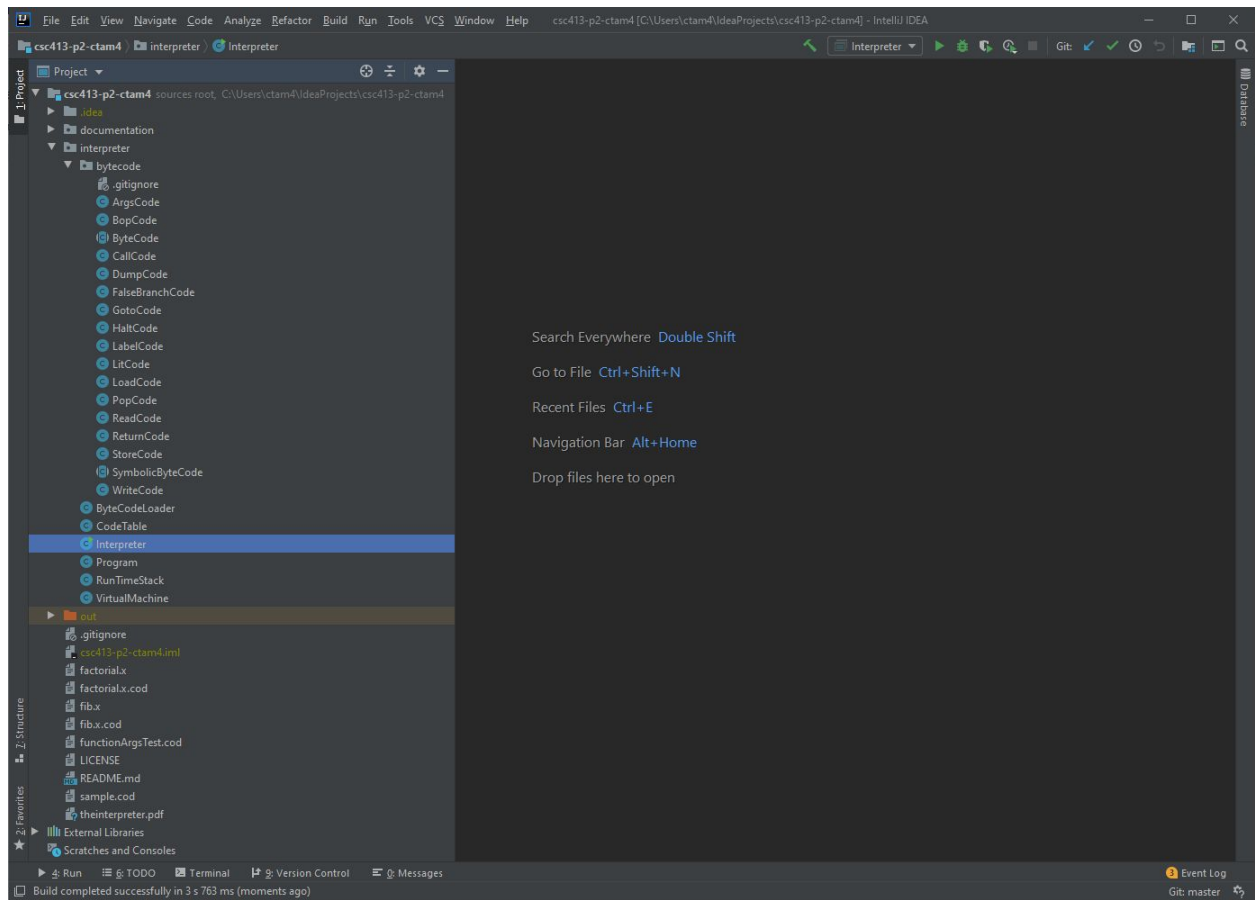


Step 12: Select "Finish".

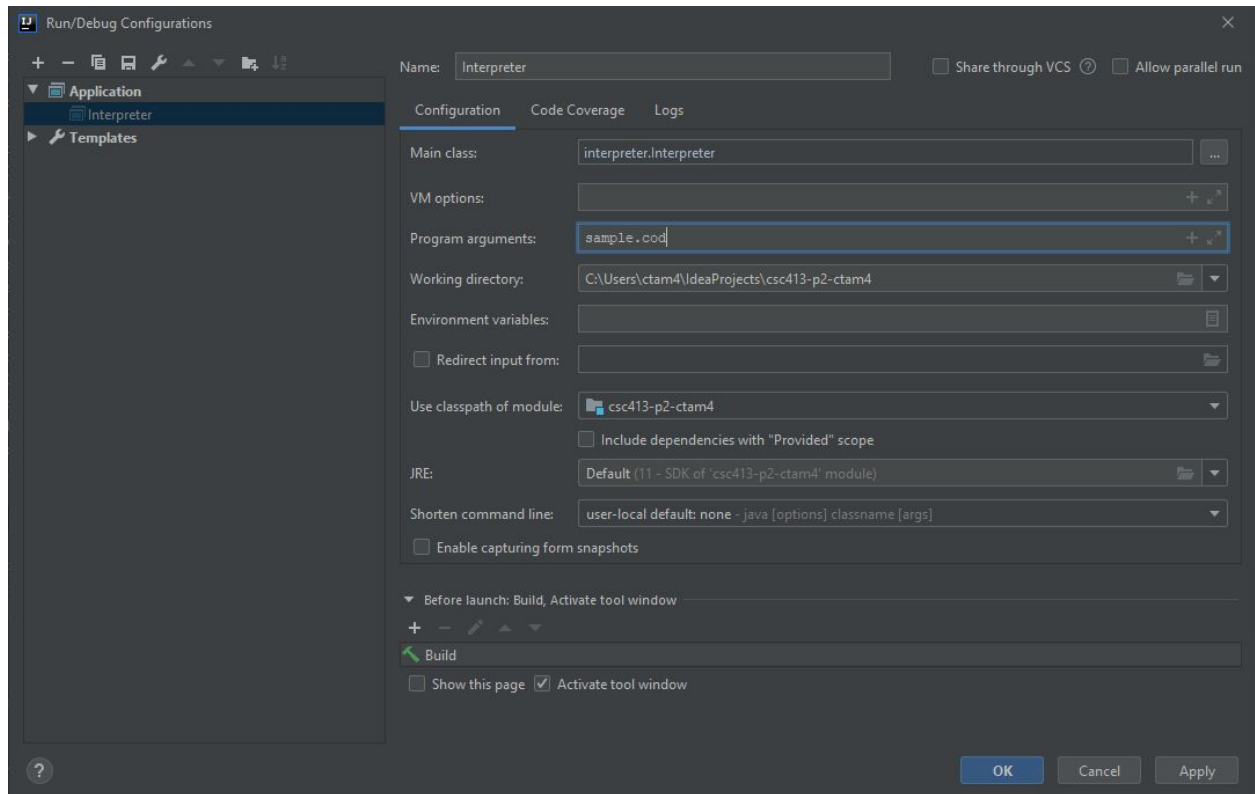


4 How to Run Your Project

Step 1: Select “Interpreter.java”. Right click and select “Run ‘Interpreter.main()’”.



Step 2: Select “Interpreter” from the top bar (right next to hammer) and select “Edit Configurations...”. Enter file name at “Program arguments”. (“sample.cod” is the sample compiled code provided in the requirement.)



5 Assumption Made

End-user Environment:

- 1) JRE 11 or above

Developer Environment:

- 1) JDK 11 or above
- 2) IntelliJ IDEA 2019 or above, or other IDE alternatives

Program Functionality:

- 1) The program only supports the following ByteCodes:
 - a) HALT
 - b) POP
 - c) FALSEBRANCH
 - d) GOTO
 - e) STORE
 - f) LOAD
 - g) LIT
 - h) ARGS
 - i) CALL
 - j) RETURN
 - k) BOP
 - l) READ
 - m) WRITE
 - n) LABEL
 - o) DUMP
- 2) The program throws `IllegalArgumentException` when the number of arguments or the value received is not permitted in ByteCodes' methods. This also applies to methods at `RunTimeStack`.
- 3) The program throws `EmptyStackException` when ByteCodes try to pop the stack and the stack is empty.
- 4) The program throws `IndexOutOfBoundsException` when ByteCodes try to get resolved index and the symbolic address is not resolved.

6 Implementation Discussion

6.1 Design Choices

Since `ByteCode` class is not provided, I created one. It includes the name of the `ByteCode`, `init()` that initialize it, `execute()`, and `toString()` for dumping all arguments. All methods in this class are abstract, meaning that they need to be implemented in the subclasses. Since `CALL`, `FALSEBRANCH`, `GOTO`, and `RETURN` involve symbolic address, I have created another abstract class and subclass of `ByteCode` named `SymbolicByteCode`. This class includes the symbolic address (label) and resolved index, their getters, and setters, and implemented `toString()`. If subclass requires special dumping format, I @Override `SymbolicByteCode's toString()` for the requirements. If `ByteCodes` implemented requires to have access to the `runStack`, they are calling corresponding methods implemented in `VirtualMachine` class. `ByteCodes` do not have direct access to the `runStack` because of encapsulation.

In the `RunTimeStack` class, I have implemented the following methods for `runTimeStack`: 1) `getSize()`; 2) `dump()`; 3) `peek()`; 4) `push()`; 5) `pop()`; 6) `store()`; and 7) `load()`. Nevertheless, I have also implemented these methods for `framePointer`: 1) `getFrameSize()`; 2) `dumpFrame()`; 3) `peekFrame()`; 4) `newFrameAt()`; and 5) `popFrame()`. For these functions' description, parameters and return value, please refer to code files.

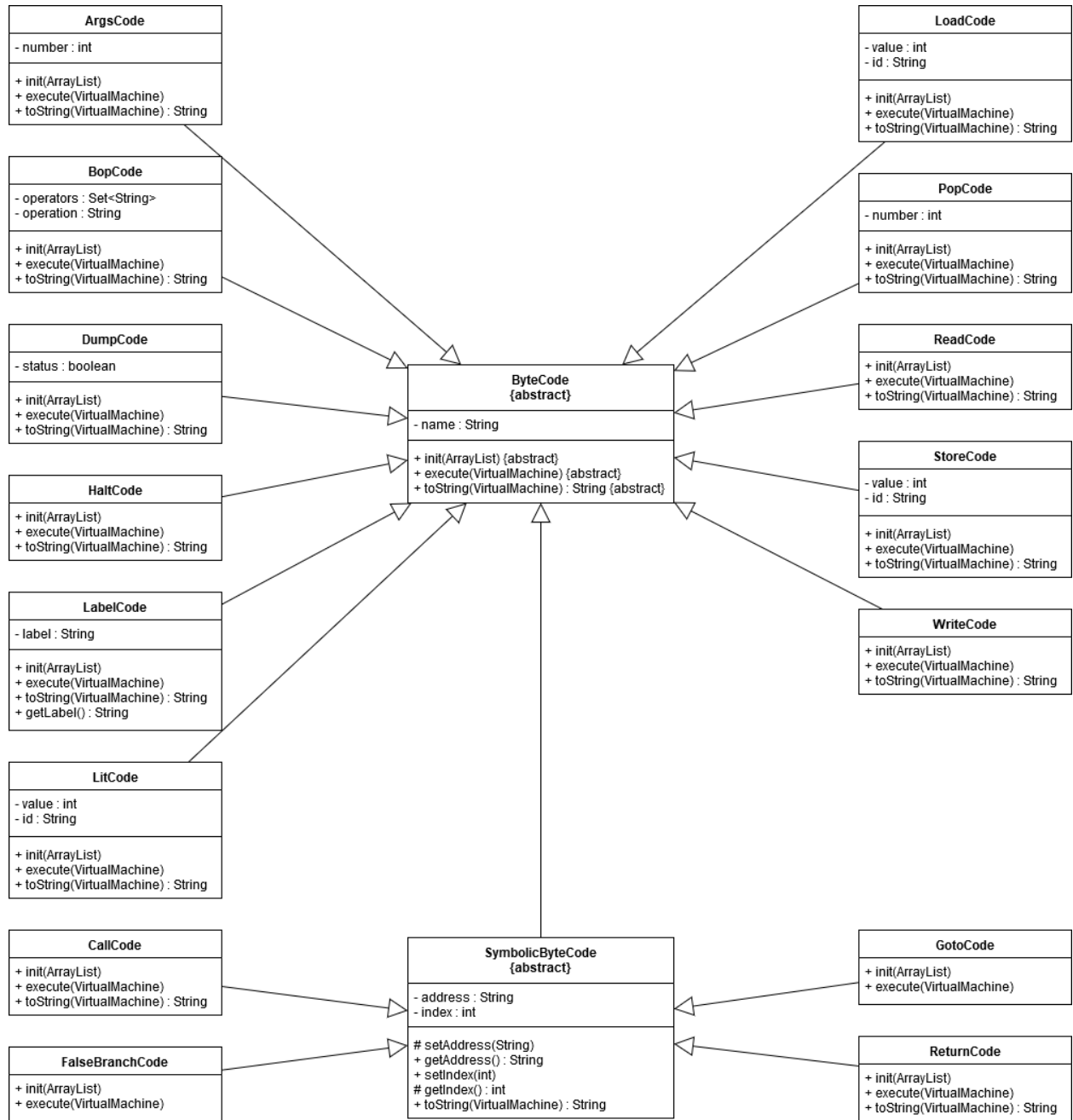
I have implemented error checkings at all `ByteCode` classes and `RunTimeStack` where applicable. The program checks for disallowed argument, so when the argument is `null` or the number of arguments does not match the method expectation, it prints out error message on the console and throws `IllegalArgumentException`. The program also checks for stack size, so when the stack is empty, it does the same as above. Undoubtedly, I have added `try...catch` to catch all exceptions thrown and terminates the program with error code `-1`.

6.2 Class Diagram

`ByteCode` is an abstract class, and it has one private variable, and three two public abstract methods. `SymbolicByteCode` is an abstract class, and a subclass of `ByteCode`. In addition to `ByteCode's` property, it has two private variables, two protected methods, and two public methods. One of the public abstract methods from `ByteCode` has been implemented in `SymbolicByteCode`.

The following classes are inherited from `ByteCode` class: 1) `ArgsCode`; 2) `BopCode`; 3) `DumpCode`; 4) `HaltCode`; 5) `LabelCode`; 6) `LitCode`; 7) `LoadCode`; 8) `PopCode`; 9) `ReadCode`; 10) `StoreCode`; and 11) `WriteCode`.

The following classes are inherited from `SymbolicByteCode` class: 1) `CallCode`; 2) `FalseBranchCode`; 3) `GotoCode`; and 4) `ReturnCode`.



7 Project Reflection

For this assignment, the requirements are long. It is 28 pages long (and I found typos too). The introduction given does not help me understand the program structure. I was confused with the file and program structure until I found some explaining at section 12 (Coding Hints) and 14 (Requirements).

First, there is no file tree of the project files provided. I would suggest having a file tree and a short description of the main classes while leaving the detailed description in their dedicated sections.

Second, the explanation of the program structure is not located in the introduction. They are spread across all sections (mainly in section 14 (Requirements)). The point of the requirement is to help us understand the program structure.

Third, the requirements do not have a dedicated section for `ByteCode`, and all relevant information about this class is scattered across section 3 (Supported ByteCodes), 12.1 and 12.2 (both in Coding Hints). I suppose the missing `ByteCode` section includes the following: 1) this class is abstract; 2) supported `ByteCodes` are subclasses of this class; 3) the package name for `ByteCode` and its subclass is `interpreter.bytecode`; 4) `toString()` method is needed for dumping.

Fourth, there are no test cases provided. The requirements must provide some test cases for us to debug the program, especially unit tests are missing. Without expected input and output value, there is no way I can guarantee the program works as exact.

I managed to figure these out with the help from section 12, but this is not a pleasant experience for anyone to work on this project. The most difficult part of the project is writing this documentation. While I code, I write the comments when I complete the methods, and they are in Javadoc format. It would not only help me using these methods inside IDE but also ease anyone who tries to read the code.

8 Project Conclusion/Results

8.1 Test Case 1

File: "sample.cod" (Sample compiled code)

Result: CORRECT (This matches the sample execution trace provided in the requirement.)

```
DUMP ON
GOTO start<<1>>
[]
LABEL start<<1>>
[]
LIT 0 i      int i
[0]
LIT 0 j      int j
[0,0]
GOTO continue<<3>>
[0,0]
LABEL continue<<3>>
[0,0]
LIT 0 m      int m
[0,0,0]
LIT 3
[0,0,0,3]
ARGS 1
[0,0,0] [3]
CALL f<<2>>   f(3)
[0,0,0] [3]
LABEL f<<2>>
[0,0,0] [3]
LIT 0 j      int j
[0,0,0] [3,0]
LIT 0 k      int k
[0,0,0] [3,0,0]
LOAD 0 i     <load i>
[0,0,0] [3,0,0,3]
LOAD 1 j     <load j>
[0,0,0] [3,0,0,3,0]
BOP +
[0,0,0] [3,0,0,3]
LOAD 2 k     <load k>
[0,0,0] [3,0,0,3,0]
BOP +
[0,0,0] [3,0,0,3]
```

```
LIT 2
[0,0,0] [3,0,0,3,2]
BOP +
[0,0,0] [3,0,0,5]
RETURN f<<2>>  EXIT f : 5
[0,0,0,5]
STORE 2 m    m=5
[0,0,5]
LOAD 1 j     <load j>
[0,0,5,0]
LOAD 2 m     <load m>
[0,0,5,0,5]
BOP +
[0,0,5,5]
ARGS 1
[0,0,5] [5]
CALL Write   Write(5)
[0,0,5] [5]
LABEL Write
[0,0,5] [5]
LOAD 0 dummyformal <load
dummyformal>
[0,0,5] [5,5]
5
WRITE
[0,0,5] [5,5]
RETURN EXIT 5
[0,0,5,5]
STORE 0 i    i=5
[5,0,5]
POP 3
[]
HALT
[]

Process finished with exit code
0
```

8.2 Test Case 2

File: "functionArgsTest.cod"

Result: CORRECT (`EmptyStackException` is expected because the next ByteCode to execute is POP 66, which is impossible when there is only one item at the stack.)

```
DUMP ON
GOTO CONTINUE<<1>>
[]
LABEL CONTINUE<<1>>
[]
GOTO CONTINUE<<2>>
[]
LABEL CONTINUE<<2>>
[]
GOTO CONTINUE<<3>>
[]
LABEL CONTINUE<<3>>
[]
GOTO CONTINUE<<4>>
[]
LABEL CONTINUE<<4>>
[]
LIT 0
[0]
LIT 1
[0,1]
ARGS 2
[] [0,1]
CALL doublePrint<<1>>
doublePrint(0,1)
[] [0,1]
LABEL doublePrint<<1>>
[] [0,1]
LOAD 0
[] [0,1,0]
0
WRITE
[] [0,1,0]
POP 1
[] [0,1]
LOAD 1
[] [0,1,1]
1
WRITE
[] [0,1,1]
POP 1
```

```
[] [0,1]
RETURN doublePrint<<1>> EXIT
doublePrint : 1
[1]
POP 1
[]
LIT 0
[0]
LIT 1
[0,1]
LIT 2
[0,1,2]
ARGS 3
[] [0,1,2]
CALL triplePrint<<1>>
triplePrint(0,1,2)
[] [0,1,2]
LABEL triplePrint<<1>>
[] [0,1,2]
LOAD 0
[] [0,1,2,0]
0
WRITE
[] [0,1,2,0]
POP 1
[] [0,1,2]
LOAD 1
[] [0,1,2,1]
1
WRITE
[] [0,1,2,1]
POP 1
[] [0,1,2]
LOAD 2
[] [0,1,2,2]
2
WRITE
[] [0,1,2,2]
POP 2
[] [0,1]
```

```

RETURN triplePrint<<1>> EXIT
triplePrint : 1
[1]
POP 1
[]
LIT 0
[0]
LIT 1
[0,1]
LIT 2
[0,1,2]
LIT 3
[0,1,2,3]
ARGS 4
[] [0,1,2,3]
CALL quadruplePrint<<1>>
quadruplePrint(0,1,2,3)
[] [0,1,2,3]
LABEL quadruplePrint<<1>>
[] [0,1,2,3]
LOAD 0
[] [0,1,2,3,0]
0
WRITE
[] [0,1,2,3,0]
POP 1
[] [0,1,2,3]
LOAD 1

```

```

[] [0,1,2,3,1]
1
WRITE
[] [0,1,2,3,1]
POP 1
[] [0,1,2,3]
LOAD 2
[] [0,1,2,3,2]
2
WRITE
[] [0,1,2,3,2]
POP 1
[] [0,1,2,3]
LOAD 3
[] [0,1,2,3,3]
3
WRITE
[] [0,1,2,3,3]
POP 1
[] [0,1,2,3]
RETURN quadruplePrint<<1>>
EXIT quadruplePrint : 3
[3]
****
java.util.EmptyStackException

Process finished with exit code
-1

```

8.3 Test Case 3

File: "factorial.x.cod"

Input: 6

Result: CORRECT (EmptyStackException is expected because the next ByteCode to execute is POP 3, which is impossible when there is only one item at the stack.)

```
DUMP ON
GOTO start<<1>>
[]
LABEL start<<1>>
[]
GOTO continue<<3>>
[]
LABEL continue<<3>>
[]
ARGS 0
[] []
CALL Read    Read()
[] []
LABEL Read
[] []
Enter an integer: 6
READ
[] [6]
RETURN  EXIT 6
[6]
ARGS 1
[] [6]
CALL factorial<<2>>
factorial(6)
[] [6]
LABEL factorial<<2>>
[] [6]
LOAD 0 n    <load n>
[] [6,6]
LIT 2
[] [6,6,2]
BOP <
[] [6,0]
FALSEBRANCH else<<4>>
[] [6]
LABEL else<<4>>
[] [6]
LOAD 0 n    <load n>
[] [6,6]
LOAD 0 n    <load n>
```

```
[] [6,6,6]
LIT 1
[] [6,6,6,1]
BOP -
[] [6,6,5]
ARGS 1
[] [6,6] [5]
CALL factorial<<2>>
factorial(5)
[] [6,6] [5]
LABEL factorial<<2>>
[] [6,6] [5]
LOAD 0 n    <load n>
[] [6,6] [5,5]
LIT 2
[] [6,6] [5,5,2]
BOP <
[] [6,6] [5,0]
FALSEBRANCH else<<4>>
[] [6,6] [5]
LABEL else<<4>>
[] [6,6] [5]
LOAD 0 n    <load n>
[] [6,6] [5,5]
LOAD 0 n    <load n>
[] [6,6] [5,5,5]
LIT 1
[] [6,6] [5,5,5,1]
BOP -
[] [6,6] [5,5,4]
ARGS 1
[] [6,6] [5,5] [4]
CALL factorial<<2>>
factorial(4)
[] [6,6] [5,5] [4]
LABEL factorial<<2>>
[] [6,6] [5,5] [4]
LOAD 0 n    <load n>
[] [6,6] [5,5] [4,4]
LIT 2
```

```

[] [6,6] [5,5] [4,4,2]
BOP <
[] [6,6] [5,5] [4,0]
FALSEBRANCH else<<4>>
[] [6,6] [5,5] [4]
LABEL else<<4>>
[] [6,6] [5,5] [4]
LOAD 0 n <load n>
[] [6,6] [5,5] [4,4]
LOAD 0 n <load n>
[] [6,6] [5,5] [4,4,4]
LIT 1
[] [6,6] [5,5] [4,4,4,1]
BOP -
[] [6,6] [5,5] [4,4,3]
ARGS 1
[] [6,6] [5,5] [4,4] [3]
CALL factorial<<2>>
factorial(3)
[] [6,6] [5,5] [4,4] [3]
LABEL factorial<<2>>
[] [6,6] [5,5] [4,4] [3]
LOAD 0 n <load n>
[] [6,6] [5,5] [4,4] [3,3]
LIT 2
[] [6,6] [5,5] [4,4] [3,3,2]
BOP <
[] [6,6] [5,5] [4,4] [3,0]
FALSEBRANCH else<<4>>
[] [6,6] [5,5] [4,4] [3]
LABEL else<<4>>
[] [6,6] [5,5] [4,4] [3]
LOAD 0 n <load n>
[] [6,6] [5,5] [4,4] [3,3]
LOAD 0 n <load n>
[] [6,6] [5,5] [4,4] [3,3,3]
LIT 1
[] [6,6] [5,5] [4,4] [3,3,3,1]
BOP -
[] [6,6] [5,5] [4,4] [3,3,2]
ARGS 1
[] [6,6] [5,5] [4,4] [3,3] [2]
CALL factorial<<2>>
factorial(2)
[] [6,6] [5,5] [4,4] [3,3] [2]
LABEL factorial<<2>>
[] [6,6] [5,5] [4,4] [3,3] [2]
LOAD 0 n <load n>

```

```

[] [6,6] [5,5] [4,4] [3,3]
[2,2]
LIT 2
[] [6,6] [5,5] [4,4] [3,3]
[2,2,2]
BOP <
[] [6,6] [5,5] [4,4] [3,3]
[2,0]
FALSEBRANCH else<<4>>
[] [6,6] [5,5] [4,4] [3,3] [2]
LABEL else<<4>>
[] [6,6] [5,5] [4,4] [3,3] [2]
LOAD 0 n <load n>
[] [6,6] [5,5] [4,4] [3,3]
[2,2]
LOAD 0 n <load n>
[] [6,6] [5,5] [4,4] [3,3]
[2,2,2]
LIT 1
[] [6,6] [5,5] [4,4] [3,3]
[2,2,2,1]
BOP -
[] [6,6] [5,5] [4,4] [3,3]
[2,2,1]
ARGS 1
[] [6,6] [5,5] [4,4] [3,3]
[2,2] [1]
CALL factorial<<2>>
factorial(1)
[] [6,6] [5,5] [4,4] [3,3]
[2,2] [1]
LABEL factorial<<2>>
[] [6,6] [5,5] [4,4] [3,3]
[2,2] [1]
LOAD 0 n <load n>
[] [6,6] [5,5] [4,4] [3,3]
[2,2] [1,1]
LIT 2
[] [6,6] [5,5] [4,4] [3,3]
[2,2] [1,1,2]
BOP <
[] [6,6] [5,5] [4,4] [3,3]
[2,2] [1,1]
FALSEBRANCH else<<4>>
[] [6,6] [5,5] [4,4] [3,3]
[2,2] [1]
LIT 1

```



```

[] [6,6] [5,5] [4,4] [3,3]
[2,2] [1,1]
RETURN factorial<<2>> EXIT
factorial : 1
[] [6,6] [5,5] [4,4] [3,3]
[2,2,1]
BOP *
[] [6,6] [5,5] [4,4] [3,3]
[2,2]
RETURN factorial<<2>> EXIT
factorial : 2
[] [6,6] [5,5] [4,4] [3,3,2]
BOP *
[] [6,6] [5,5] [4,4] [3,6]
RETURN factorial<<2>> EXIT
factorial : 6
[] [6,6] [5,5] [4,4,6]
BOP *
[] [6,6] [5,5] [4,24]
RETURN factorial<<2>> EXIT
factorial : 24
[] [6,6] [5,5,24]
BOP *
[] [6,6] [5,120]
RETURN factorial<<2>> EXIT
factorial : 120

```

```

[] [6,6,120]
BOP *
[] [6,720]
RETURN factorial<<2>> EXIT
factorial : 720
[720]
ARGS 1
[] [720]
CALL Write Write(720)
[] [720]
LABEL Write
[] [720]
LOAD 0 dummyFormal <load
dummyFormal>
[] [720,720]
720
WRITE
[] [720,720]
RETURN EXIT 720
[720]
****
java.util.EmptyStackException

Process finished with exit code
-1

```

8.4 Test Case 4

File: "fib.x.cod"

Input: 4

Result: CORRECT

```
DUMP ON
GOTO start<<1>>
[]
LABEL start<<1>>
[]
LIT 0 x      int x
[0]
GOTO continue<<3>>
[0]
LABEL continue<<3>>
[0]
LIT 0 k      int k
[0,0]
LIT 5
[0,0,5]
STORE 0 x    x=0
[5,0]
ARGS 0
[5,0] []
CALL Read    Read()
[5,0] []
LABEL Read
[5,0] []
Enter an integer: 4
READ
[5,0] [4]
RETURN EXIT 4
[5,0,4]
ARGS 1
[5,0] [4]
CALL fib<<2>>  fib(4)
[5,0] [4]
LABEL fib<<2>>
[5,0] [4]
LOAD 0 n      <load n>
[5,0] [4,4]
LIT 1
[5,0] [4,4,1]
BOP <=
[5,0] [4,0]
FALSEBRANCH else<<4>>
```

```
[5,0] [4]
LABEL else<<4>>
[5,0] [4]
LOAD 0 n      <load n>
[5,0] [4,4]
LIT 2
[5,0] [4,4,2]
BOP ==
[5,0] [4,0]
FALSEBRANCH else<<6>>
[5,0] [4]
LABEL else<<6>>
[5,0] [4]
LOAD 0 n      <load n>
[5,0] [4,4]
LIT 2
[5,0] [4,4,2]
BOP -
[5,0] [4,2]
ARGS 1
[5,0] [4] [2]
CALL fib<<2>>  fib(2)
[5,0] [4] [2]
LABEL fib<<2>>
[5,0] [4] [2]
LOAD 0 n      <load n>
[5,0] [4] [2,2]
LIT 1
[5,0] [4] [2,2,1]
BOP <=
[5,0] [4] [2,0]
FALSEBRANCH else<<4>>
[5,0] [4] [2]
LABEL else<<4>>
[5,0] [4] [2]
LOAD 0 n      <load n>
[5,0] [4] [2,2]
LIT 2
[5,0] [4] [2,2,2]
BOP ==
[5,0] [4] [2,1]
```

```

FALSEBRANCH else<<6>>
[5,0] [4] [2]
LIT 1
[5,0] [4] [2,1]
RETURN fib<<2>> EXIT fib : 1
[5,0] [4,1]
LOAD 0 n <load n>
[5,0] [4,1,4]
LIT 1
[5,0] [4,1,4,1]
BOP -
[5,0] [4,1,3]
ARGS 1
[5,0] [4,1] [3]
CALL fib<<2>> fib(3)
[5,0] [4,1] [3]
LABEL fib<<2>>
[5,0] [4,1] [3]
LOAD 0 n <load n>
[5,0] [4,1] [3,3]
LIT 1
[5,0] [4,1] [3,3,1]
BOP <=
[5,0] [4,1] [3,0]
FALSEBRANCH else<<4>>
[5,0] [4,1] [3]
LABEL else<<4>>
[5,0] [4,1] [3]
LOAD 0 n <load n>
[5,0] [4,1] [3,3]
LIT 2
[5,0] [4,1] [3,3,2]
BOP ==
[5,0] [4,1] [3,0]
FALSEBRANCH else<<6>>
[5,0] [4,1] [3]
LABEL else<<6>>
[5,0] [4,1] [3]
LOAD 0 n <load n>
[5,0] [4,1] [3,3]
LIT 2
[5,0] [4,1] [3,3,2]
BOP -
[5,0] [4,1] [3,1]
ARGS 1
[5,0] [4,1] [3] [1]
CALL fib<<2>> fib(1)
[5,0] [4,1] [3] [1]

```

```

LABEL fib<<2>>
[5,0] [4,1] [3] [1]
LOAD 0 n <load n>
[5,0] [4,1] [3] [1,1]
LIT 1
[5,0] [4,1] [3] [1,1,1]
BOP <=
[5,0] [4,1] [3] [1,1]
FALSEBRANCH else<<4>>
[5,0] [4,1] [3] [1]
LIT 1
[5,0] [4,1] [3] [1,1]
RETURN fib<<2>> EXIT fib : 1
[5,0] [4,1] [3,1]
LOAD 0 n <load n>
[5,0] [4,1] [3,1,3]
LIT 1
[5,0] [4,1] [3,1,3,1]
BOP -
[5,0] [4,1] [3,1,2]
ARGS 1
[5,0] [4,1] [3,1] [2]
CALL fib<<2>> fib(2)
[5,0] [4,1] [3,1] [2]
LABEL fib<<2>>
[5,0] [4,1] [3,1] [2]
LOAD 0 n <load n>
[5,0] [4,1] [3,1] [2,2]
LIT 1
[5,0] [4,1] [3,1] [2,2,1]
BOP <=
[5,0] [4,1] [3,1] [2,0]
FALSEBRANCH else<<4>>
[5,0] [4,1] [3,1] [2]
LABEL else<<4>>
[5,0] [4,1] [3,1] [2]
LOAD 0 n <load n>
[5,0] [4,1] [3,1] [2,2]
LIT 2
[5,0] [4,1] [3,1] [2,2,2]
BOP ==
[5,0] [4,1] [3,1] [2,1]
FALSEBRANCH else<<6>>
[5,0] [4,1] [3,1] [2]
LIT 1
[5,0] [4,1] [3,1] [2,1]
RETURN fib<<2>> EXIT fib : 1
[5,0] [4,1] [3,1,1]

```

```

BOP +
[5,0] [4,1] [3,2]
RETURN fib<<2>> EXIT fib : 2
[5,0] [4,1,2]
BOP +
[5,0] [4,3]
RETURN fib<<2>> EXIT fib : 3
[5,0,3]
ARGS 1
[5,0] [3]
CALL Write Write(3)
[5,0] [3]
LABEL Write
[5,0] [3]
LOAD 0 dummyFormal <load
dummyFormal>
[5,0] [3,3]
3
WRITE
[5,0] [3,3]
RETURN EXIT 3
[5,0,3]

```

```

STORE 1 k k=3
[5,3]
LIT 0 x int x
[5,3,0]
LIT 7
[5,3,0,7]
STORE 2 x x=7
[5,3,7]
LIT 8
[5,3,7,8]
STORE 2 x x=8
[5,3,8]
POP 1
[5,3]
POP 2
[]
HALT
[]

```

Process finished with exit code
0