

# Compile to JVM

Jiawei Chen, Zhetuo Qi

Zhejiang University

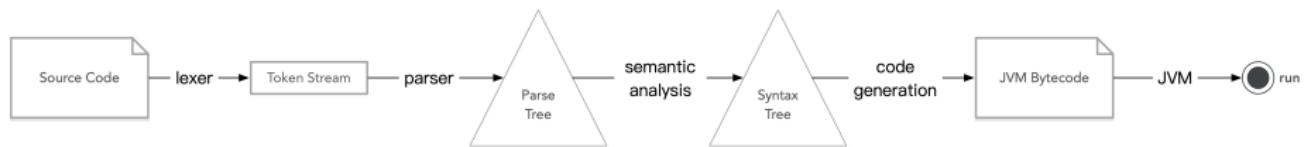
June 15, 2019

# Language Specification

## Example

```
1 namespace Test {  
2     struct MyStruct {  
3         int a = f();  
4     }  
5  
6     double d = 4;  
7  
8     def void main(String[] args) {  
9         print("Hello world!\n");  
10    }  
11  
12    def int f() {  
13        return 3;  
14    }  
15 }
```

# Compiler Architecture



# Platform & Dependency

- Java 8
- Gradle
- COMMONS-CLI: parse the command line
- ANTLR 4: lexical/syntactic analysis
- ASM: help to generate Java bytecode

# Lexical Analysis

With the help of ANTLR, lexical analysis can be easy:

## Lexical Rules Example

```
1 WHITE_SPACE: [ \t\r\n]+ -> skip;
2 LINE_COMMENT: '//' r\n]* -> skip;
3 CHAR_LITERAL: '\'' [a-zA-Z\\] '\'';
```

# Syntactic Analysis

By ANTLR, we can write the following parsing rules. All the rules are placed in a .g4 file.

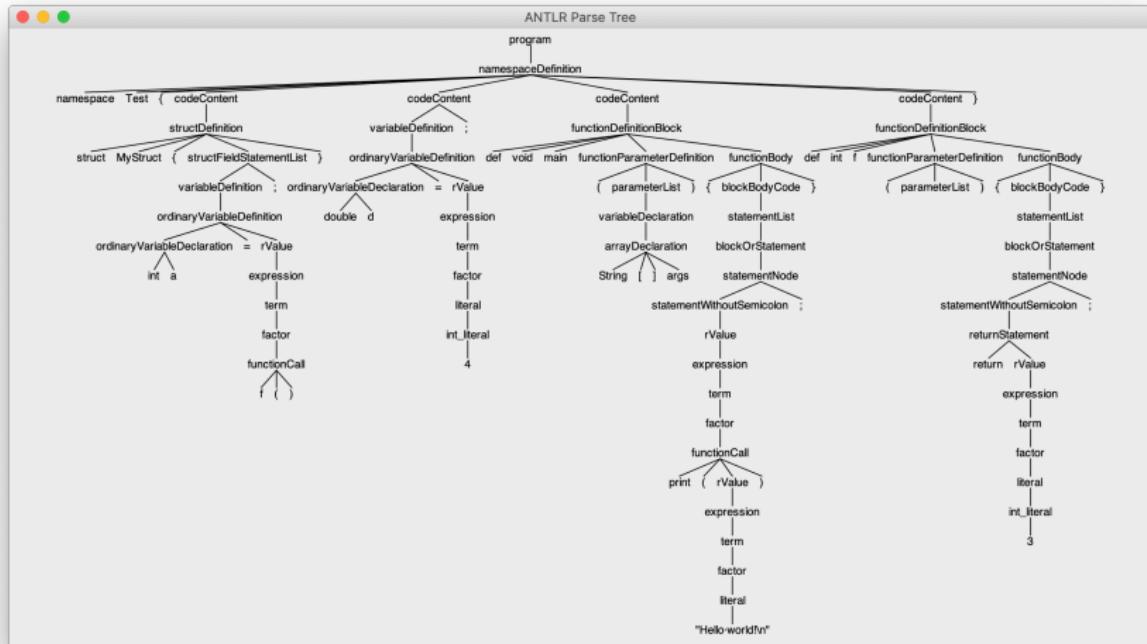
## Parsing Rules Example

```
1 namespaceDefinition:  
2     NAMESPACE_SYMBOL IDENTIFIER LEFT_CURLY_BRACE codeContent+  
         RIGHT_CURLY_BRACE;
```

**Note** : ANTLR doesn't support left-recursive, we have to solve it manually.

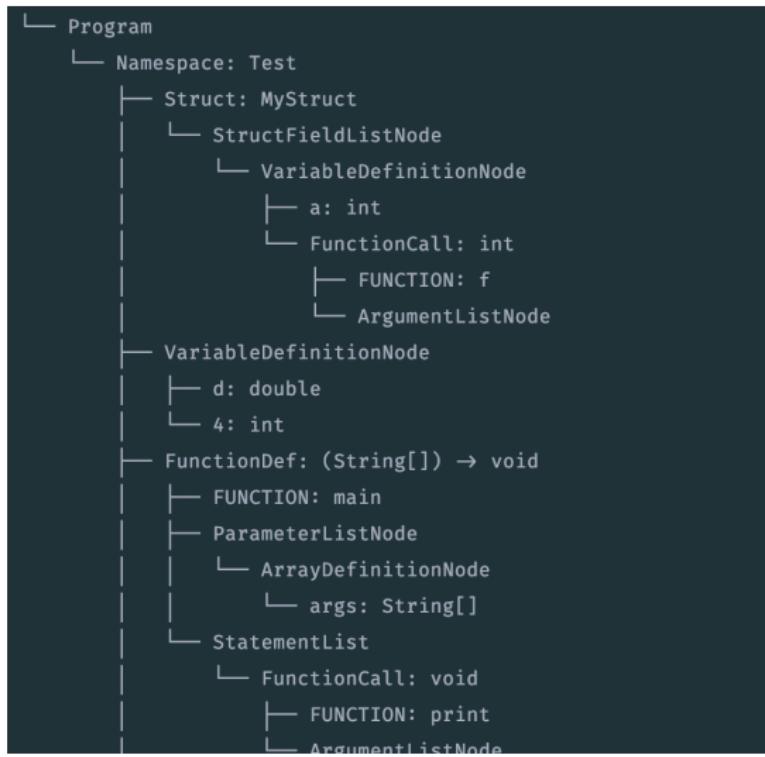
## Syntactic Analysis

## Result of Syntactic Analysis:



# Semantic Analysis

## Syntax Tree:



## ■ Map from language to JVM inner structure:

Structure in this language	Corresponding structure in JVM
Namespace	Class
Struct	Inner class
Fields in Struct	Fields in inner class
Variable declared in namespace	Static fields in class
function	Static method in class
other	Same as JVM

# Code Generation

- Equivalent code between this language and Java:

```
namespace Test {  
    struct MyStruct {  
        int a = f();  
    }  
  
    double d = 4;  
  
    def void main(String[] args) {  
        print("Hello world!\n");  
    }  
  
    def int f() {  
        return 3;  
    }  
}
```



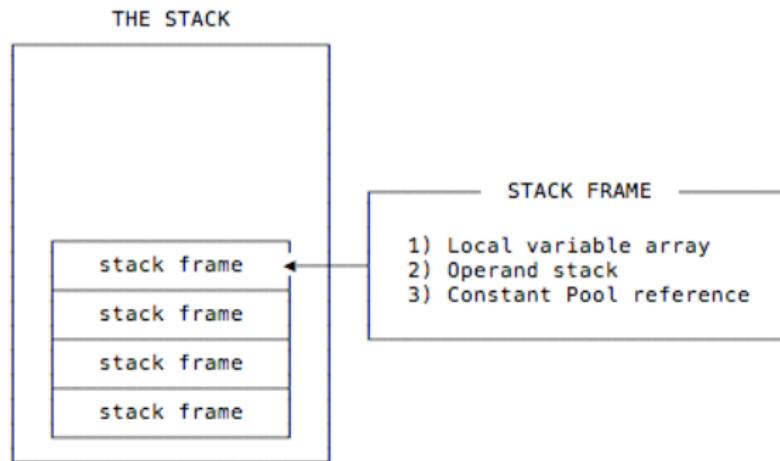
```
public class Test {  
    public static class MyStruct {  
        public int a = f();  
    }  
  
    public static double d = 4;  
  
    public static void main(String[] args) {  
        System.out.println("Hello world!\n");  
    }  
  
    public static int f() {  
        return 3;  
    }  
}
```

## ■ Format of JVM .class file

Magic	Version
Constant Pool	
Access Flags	
this Class	
super Class	
Interfaces	
Fields	
Methods	
Attributes	

# Code Generation

- JVM is a stack machine
- JVM assembly language is a kind of P-Code



# Code Generation

## Constant Pool

Constant Pool:

```
num I str, Ljava/lang/String, "hello", func, Test, java/lang/Object ...
```

```
class Test {  
    public static int num = 3;  
    public static String str = "hello";  
  
    public static String func() {  
        return num + str;  
    }  
}
```

# Code Generation

index of constant pool

```
public static java.lang.String func();
descriptor: ()Ljava/lang/String;
flags: (0x0009) ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=0, args_size=0
0: getstatic    #2           // Field num:I
3: getstatic    #3           // Field str:Ljava/lang/String;
6: invokedynamic #4,  0      // InvokeDynamic #0:makeConcatWithConstants
11: areturn
```

```
class Test {
    public static int num = 3;
    public static String str = "hello";

    public static String func() {
        return num + str;
    }
}
```

# Contributions

- Jiawei Chen
  - lexical analysis (part), semantic analysis, code generation (part), test
- Zhetuo Qi
  - lexical analysis (part), syntactic analysis, code generation (part)