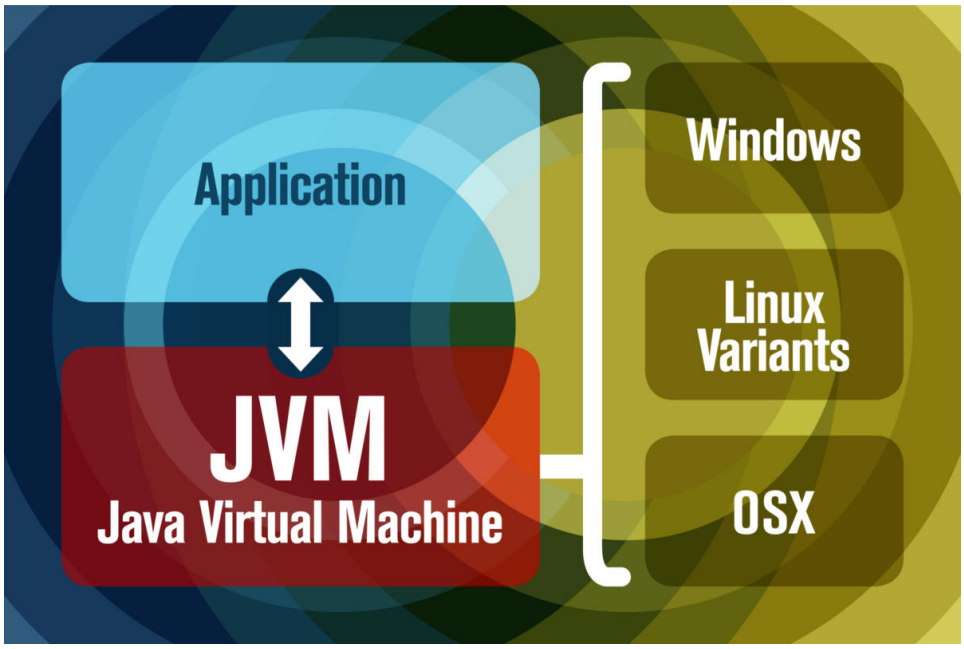
Compiler



Emre Yavuz
17-03-2020
Vak: Compilers & Operating Systems

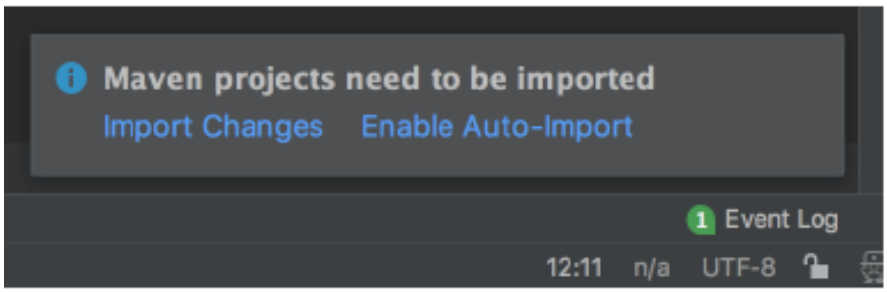
Inhoudsopgave

1 Inleiding

Tijdens dit opdracht ging je een eigen Java Virtual Machine taal implementeren door middel van Antlr en Jasmin. Ik heb ervoor gekozen om de taal KiragedScript te noemen.

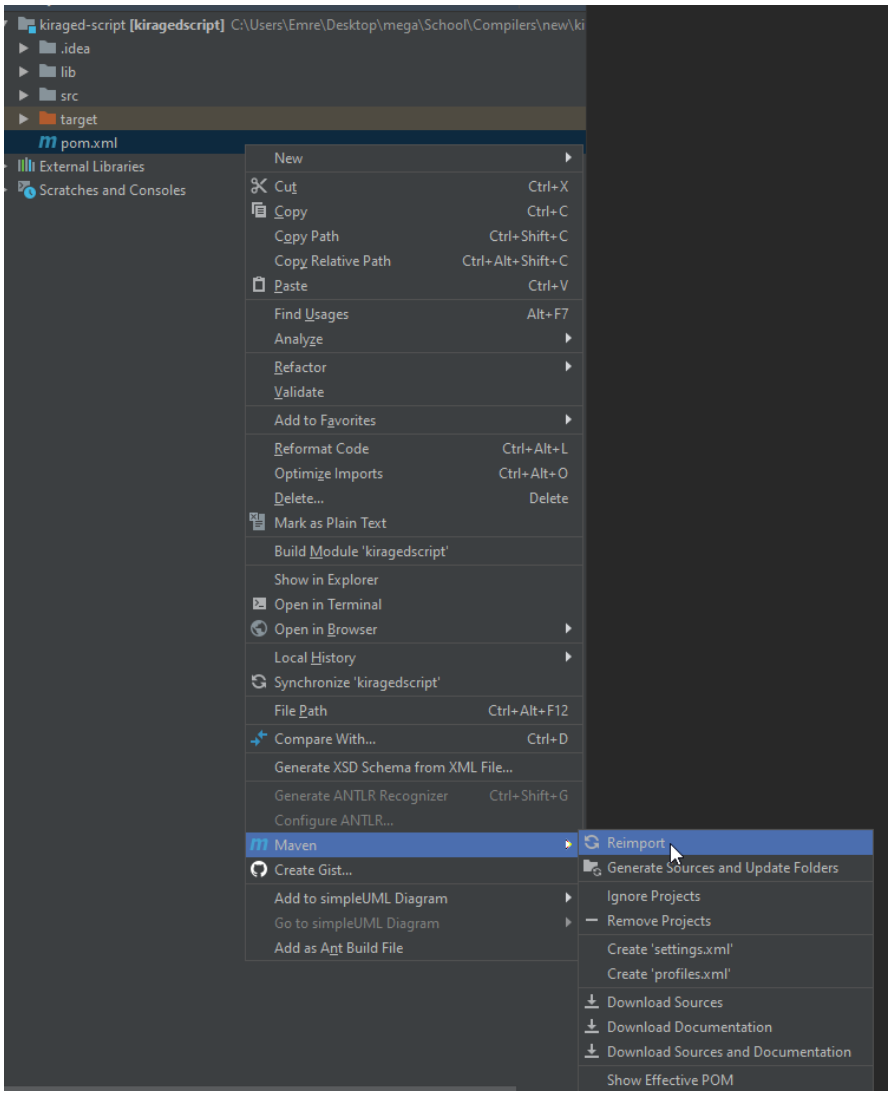
2 Installatie Gids

1. Open de compiler in Intellij.
2. Als de server is geopend in IntellIj zou je een pop up moeten krijgen.



3. Klik op Import Changes.

4. Als dit niet het geval is, is het ook mogelijk om hierop te klikken:

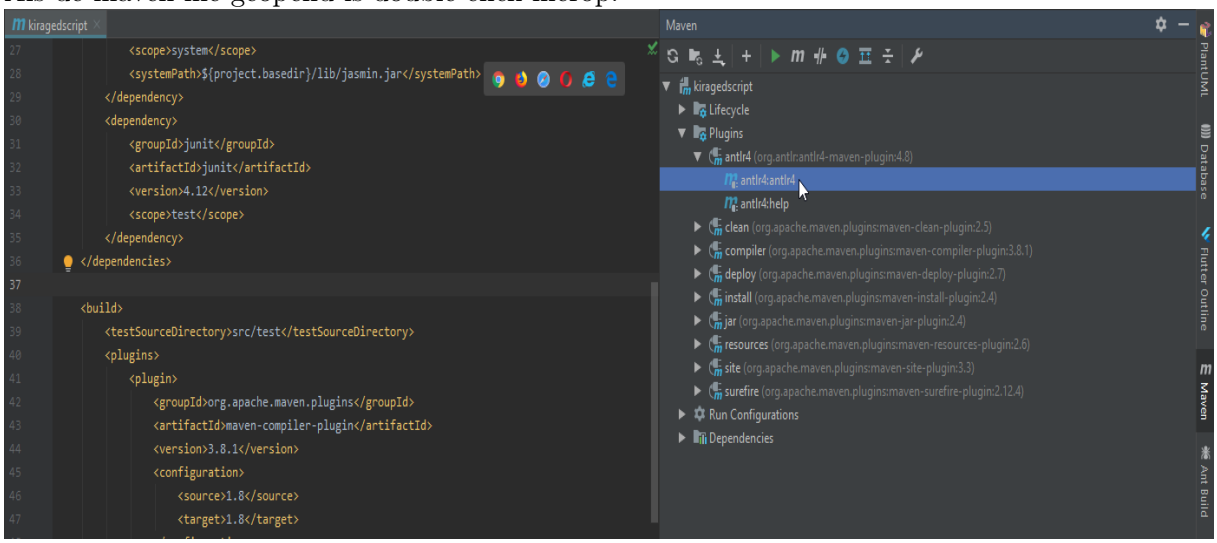


5. Om Compiler.java te runnen is een Program argument nodig. Om Script.txt te runnen in src/main/kiraged moet er src/main/kiraged/Script.txt in de Program arguments staan. De tests werken zonder iets ervoor te doen.

2.1 Antlr Generator

Dit zou niet nodig moeten zijn maar als het toch nodig is dan staat hier hoe het moet met maven.

1. Open de maven file in Intellij.
2. Als de maven file geopend is double click hierop:



3. De generated files zijn in `target/generated-sources/antlr4`. De grammar file is in `src/main/antlr4/com/kiraged/script/parser`.

3 Taal

Ik heb ervoor gekozen om een mix van Python en Java te gebruiken. Er is dus geen main function. Alle statements die buiten functions staan worden gecalled.

In de tests zouden alle mogelijkheden moeten staan.

3.1 Variables

Ik heb ervoor gekozen om 3 variables te implementeren, `int`, `double` en `boolean`. Het is ook mogelijk om de `var` keyword te gebruiken maar die moet wel initialized zijn.

3.1.1 Static Variables

Alle variables die buiten functions zijn gedeclared zijn static variables en kunnen dus in elke scope gebruikt worden. Local variables hebben wel prioriteit in functions.

Listing 1: Static variables.

```
//Static double
var x = 50.0;

//Static int, default value = 0
int y;
```

3.1.2 Local Variables

Alle variables die binnen functions zijn gedeclared zijn local variables. Local variables hebben prioriteit over static variables.

Listing 2: Local variables.

```
//Static double
var x = 50.0;

//Static int, default value = 0
int y;

void test() {
        int y = 10; //Local variable
        print(y); //Prints 10
        print(x); //Prints 50.0
}
```

3.1.3 Errors

Dit zijn alle errors die mogelijk zijn met variables.

Listing 3: Variable errors.

```
double x = 50.0;

int x = 50; //Duplicate variable

test y = 50.0; //No type found

double z = k + 10; //Variable not found

double q = true; //Double doesn't match boolean
```

3.2 Functions

Er zijn 4 soorten functions, void, int, double en boolean. Een void function kan niet een waarde returnen, de anderen wel.

Listing 4: Functions.

```
//Normal function with no return type
void test() {
        print();
}

//Returning no value does work in normal functions,
   print(100) will be ignored.
void test() {
        if (50 == 50) {
                print(50);
                return;
        }
        print(100);
}
```

```
//Returning int
int test() {
        return 50;
}

//Returning double
double test() {
        return 50.0;
}

//Returning boolean
boolean test() {
        return false;
}

//Function parameters
int test(int x, int y) {
        return x + y;
}
```

3.2.1 Errors

Dit zijn alle errors die mogelijk zijn met functions.

Listing 5: Function errors.

```
void test() {
        return;
        print(true); //Unreachable code
}

void test(int x, int x) { //Duplicate paramater

}

void test() {
        return 1; //Not allowed to return value
}

hello(); //Function not found
```

3.2.2 Core Functions

Er zijn ook een aantal core functions die gecalled kunnen worden.

Listing 6: Core Functions.

```
//Asking int user input
int x = askInt();

//Asking double user input
double x = askDouble();


//Asking boolean user input
boolean x = askBoolean();

//Printing new line
print();

//Printing int
print(50);

//Printing double
print(50.0);

//Printing boolean
print(false);

//Generating random int
int x = randomInt();

//Generating random int with range from 0 to 500
int x = randomInt(500);

//Generating random double
double x = randomDouble();

//Generating random boolean
boolean x = randomBoolean();
```

3.3 Expressions

Expressions kunnen gebruikt worden bij variables, functions en if/while statements.

3.3.1 Variables

Om meerdere expressions te gebruiken in een variable bijvoorbeeld een plus som moet de linker en rechter expression hetzelfde type hebben. Het is dus niet mogelijk om een double + boolean te doen.

3.3.1.1 Variable declaration

Dit zijn alle mogelijkheden als het komt op variable declaration:

Listing 7: Variable declaration expressions.

```
//Simple boolean assignment
boolean x = true;

//Comparison boolean assignment
boolean x = 50 == 50;

//Simple assignment
int x = 50;

//Simple +
int x = 50 + 50;

//Simple -
int x = 50 - 50;

//Simple *
int x = 50 * 50;

//Simple /
int x = 50 / 50;
```

```
//Using () to prioritise
int x = 50 * (50 + 50) + 50;

//Using variables
int y = 10;
int x = y * 50;

//All the above with int also work using double of
   course.
double x = 50.0;
//etc
```

3.3.2 Errors

Dit zijn alle errors die mogelijk zijn met functions.

3.3.3 Functions

Het is alleen mogelijk om functions als expressions te gebruiken als ze iets returnen.

Dit zijn alle mogelijkheden als het komt op functions:

Listing 8: Function expressions.

```
//Returning boolean
boolean x(boolean y) {
        return y;
}

boolean y(int value, int value2) {
    return x(value == value2);
}

//Test function
int test(int x) {
    return x + 10;
}
```

```
//Using test function and using +, all other
   expressions shown above will also work
int test2() {
        return test(50) + 50;
}


double test3() {
        return 10.0 * 500.0;
}
```

3.3.4 If/While Statements

Listing 9: If/While Statements and expressions.

```
//This will always print
if (true) {
        print(true);
}

//This will also always print since 50 is equal to 50
if (50 == 50) {
        print(true);
}

//This will also always print since 10 is equal to 10
if (50 == 20 || 10 == 10) {
        print(true);
}

//This will also always print since 50 is equal to 50
   and 20 is equal to 20
if (50 == 50 && 20 == 20) {
        print(true);
}
```

```
//This will also always print since 50 is greater
   than 10
if (50 >= 10 || 50 > 10) {
        print(true);
}

//This won't print since 50 is not lower than 10
if (50 <= 10 || 50 < 10) {
        print(true);
}

//This will also always print since 50 is not equal
   to 10
if (50 != 10) {
        print(true);
}

//This will also work
if (50 == 50 || (20 == 20 || 10 == 10)) {
        print(true);
}

//All of the above will also work for while loops
//This is an infinite loop
while(true) {
        print(true);
}
```

4 Design Choices

- **Waarom Maven?**
  Ik heb ervoor gekozen om Maven te gebruiken omdat iedereen die Intellij heeft dan ook de antlr files kan generaten.
- **Waarom geen seperate visitor voor type checking?**
  Een seperate visitor voor type checking leek mij niet echt logisch omdat het ook niet nodig is. Ik vind ook dat mijn design beter gebruik maakt van Object Oriented Programming.
- **Waarom een mix van Python en Java?**
  Java is mijn meest favoriete taal en ik wou een soort van Java scripting language maken.

5 Conclusie

Ik vind dat de taal goed en duidelijk werkt en ook dat de code goed gebruik maakt van Object Oriented Programming.