



іТМО

VASC

Валерий Кечин, Артем Унтила, Вячеслав Ковалеский

Задание

Исходные данные

- Объектно-ориентированный язык
- Инструменты: ANTLR
- Целевая платформа/язык – JVM/Jasmin assembler
- Язык(и) реализации: Java, Kotlin

Цели

- Анализ и адаптация спецификации
- Написание грамматики: ANTLR
- Синтаксический анализ
- Семантический анализ
- Кодогенерация: Jasmin assembler
- Запуск: .class файл(ы)

Добавлено:



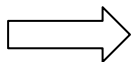
- null
- isnull()
- super
- println()
- Неинициализированные поля и локальные переменные:

```
var i: Integer
```

Изменено:

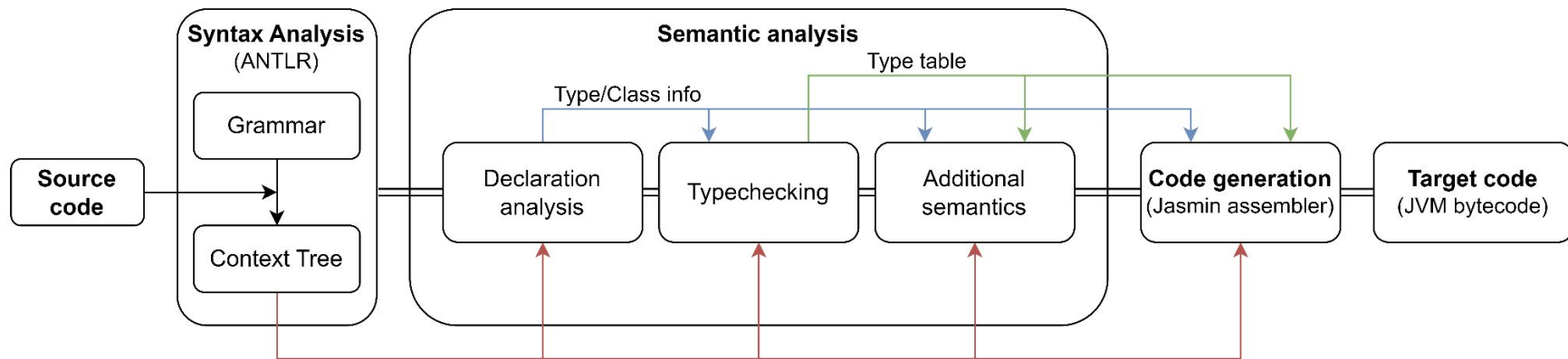
- Создание полей/локальных переменных и присваивание:

```
var i : Integer(1)  
i := 2
```



```
var i: Integer = Integer(1)  
i = 2
```

Общая архитектура проекта



Lexical and Syntax Analysis

<i>VascLexer.g4</i>	<i>VascParser.g4</i>
<pre>CLASS : 'class' ; IS : 'is' ; EXTENDS : 'extends' ; END : 'end' ; IDENTIFIER : [a-zA-Z_\$][a-zA-Z0-9_\$]* ; WS : [\t]+ -> channel(HIDDEN) ;</pre>	<pre>program : NL* (classDeclarations += classDeclaration semi?)* EOF ; classDeclaration : CLASS NL* name=identifier NL* (EXTENDS NL* parentName=identifier)? NL* IS NL* classBody NL* END ; classBody : (memberDeclarations += memberDeclaration semi)* ; semi : NL+ ;</pre>

IntelliJ Plugin

```
17 class RandomRandu is
18
19     var randMax: Integer = 2147483648
20     var seed: Integer
21     var state: Integer
22
23     this(_seed: Integer) is
24         seed = _seed
25         state = 1
26     end
27
28     this() is
29         seed = 0
30         state = 1
31     end
32
33     method nextInt(): Integer is
34         state = state.mul(65539).plus(seed).rem(randMax)
35         return state
36     end
37
38 end
```

```
method main() is
```

```
    var i: Integer = 0
```

```
    var rand = RandomRandu()
```

```
    while i.les
```

mismatched input '=' expecting ':'

```
        var r
```

```
        println("$res")
```

```
        i = i.plus(1)
```

```
method main() is
```

```
    var i: Integer = 0
```

```
    var rand: = RandomRandu()
```

```
    while i.les
```

extraneous input '=' expecting {'Array', 'List', 'Boolean', 'Real', 'Inte

```
        var res
```

```
        println("$res")
```

```
        i = i.plus(1)
```

```
    end
```

```
end
```

Declaration analysis: представление типа


```
interface VascType {  
    val name: String  
    val parent: VascType?  
  
    val declaredFields: Collection<VascVariable>  
    val declaredConstructors: Collection<VascConstructor>  
    val declaredMethods: Collection<VascMethod>  
  
    val fields: Collection<VascVariable>  
    val methods: Collection<VascMethod>  
  
    fun getDeclaredField(name: String): VascVariable?  
    fun getDeclaredConstructor(parameterTypes: List<VascType>): VascConstructor?  
    fun getDeclaredMethod(name: String, parameterTypes: List<VascType>): VascMethod?  
  
    fun getField(name: String): VascVariable?  
    fun getMethod(name: String, parameterTypes: List<VascType>): VascMethod?  
  
    fun isAssignableFrom(subtype: VascType): Boolean  
}
```

Declaration analysis: пример

```
class A is
  var a: Integer
  var b: Boolean

  this(_a: Integer, _b: Boolean) is
    a = _a
    b = _b
  end

  method getA(): Integer is
    if b then return a
    else return 0
  end
end
```



```
f parent = null
v f declaredFields = {LinkedHashSet@1233} size = 2
  v 0 = {VascVariable@1259} a: Integer
    > f name = "a"
    > f type = {VascInteger@1286} Integer
    > 1 = {VascVariable@1260} b: Boolean
v f declaredConstructors = {LinkedHashSet@1234} size = 1
  v 0 = {VascConstructor@1289} this(_a: Integer, _b: Boolean)
    v f parameters = {ArrayList@1293} size = 2
      > 0 = {VascVariable@1296} _a: Integer
      > 1 = {VascVariable@1297} _b: Boolean
      > f parameterTypes = {ArrayList@1294} size = 2
v f declaredMethods = {LinkedHashSet@1235} size = 1
  v 0 = {VascMethod@1301} getA(): Integer
    > f name = "getA"
    > f returnType = {VascInteger@1286} Integer
    > f parameters = {EmptyList@1315} size = 0
    > f parameterTypes = {ArrayList@1316} size = 0
> f name = "A"
```


Declaration analysis: ошибки

- Повторяющееся имя класса



1	<code>class A is</code>	<code>Class A is already declared in this scope at [4:6] class A is ^ end</code>
2	<code>end</code>	
3		
4	<code>class A is</code>	
5	<code>end</code>	

- Неизвестный тип/класс

1	<code>class A extends B is</code>	<code>Unknown type: B at [1:16] class A extends B is ^ end</code>
2	<code>end</code>	

- 

```
Cyclic inheritance involving 'B'
at [4:16]
class B extends A is
    ^
end
```

```
Duplicate parameter name 'a'
at [2:14]
      method foo(a: Integer, a: Real) is
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      end
```

Declaration analysis: ошибки

- Повторяющиеся члены класса: поля, методы, конструкторы



```
1 class A is
2     var a: Integer
3     var a: Boolean
4 end
```

```
Duplicate field 'A.a'
at [3:4]
    var a: Integer
    var a: Boolean
^^^^^^^^^^^^^^^^
```

- Несовместимость возвращаемого типа переопределенного метода

```
1 class A is
2     method foo() is
3     end
4 end
5
6 class B extends A is
7     method foo(): Integer is
8     end
9 end
```

```
Return type of 'B.foo(): Integer' is
incompatible with parent
at [7:4]
    method foo(): Integer is
    ^^^^^^^
    end
```

Typechecking: область видимости

```
class Scope(  
    private val vars: MutableMap<String, VascType>,  
    private val parent: Scope? = null,  
) {  
  
    fun find(name: String): VascType? {  
        return vars[name] ?: parent?.find(name)  
    }  
  
    fun add(name: String, t: VascType) {  
        vars[name] = t  
    }  
  
    fun enclosed(  
        vars: MutableMap<String, VascType> = mutableMapOf()  
    ) = Scope(vars, this)  
}
```



```
override fun visitIfStatement(ctx: IfStatementContext) {  
    val expectT = VascBoolean  
    tryWithContext(ctx.expression()) {  
        val actualT = it.typeOrThrow()  
        if (expectT != actualT) {  
            throw UnexpectedTypeException(expectT, actualT, it)  
        }  
    }  
    ctx.thenBody.accept(copy(scope.enclosed()))  
    ctx.elseBody?.accept(copy(scope.enclosed()))  
}
```



1	class Main is	expected type:
2	this() is	Boolean
3	if 42 then	but got:
4	end	Integer
5	end	at [3:11]
6	end	if 42 then
7	end	^^
		end

Typeschecking: ошибки

- Неизвестная переменная

1	<code>class Main is</code>	<code>unknown variable 'a'</code> <code>at [6:12]</code> <code>a</code> <code>^</code>
2	<code> this() is</code>	
3	<code> if true then</code>	
4	<code> var a : Integer</code>	
5	<code> else</code>	
6	<code> a</code>	
7	<code> end</code>	
8	<code> end</code>	
9	<code>end</code>	

- Присваивание переменной значения другого типа

```
1  class Main is
2      this() is
3          var a : Integer = 0
4          a = false
5      end
6  end
```

```
expected type:
  Integer
but got:
  Boolean
at [4:8]
    var a : Integer = 0
    a = false
    ^^^^^^^^^
```

- Использование несуществующего значения после вызова метода



```
1 class Main is
2     this() is
3         var a : Array[Integer] =
4         Array[Integer](1)
5         a.set(1, 2).set(2, 3)
6     end
7 end
```

```
method 'Array[Integer].set' does not
return a value
at [4:9]
    a.set(1, 2).set(2, 3)
    ^^^^^^^^^^^
```


- Отсутствие конструктора/метода с типами переданных аргументов



```
1 class Main is
2   this() is
3     A(42)
4   end
5 end
6
7 class A is
8   this(b: Boolean) is
9   end
10 end
```

```
constructor not found 'A(Integer)'
at [3:8]
    A(42)
    ^^^^^
```

- И другие ошибки при проверках:
 - наличия поля с указанным именем
 - наличия родителя при вызове родительских методов/конструкторов
 - ...

Дополнительные семантические проверки реализованы в отдельных обходах:



- Exhaustiveness Check:
 - Недостижимый код
 - Неполный return
- Constructor Check:
 - Циклический вызов конструкторов
 - Отсутствие вызова super (или др. конструктора с super) у наследников
 - Вызов super или this не в первую очередь

- Недостижимый код



```
..  ...
9   if a then
10      return a
11      a = false
12  end
```

Unreachable code
at [11:8]

```
...
if a then
  return a
  a = false
^^^^^^^^^^
```

- Неполный return

```
..  ...
5   method foo(a: Boolean) : Boolean is
6     if a then
7       return a
8     end
9   end
```

Non exhaustive return
at [5:8]

```
method foo(a: Boolean) : Boolean is
  ...
```

Additional semantics: Constructors

- Циклический вызов конструкторов



```
1  class Main is
2      this(a: Integer, b: Integer) is
3          this(a)
4      end
5
6      this(a: Integer) is
7          this(a, a)
8      end
9  end
```

```
Cyclic constructor usage
at [7:7]
  this(a, a)
  ^^^^^^^^^^^
```

Additional semantics: Constructors

- Отсутствие вызова super

```
1 class Main is
2     this(a: Integer) is
3     end
4 end
5
6 class A extends Main is
7     this(a: Integer) is
8     end
9 end
```

Default constructor not exists. Use super
call.
at [7:4]
this(a: Integer) is
^^^^^^

Additional semantics: Constructors

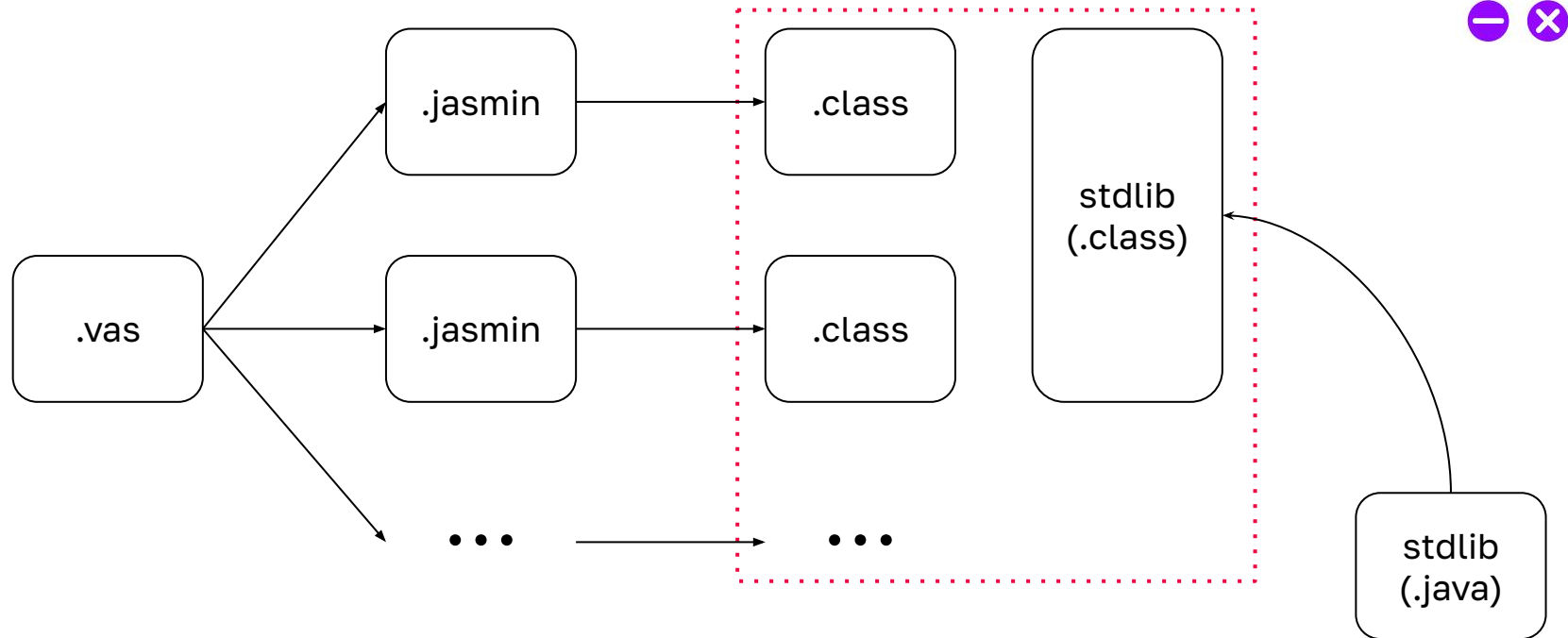
- Вызов super или this не в первую очередь

```
1  class Main is
2      this(a: Integer) is
3      end
4  end
5
6  class A extends Main is
7
8      var b: Integer
9
10     this(a: Integer) is
11         b = a
12         super(a)
13     end
14 end
```

Super Constructor must be called first in
constructor body
at [12:8]

```
    b = a
    super(a)
    ^^^^^^^
```

Code generation



Code generation: структура файла

```
.class public <имя класса>
.super <имя родителя>
// поля //
.field public <имя поля> <тип поля>
// точки входа в программу //
.method public static main([Ljava/lang/String;)V
.method public static main_ctor_<N>([Ljava/lang/Object;)Z
// конструкторы //
.method public <init>(<типы аргументов конструктора>)V

// методы //
.method public <имя>(<типы аргументов метода>) <возвращаемый тип>
    .limit stack <максимальный размер стека>
    .limit locals <максимальное число переменных>
    <байткод*>
.end method
```



Code generation: пример

```
.method public <init>(Lcom/vasc/Boolean;Lcom/vasc/Boolean;)V
```

```
.limit stack 32
```

```
.limit locals 3
```

```
aload_0      ;/// load this
```

```
invokespecial com/vasc/Adder/<init>()V      ;/// call default parent constructor
```

```
.line 86
```

```
aload 1      ;/// read local a: Boolean
```

```
aload 2      ;/// read local b: Boolean
```

```
invokevirtual com/vasc/Boolean/xor(Lcom/vasc/Boolean;)Lcom/vasc/Boolean;      ;/// call Boolean.xor(p: Boolean): Boolean
```

```
aload_0      ;/// load this
```

```
swap
```

```
putfield com/vasc/HalfAdder/s Lcom/vasc/Boolean;      ;/// assign field HalfAdder.s: Boolean
```

```
.line 87
```

```
aload 1      ;/// read local a: Boolean
```

```
aload 2      ;/// read local b: Boolean
```

```
invokevirtual com/vasc/Boolean/and(Lcom/vasc/Boolean;)Lcom/vasc/Boolean;      ;/// call Boolean.and(p: Boolean): Boolean
```

```
aload_0      ;/// load this
```

```
swap
```

```
putfield com/vasc/HalfAdder/c Lcom/vasc/Boolean;      ;/// assign field HalfAdder.c: Boolean
```

```
return
```

```
.end method
```

```
class Adder is
    var s : Boolean // sum
    var c : Boolean // carry
end
```

```
class HalfAdder extends Adder is
    this(a : Boolean, b : Boolean) is
        s = a.xor(b)
        c = a.and(b)
    end
end
```

Code generation: байткод*

```
override fun visitIfStatement(ctx: IfStatementContext) {
    withLineInfo(ctx.start.line) {
        val endLabel = "If_End_${ctx.pos()}"
        val elseLabel = "If_Else_${ctx.pos()}"
        generateBooleanExpression(ctx.condition)
        if (ctx.elseBody != null) {
            appendLine("ifeq $elseLabel")
            ctx.thenBody.accept(this)
            appendLine("goto $endLabel")
            appendLine("$elseLabel:")
            ctx.elseBody.accept(this)
            appendLine("$endLabel:")
        } else {
            appendLine("ifeq $endLabel")
            ctx.thenBody.accept(this)
            appendLine("$endLabel:")
        }
    }
}
```

```
override fun visitIntegerLiteral(ctx:
IntegerLiteralContext) {
    appendLine("new $integerClass")
    appendLine("dup")
    appendLine("ldc2_w ${ctx.text}")
    appendLine("invokespecial $integerClass/<init>(J)V")
}
```

```
override fun visitExpressionStatement(ctx: ExpressionStatementContext) {
    withLineInfo(ctx.start.line) {
        ctx.expression().accept(this)
        if (typeTable[ctx.expression()] != VascVoid) {
            appendLine("pop", "discard result")
        }
    }
}
```

Code generation: stdlib

```
public class Integer {
    public final long value;
    public Integer(long value) {
        this.value = value;
    }
    public Integer(com.vasc.Integer value) {
        this.value = value.value;
    }
    public Integer(com.vasc.Real value) {
        this.value = (long) value.value;
    }
    public com.vasc.Real toReal() {
        return new Real(this);
    }
    public com.vasc.Boolean toBoolean() {
        return new Boolean(value != 0);
    }
    public com.vasc.Integer unaryMinus() {
        return new Integer(-value);
    }
    public com.vasc.Integer plus(com.vasc.Integer other) {
        return new Integer(value + other.value);
    } ...
}
```

```
public class Array<T> {
    private final T[] value;
    public Array(com.vasc.Integer length) {
        this.value = (T[]) new Object[(int) length.value];
    }
    public com.vasc.List<T> toList() {
        return new List<>(Arrays.asList(value));
    }
    public com.vasc.Integer length() {
        return new Integer(value.length);
    }
    public T get(com.vasc.Integer index) {
        return value[(int) index.value];
    }
    public void set(com.vasc.Integer index, T v) {
        value[(int) index.value] = v;
    }
    @Override
    public String toString() {
        return Arrays.toString(value);
    }
}
```



Code generation: main

```
public static boolean main_ctor_2(Object[] var0) {
```

```
    Main var10000;
```

```
    Integer var10001;
```

```
    Boolean var10002;
```

```
    try {
```

```
        var10000 = new Main;
```

```
        var10001 = (Integer)var0[0];
```

```
        var10002 = (Boolean)var0[1];
```

```
    } catch (ClassCastException var1) {
```

```
        return false;
```

```
    }
```

```
    var10000.<init>(var10001, var10002);
```

```
    return true;
```

```
}
```

```
public static void main(String[] var0) {
```

```
    Object[] var1 = ArgumentParser.parse(var0);
```

```
    if (var1.length != 0 || !main_ctor_0(var1)) {
```

```
        if (var1.length != 1 || !main_ctor_1(var1)) {
```

```
            if (var1.length != 2 || !main_ctor_2(var1)) {
```

```
                System.out.println("error: ...");
```

```
            }
```

```
        }
```

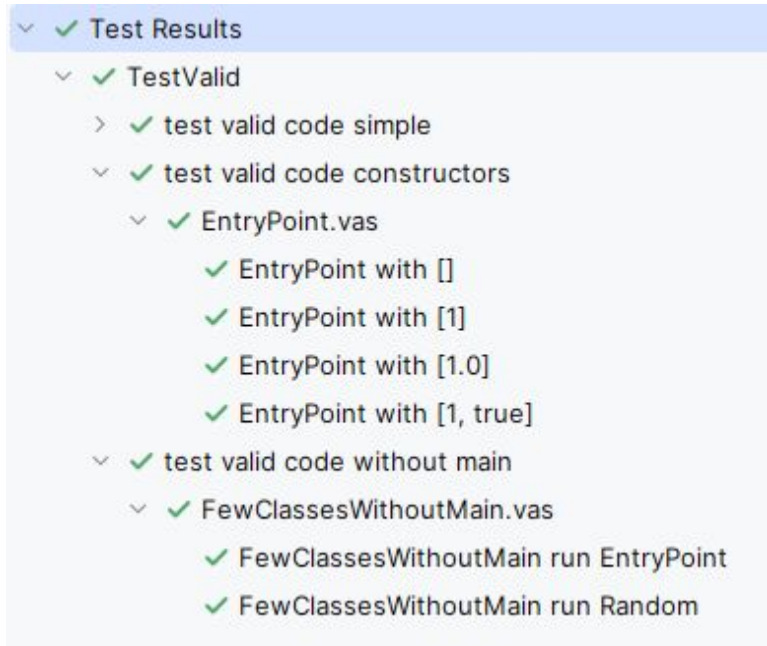
```
    }
```

```
}
```

Код из декомпилированного .class файла



- Тесты для корректных программ - сравнение с ожидаемым выводом
 - Вызов из Main без аргументов
 - Вызов с из Main аргументами
 - Вызов не из Main
- Тесты для некорректных программ - ожидаемое исключение
 - Дополнительные проверки семантики
 - Typechecking



Ссылка на репозиторий и демо

<https://github.com/kechinvv/VASC>

