Centro Universitário de Belo Horizonte – UniBH
Curso: Ciência da Computação
Disciplina: Compiladores
Professor: Gustavo Alves Fernandes

Linguagem Javazim (Corrigida) com Ações Semânticas

Programa → Classe $ 1

Classe → "public" "class" ID {TS.setTipo(ID.lexval, vazio)}
"{" ListaMetodo Main "}" 2

DeclaracaoVar → Tipo ID { TS.setTipo(id, Tipo.t) } ";" 3

ListaMetodo → ListaMetodo' 4

ListaMetodo' → Metodo ListaMetodo' 5 | ε 6

Metodo → Tipo ID {TS.setTipo(ID.lexval, Tipo.t)}
"(" RegexListaParam ")" "{" RegexDeclaraVar ListaCmd Retorno "}"
{se Retorno.t != Tipo.t: **sinalizar erro para tipo de retorno incompatível**}7

RegexListaParam → ListaParam 8 | ε 9

RegexDeclaraVar → DeclaracaoVar RegexDeclaraVar 10 | ε 11

ListaParam → Param ListaParam' 12

ListaParam' → ", " ListaParam 13 | ε 14

Param → Tipo ID {TS.setTipo(ID.lexval, Tipo.t)} 15

Retorno → "return" Expressao";" {Retorno.t = Exp.t} 16 | ε {Retorno.t = vazio} 17

Main → "public" "static" "void" "main" "(" ")" "{" RegexDeclaraVar ListaCmd "}" 18

Tipo → "boolean"{Tipo.t = logico} 19 | "int" {Tipo.t = numerico} 20
| "string" {Tipo.t = literal} 21 | "float" {Tipo.t = numerico} 22
| "void" {Tipo.t = vazio} 23

ListaCmd → ListaCmd' 24

ListaCmd' → Cmd ListaCmd' 25 | ε 26

Cmd → CmdIF 27 | CmdWhile 28 | CmdPrint 29 | CmdPrintln 30
| ID Cmd'
{se Cmd'.t != vazio && TS.getTipo(ID.lexval) != Cmd'.t:
sinalizar erro de atribuição incompatível} 31

Cmd' → CmdAtrib {Cmd'.t = CmdAtrib.t} 32 | CmdMetodo {Cmd'.t = vazio} 33

CmdIF → "if" "(" Expressao ")"
{se Exp.t != logico: **sinalizar erro**} "{" Cmd "}" CmdIF' 34

CmdIF' → "else" "{" Cmd "}" 35 | ε 36

CmdWhile → "while" "(" Expressao ")"
{se Exp.t != logico: **sinalizar erro**} "{" Cmd "}" 37

CmdPrint → "print" "(" Expressao ")" ";" 38

CmdPrintln → "println" "(" Expressao ")" ";" 39

CmdAtrib → "=" Expressao ";" {CmdAtrib.t = Exp.t} 40

CmdMetodo → "(" RegexExp4 ")" ";" 41

Expressao → Exp1 Exp'
{se Exp'.t == vazio: Exp.t = Exp1.t;
senao se Exp'.t == Exp1.t && Exp'.t == logico: Exp.t = logico;
senao: Exp.t = **tipo_erro**} 42

Exp' → "&&" Exp1 Exp' 43 | "||" Exp1 Exp' 44
// ação semantica vale para as regras 43 e 44
{se Exp'Filho.t == vazio && Exp1.t == logico: Exp'.t = logico
senao se Exp'Filho.t == Exp1.t && Exp1.t == logico: Exp1'.t = logico;
senao: Exp1'.t = **tipo_erro**}
| ε 45 {Exp'.t = vazio}

Exp1 → Exp2 Exp1’
{se Exp1’.t == vazio: Exp1.t = Exp2.t;
senao se Exp1’.t == Exp2.t && Exp1’.t == numerico: Exp1.t = logico;
senao: Exp1.t = **tipo_erro**} 46

Exp1’ → "<" Exp2 Exp1’ 47 | "<=" Exp2 Exp1’ 48 | ">" Exp2 Exp1’ 49
| ">=" Exp2 Exp1’ 50 | "==" Exp2 Exp1’ 51 | "!=" Exp2 Exp1’ 52
// ação semantica vale para as regras 47, 48,..., 52
{se Exp1’Filho.t == vazio && Exp2.t == numerico: Exp1’.t = numerico;
senao se Exp1’Filho.t == Exp2.t && Exp2.t == numerico: Exp1’.t = numerico;
senao Exp1’.t = **tipo_erro**}
| ε 53 {Exp1’.t = vazio}

Exp2 → Exp3 Exp2’ 54
{se Exp2’.t == vazio: Exp2.t = Exp3.t;
senao se Exp2’.t == Exp3.t && Exp2’.t == numerico: Exp2.t = numerico;
senao: Exp2.t = **tipo_erro**}

Exp2’ → + Exp3 Exp2’ 55 | - Exp3 Exp2’ 56
// ação semantica vale para as regras 55 e 56
{se Exp2’Filho.t == vazio && Exp3.t == numerico: Exp2’.t = numerico;
senao se Exp2’Filho.t == Exp3.t && Exp3.t == numerico: Exp2’.t = numerico;
senao Exp3’.t = **tipo_erro**}
| ε {Exp2’.t = vazio} 57

Exp3 → Exp4 Exp3’ 58
{se Exp3’.t == vazio: Exp3.t = Exp4.t;
senao se Exp3’.t == Exp4.t && Exp3’.t == numerico: Exp3.t = numerico;
senao: Exp3.t = **tipo_erro**}

Exp3’ → * Exp4 Exp3’ 59 | / Exp4 Exp3’ 60
// ação semantica vale para as regras 59 e 60
{se Exp3’Filho.t == vazio && Exp4.t == numerico: Exp3’.t = numerico
senao se Exp3’Filho.t == Exp4.t && Exp4.t == numerico: Exp3’.t = numerico;
senao Exp3’.t = **tipo_erro**}
| ε {Exp3’.t = vazio} 61

Exp4 → ID Exp4’ {Exp4.t = TS.getTipo(ID.lexval);
se Exp4.t == *null*: **sinalizar erro para ID não declarado**} 62
| ConstInteira {Exp4.t = ~~numerico~~} 63 | ConstReal {Exp4.t = numerico} 64
| ConstString {Exp4.t = literal} 65
| "true" {Exp4.t = logico} 66 | "false" {Exp4.t = logico} 67
| OpUnario Expressao
{se Exp.t == OpUnario.t && OpUnario.t == numerico: Exp4.t = numerico;
senão se Exp.t == OpUnario.t && OpUnario.t == logico: Exp4.t = logico;
senão Exp4.t = **tipo_erro**} 68
| "(" Expressao")" {Exp4.t = Exp.t} 69

Exp4’ → "(" RegexExp4 ")" 70 | ε 71

RegexExp4 → Expressao RegexExp4’ 72 | ε 73

RegexExp4’ → "," Expressao RegexExp4’ 74 | ε 75

OpUnario → "-" {OpUnario.t = numerico} 76 | "!" {OpUnario.t = logico} 77