Práctica 4

Procesadores del lenguaje

Ángel Luis Ortiz Folgado 50774922-F
Oscar Eduardo Pérez la Madrid 54268894-A
Esteban Vargas Rastrollo 09207146-Q

Gramatica Atribuida

Programa ::= PROGRAM SeccionTipos SeccionVariables SeccionSubprogramas IBloque;
Programa.a = prog(SeccionTipos.a,SeccionVariables.a,SeccionSubprogramas.a,IBloque.a)

SeccionTipos ::= TYPES DecTipos
SeccionTipos.a= crearSeccionTipos(DecTipos.a)
SeccionTipos ::=ε
SeccionTipos .a = crearSeccionTiposVacia()

SeccionVariables ::= VARIABLES DecVariables
SeccionVariables.a = crearSeccionVariables(DecVariables.a)

SeccionVariables::= ε
SeccionVariables.a= crearSeccionVarVacia()

SeccionSubprogramas ::= SUBPROGRAMS DecSubprogramas
SeccionSubprogramas.a= crearSeccionsubprogramas (DecSubprogramas .a)

SeccionSubprogramas ::=ε
SeccionSubprogramas.a= crearSeccionSubVacia()

DecTipos ::= DecTipos PYCOMA DecTipo
DecTipos.a= hazListaDectipos(DecTipos.a,DecTipo.a)

DecTipos ::= DecTipo
DecTipos.a== hazListaDectipos (ListaTipoVacia(),decTipo.a)

DecTipo ::= ID DOSPUNTOS Tipo;
DecTipo.a= crearDecTipo(Tipo.a,ID.lex)

Tipo ::= INT
Tipo.a= crearInteger()

Tipo ::=DOUBLE
Tipo.a= crearDouble()

Tipo ::= BOOLEAN
Tipo.a=crearBoolean()

Tipo ::= ID
Tipo.a=crearId(ID.lex)

Tipo ::= ARRAY Tipo CAP NUMERO_NATURAL CCIERRE
Tipo.a = crearArray(Tipo.a, NUMERO_NATURAL.a)

Tipo ::= STRUCT LLAVEAP Campos LLAVECIERRE
Tipo.a=crearReg(Campos.a)

Tipo ::= POINTER Tipo;
Tipo.a= crearPointer(Tipo1.a)

Campos ::= Campos PYCOMA Campo

Campos.a= haz Lista Campos (Campos1.a,Campo.a)

Campos::= Campo
Campos.a= = haz Lista Campos (ListaCamposVacia(),Campo.a)

Campo ::= ID DOSPUNTOS Tipo;
Campo.a= hazCampo(ID.lex,Tipo.a)

DecVariables ::= DecVariables PYCOMA DecVariable
DecVarianle.a = HazListaDecVariables(Decvariables.a,Decvariable.a)

DecVariables ::= DecVariable
Decvariables.a== HazListaDecVariables (ListaDecVariablesvacia(),DecVariable.a)

DecVariable ::= ID DOSPUNTOS Tipo
DecVariable.a=crearDecVariable(ID.lex,Tipo.a)

DecSubprogramas ::= DecSubprogramas PYCOMA DecSubprograma
DecSubprogramas.a= HazListaProcs(DecSubprogramas.a, DecSubprograma.a)

DecSubprogramas ::=DecSubprograma
DecSubprogramas.a= HazListaProcs(ListaProcsvacia(), DecSubprograma.a)

DecSubprograma ::= SUBPROGRAM ID Parametros SeccionTipos SeccionVariables
SeccionSubprogramas IBloque;

DecSubprograma.a=
CrearDecSubprograma(ID.lex,parametros.a,SeccionTipos.a,SeccionVariables.a,SeccionSubprogramas.a,Ibloque.a)

Parametros ::= PAP ListaParametros PCIERRE
Parametros.a = ListaParametros.a

Parametros ::= PAP PCIERRE
Parametros.a=ListaParametrosVacia()

ListaParametros ::= ListaParametros COMA Parametro
ListaParametros=HazListaParametros(ListaParametros.a,Parametro.a)

ListaParametros ::= Parametro
ListaParametros.a= HazListaParametros (ListaParametrosVacia(),parámetro.a)

Parametro ::= ID DOSPUNTOS Tipo
Parametro.a=HazParametroValor(ID.lex,Tipo.a)

Parametro ::= AMP ID DOSPUNTOS Tipo ;

Parametro.a= HazParametroVariable(ID.lex,Tipo.a)

Instruccion ::= IAsig
Instrucción.a=IAsig.a

Instruccion ::= IBloque
Instrucción.a=IBloque.a

Instruccion ::= ICond
Instrucción.a=Icond.a

Instruccion ::= IBucle
Instrucción.a=IBucle.a

Instruccion ::= ILlamada
Instrucción.a=ILlamada.a

Instruccion ::= IWrite
Instrucción.a=IWrite.a

Instruccion ::= New
Instrucción.a=INew.a

Instrucción::= IDelete
Instrucción.a=IDelete.a

IAsig ::= Designador ASIG Exp0
IAsig.a= crearAsignacion(Designador.a,Exp0.a)

IBloque ::= LLAVEAP Instrucciones LLAVECIERRE
IBloque.a=crearBloque(Instrucciones.a)

IBloque ::= LLAVEAP LLAVECIERRE;
IBloque.a = creaBloqueVacio()

Instrucciones ::= Instrucciones PYCOMA Instruccion
Instrucciones.a= hazListaIntruciones(Instruciones.a,Instruccion.a)

Instrucciones::= Instruccion
Instruciones.a= hazListaIntruciones (ListaInstVacia(),Instrucción.a)

ICond ::= IF Casos ENDIF ;
ICond.a= crearIf(Casos.a)

Casos ::= Casos CAP CCIERRE Caso
Casos.a= HazListacasos(Casos1.a,Caso.a)

Casos ::= Caso
Casos.a= HazListacasos (ListaCasosVacia(),caso.a)

Caso ::= Exp0 DOSPUNTOS IBloque;

Caso.a= HazCaso(Exp0.a,IBloque.a)

IBucle ::= DO Casos ENDDO
IBucle.a=crearBucle(Casos.a)

ILlamada ::= ID Argumentos;
iLlamada = crearLlamada(ID.lex,Argumentos.a)

IRead ::= READ Designador;
IRead =crear Lee(Designador.a)

IWrite ::= WRITE Exp0;
IWRITE = crearEscribe(Exp0.a)

INew ::= NEW Designador;
INEW = crearNuevo(Designador.a)

IDelete ::= DELETE Designador;
IDelete.a = crearDelete(Designador.a)

Argumentos ::= PAP ListaArgumentos PCIERRE
Argumentos.a = crearListaArg(ListaArgumentos.a)

Argumentos ::= PAP PCIERRE;
Argumentos.a = crearListaArgVacia()

ListaArgumentos ::= ListaArgumentos COMA Exp0
ListaArgumentos1.a = HazListaArgumentos(ListaArgumentos.a,Exp0.a)

ListaArgumentos ::= Exp0;
Lista Argumentos.a = HazListaArgumentos(ListaArgumentosVacia(), Exp0.a)

Designador ::= ID
Designador.a = crearDesignador(ID.lex)

Designador ::= Designador, CAP Exp0 Cierre
Desginador.a = crearDesignador(Designado1r.a, Exp0.a)

Designador ::= Designador PUNTO ID ;
Designador.a = crearDesignador(Designador1.a, ID.lex)

Designador ::= Designador FLECHA;
Designador.a = crearDesignador(Designador1.a)

Exp0 ::= Exp1 OpComp Exp1
Exp01.a = crearExpBinariaComp(OpComp, Exp1.a, Exp1.a)

Exp0 ::= Exp1
Exp0.a = Exp1.a
Exp1 ::= Exp1 OpAditivo Exp2
Exp10 = crearExpBinaria Adi(OpAditivo, Exp1.a , Exp2.a)
Exp1 ::= Exp2
Exp1.a = Exp2.a

Exp2 ::= Exp2 OpMultiplicativo Exp3
Exp2.a = crearExpBinaria Mult(OpMultiplicativo, Exp21.a , Exp3.a)

Exp2::= Exp3
Exp2.a = Exp3.a

Exp3 ::= OpUnario Exp3
Exp3.a = crearExpUnaria(OpUnario, Exp31.a)
Exp3::= Exp4
Exp3.a = Exp4.a

Exp4 ::= TRUE | FALSE | NUMERO_NATURAL | NUMERO_REAL | Designador | PAP Exp0 PCIERRE;
Exp4.a = bool(TRUE.lex)
Exp4.a = bool(FALSE.lex)
Exp4.a =entero(NUMERO_NATURAL.lex)
Exp4.a =double(NUMERO_REAL.lex)
Exp4.a=Designador.a
Exp4.a= Exp0.a

OpComp ::= IGUAL | DISTINTO | MAYOR | MAYOROIGUAL | MENOR | MENOROIGUAL;
OpComp.op = crearOpComp()
OpComp.op = crearOpDistinto()
OpComp.op = crearOpMayor()
OpComp.op = crearOprMayorOIgual()
OpComp.op = crearOpMenor()
OpComp.op = CrearOpMenorOIgual()

OpAditivo ::= MAS | MENOS | OR;
OpAditivo.op = crearOpMas()
OpAditivo.op = crearOpMenos()
OpAditivo.op = crearOpOr()

OpMultiplicativo ::= POR | DIV | MOD | AND;
OpMultiplicativo.op = crearOpPor()
OpMultiplicativo.op = crearOpDiv()
OpMultiplicativo.op = crearOpMod()
OpMultiplicativo.op = crearOpAnd()

OpUnario ::= MENOS | NOT | TOINT | TODOUBLE;
OpUnario.op = crearOpMenosUna()
OpUnario.op = crearOpNot()
OpUnario.op = crearOpToInt()
OpUnario.op = crearOpToDouble()

Diagrama constructoras

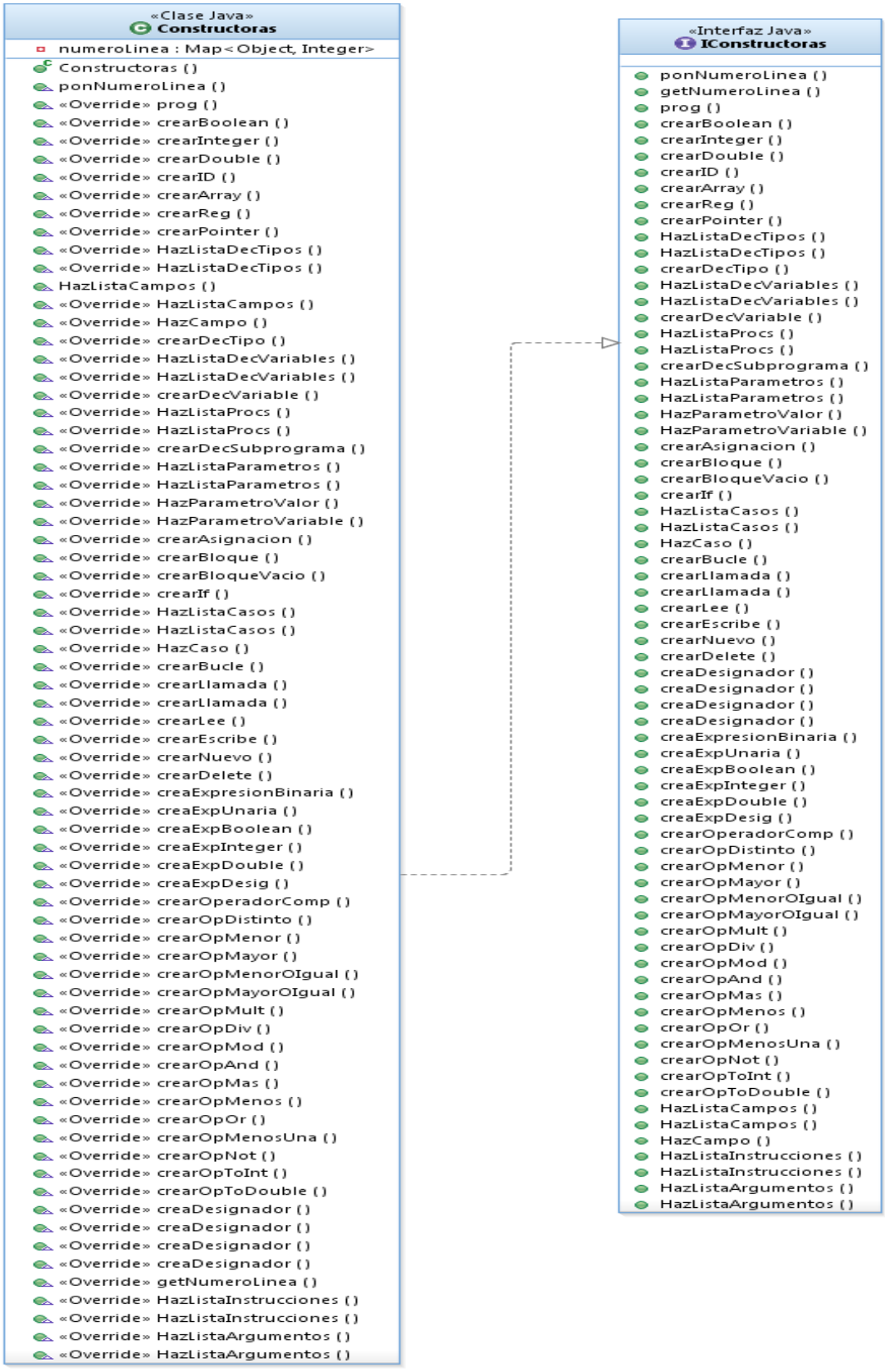


Diagrama expresiones

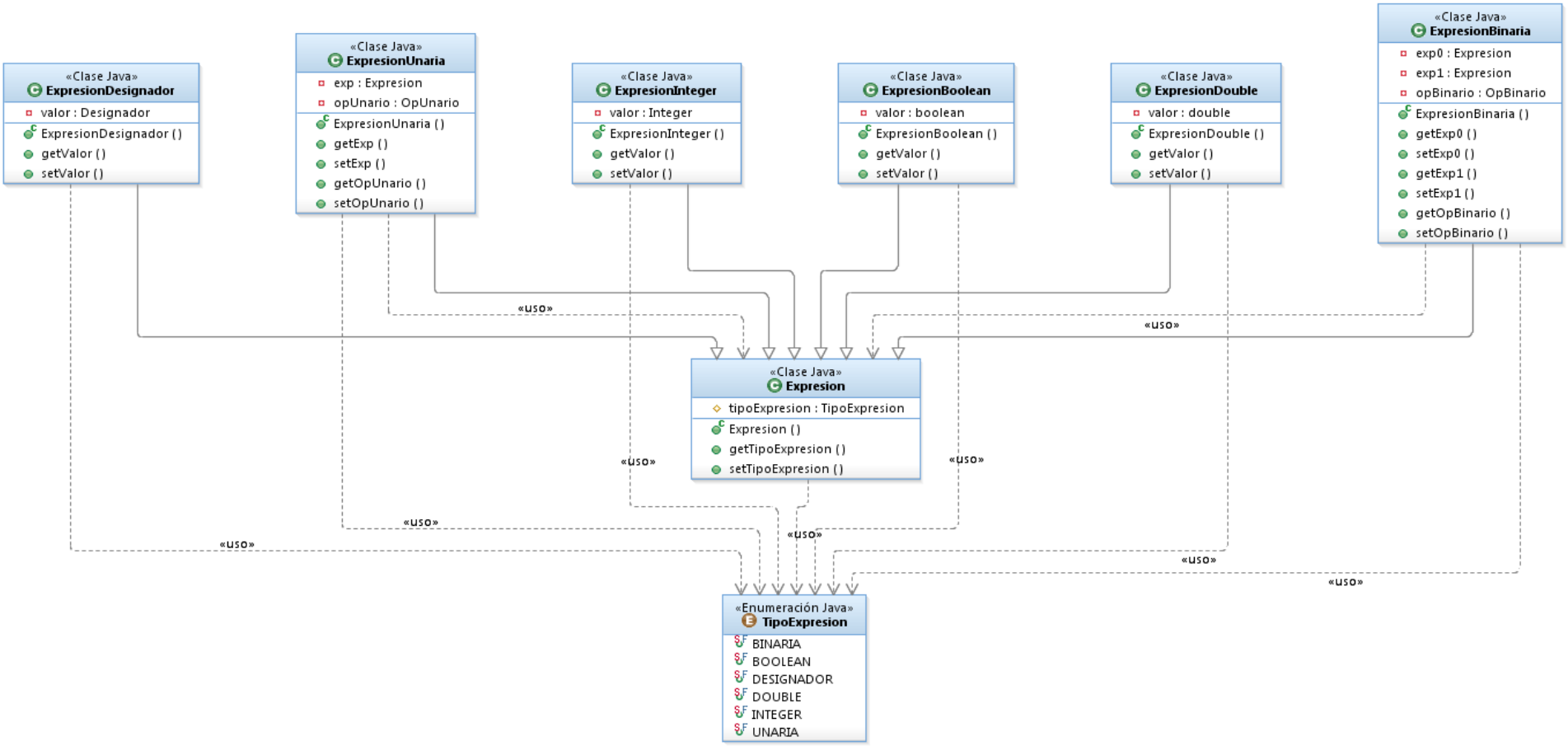


Diagrama instrucciones

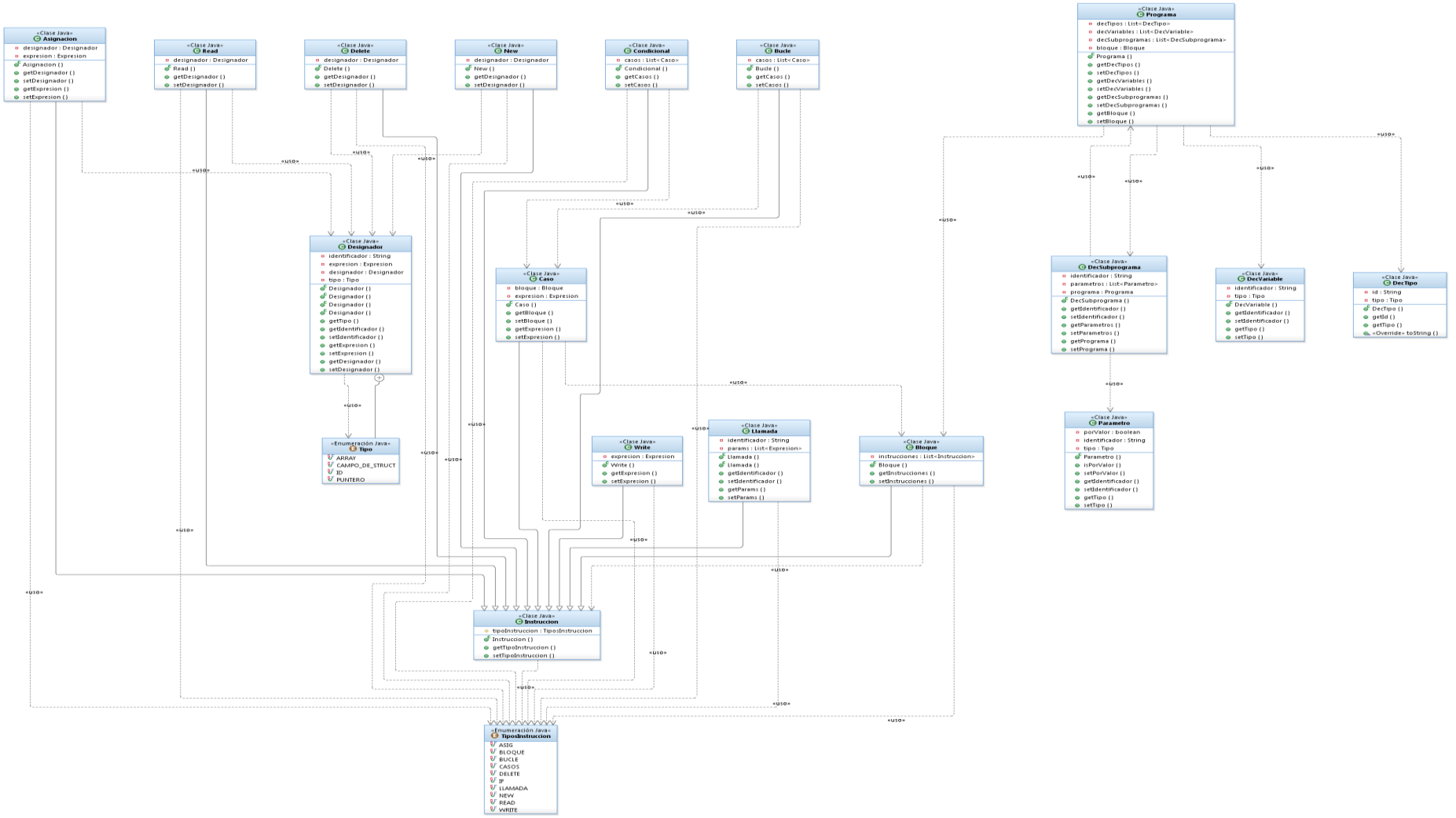


Diagrama operadores

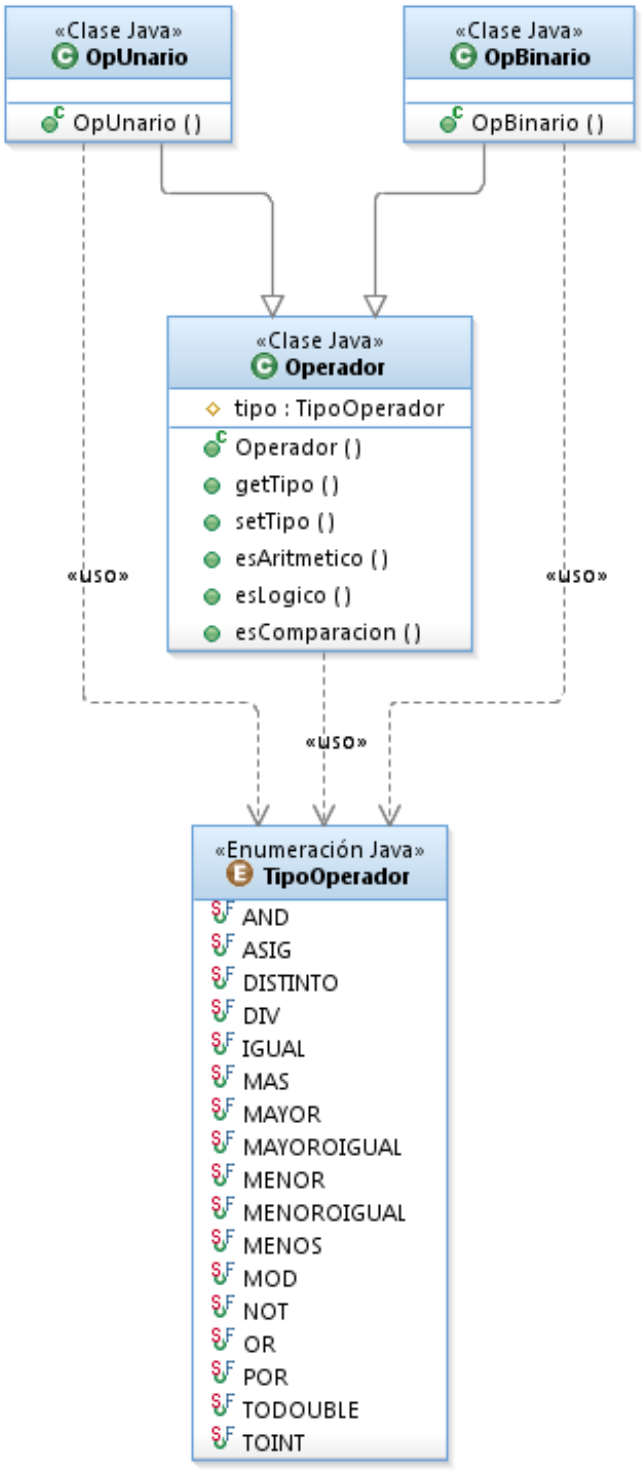
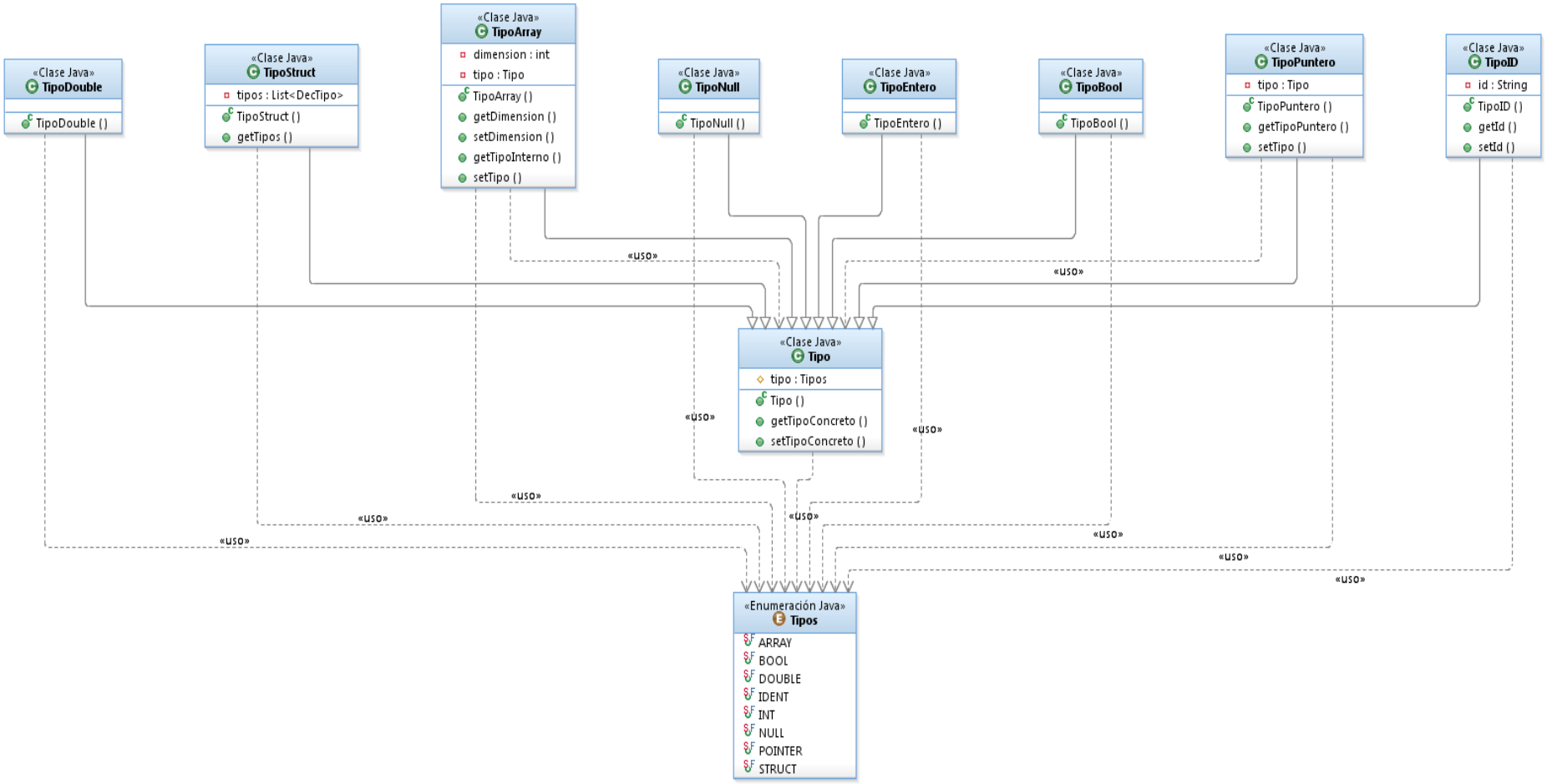


Diagrama de tipos



Constructora AST

```
prog: Lista(DecTipos) x Lista(DecVariables) x Lista(Procs) x Lista(Instr) ->
Prog

crearSeccionTipos : Lista(DecTipo) -> SeccionTipos
crearSeccionTiposVacia ->SeccionTipos
crearSeccionVarVacia: -> SeccionVariables
crearSeccionVariables: Lista(DecVariable)-> SeccionVariables

crearSeccionSubVacia: -> SeccionSubprogramas
crearSeccionsubprogramas : Lista(DecSubprograma) ->SeccionSubProgramas

ListaTipoVacia: -> Lista(DecTipo)
hazListaDectipos: Lista(DecTipo) x Dectipo-> Lista(DecTipo)
hazListaDectipos: ListaTipoVacia x DecTipo -> Lista(DecTipo)
crearDecTipo: Tipo x String -> DecTipo


ListaCamposVacia: ->Lista(campo)
HazListaCampos : Lista(Campo) x Campo -> Lista(Campo)
HazListaCampos: ListaCamposVacia x Campo -> Lista(Campo)
HazCampo: String x Tipo -> Campo

ListaDecVariablesvacia: -> Lista(DecVariable)
HazListaDecVariables: Lista(DecVariable) x DecVariable -> Lista(DecVariable)
HazListaDecVariables: ListaDecVariablesVacia x DecVariable->
Lista(DecVariable)
crearDecVariable: String x Tipo ->DecVariable



ListaDecSubprogramaVacia : ->Lista(DecSubprograma)
HazListaProcs: Lista(DecSubprograma) x DecSubprograma -> Lista(DecSubprograma)
HazListaProcs: ListaDecSubprogramaVacia x DecSubprograma -
>Lista(DecSubprograma)
crearDecSuprograma : String x Lista(Parametro) x seccionTipos x
seccionVariables x seccionSubprogramas x Instruccion



ListaParametrosVacia: -> Lista(Parametro)
HazListaParametros: Lista(Parametro) x Parametro -> Lista(Parametro)
HazListaParametros: ListaParametrosVacia x Parametro -> Lista(Parametro)
HazParametroValor: String x Tipo -> Parametro
HazParametroVariable: String x Tipo -> Parametro



ListaInstruccionesVacia: ->Lista(Instruccion)
HazListaInstrucciones : Lista(Intruccion) x Instruccion -> Lista(Instruccion)
HazListaInstrucciones : ListaInstruccionesVacia x Instruccion ->
Lista(Instruccion)

crearAsignacion: Designador x Exp ->Instruccion
crearBloque: Lista(Instruccion) -> Instruccion
crearBloqueVacio: ->Instruccion
crearIf: Lista(Caso) -> Instruccion
crearBucle: Lista(Caso)->Instruccion
crearLlamada: String x Argumentos -> Instruccion
```

```
crearLee: Designador -> Instruccion
crearEscribe: Exp -> Instruccion
crearNuevo: Designador->Instruccion
crearDelete: Designador -> Instruccion

ListaCasosVacia: -> Lista(Caso)
HazListaCasos: Lista(Caso) x Caso -> Lista(Caso)
HazListaCasos: ListaCasosVacia x Caso->Lista(Caso)
HazCaso: Exp x Instruccion-> Caso

crearListaArgVacia: ->Lista(Argumento)
crearListaArg: Lista(Argumento) -> Lista(Argumento)

ListaArgumentoVacia : ->Lista(Exp)
hazListaArgumentos: Lista(Exp) x Exp -> Lista(Exp)
hazListaArgumentos: ListaArgumentoVacia x Exp -> Lista(Exp)



crearDesignador: String -> Designador
crearDesignador: Designador x Exp -> Designador

crearDesignador: Designador x String -> Designador


crearInteger: Tipo
crearDouble:Tipo
crearBoolean:Tipo
crearID: Tipo
crearArray:Tipo x Integer -> Tipo
crearReg:  Lista(Campos) -> Tipo
crearPointer: Tipo -> Tipo


crearExpBinariaComp: Exp1 x OpComp x Exp1 -> Exp
crearExpBinariaAdi: Exp1 x OpAditivo x Exp2 -> Exp
crearExpBinariaMult: Exp2 x OpMultiplicativo x Exp3 -> Exp
crearExpUnaria:  OpUnario x Exp3 -> Exp

crearOpComp: OperadorBinarioComp
crearOpDistinto: OperadorBinarioComp
crearOpMayor: OperadorBinarioComp
crearOprMayorOIgual: OperadorBinarioComp
crearOpMenor: OperadorBinarioComp
CrearOpMenorOIgual: OperadorBinarioComp

crearOpMas: OperadorBinarioAdi
crearOpMenos: OperadorBinarioAdi
crearOpOr: OperadorBinarioAdi

crearOpPor: OperadorBinarioMult
crearOpDiv: OperadorBinarioMult
crearOpMod: OperadorBinarioMult
crearOpAnd: OperadorBinarioMult

crearOpMenosUna:OperadorUnario
crearOpNot:OperadorUnario
crearOpToInt:OperadorUnario
crearOpToDouble:OperadorUnario



decTipo: Tipo x String -> DecTipo
hazCampo: String x Tipo -> Campo
decVar: Tipo x String -> DecVariable
```

```
decProc: String x ListaParam x ListaDecTipos x ListaDecVariables x ListaInst -
> DecProc
HazParam:String x Tipo -> Parametro
HazParametroVar: String x Tipo -> Parametro
HazCaso: Exp X ListaInst -> Caso
-----------------------------Listas
hazListaDectipos: ListaDecTipos x DecTipo -> ListaDecTipos
ListaTipoVacio: -> ListaDecTipos
hazListaCampos: ListaCampos x Campo -> ListaCampos
ListaCamposVacia: -> ListaCampos
HazListaDecVariables: ListaDecVariables x DecVariable -> ListaDecVariables
ListaVariablesvacia: -> ListaDecVaribles
HazListaProcs: ListaProcs x DecProc -> ListaProcs
ListaProcsVacia: -> ListaProcs
ListaParamVacia: -> ListaParam
HazListaParametros: ListaParametros x Parametro -> ListaParametros
ListaParametrosVacia: -> ListaParametros
hazListaIntruciones: ListaInst x Instr -> ListaInst
ListaInstVacia: -> ListaInst
HazListaCasos: ListaCasos x Caso -> ListaCasos
ListaCasosVacia -> ListaCasos
HazListaArgumentos: ListaArgumentos x Argumento -> ListaArgumentos
ListaArgumentosVacia: -> ListaArgumentos
------------------------------TIPOS
bool: Tipo
int: Tipo
double: Tipo
array: Tipo x Integer -> Tipo
reg: List(String x Tipo) -> Tipo
pointer: Tipo->Tipo
-------------------------------INSTRUCCIONES
asignar: Desig x Exp -> Inst
escribe: Exp ->Inst
lee: Desig -> Inst
nuevo: Desig ->Inst
borra: Desing->Inst
bloque: ListInst->Inst
cond: Exp x Inst-> Inst
bucle: Exp x Inst ->Inst
llamada: String x ListArg

-------------------------------DESIGNADORES
var : String -> Desig
selCampo: Desig x String -> Desig
indexElem: Desig x Exp ->Desig
deref: Desig -> Desig
-------------------------------EXPRESIONES
true: ->Exp
false: ->Exp
num: Integer->Exp
mem: Desig->Exp
igual: Exp x Exp -> Exp
distinto: Exp x Exp -> Exp
menor: Exp x Exp -> Exp
mayor: Exp x Exp -> Exp
menorOIgual: Exp x Exp -> Exp
mayorOIgual: Exp x Exp -> Exp
suma: Exp x Exp -> Exp
resta: Exp x Exp -> Exp
mul: Exp x Exp -> Exp
div: Exp x Exp -> Exp
mod: Exp x Exp -> Exp
neg: Exp -> Exp
and: Exp x Exp -> Exp
or: Exp x Exp -> Exp
not: Exp -> Exp
```