**Compiladores e Intérpretes**

**Informe etapa 2**

**Analizador sintáctico**

**Bonanno, Pablo Ariel**

**Licenciatura en Ciencias de la Computación**

**Universidad Nacional del Sur**

**Módulos**

Además de los módulos especificados en el informe de la etapa anterior, el programa cuenta con un analizador sintáctico, el cual recibe un analizador léxico en su constructor.  
El analizador sintáctico utiliza al analizador léxico para solicitarle tokens y verifica que la secuencia de tokens proporcionada corresponda a una cadena generada por la gramática de MiniJava, especificada más adelante.

Si el analizador sintáctico encuentra una o más secuencias de tokens que no correspondan al lenguaje, lanza una excepción.

El módulo Main es el encargado de crear el analizador sintáctico e inyectar sus dependencias.

Además, se añadió un paquete de utilidades con una clase CustomHashSet que extiende la clase HashSet provista por Java. CustomHashSet es utilizada por el analizador sintáctico.

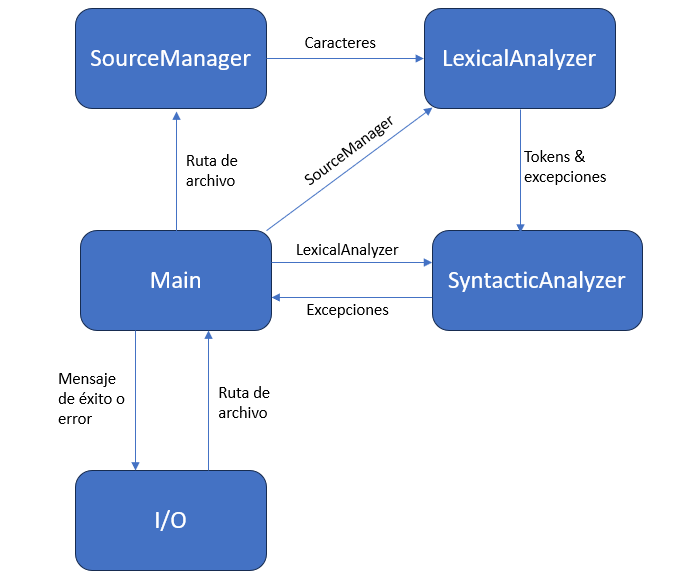


Diagrama general del funcionamiento del programa

**Excepciones**

Se crearon dos tipos de excepciones nuevas: SyntacticException y SyntacticExceptions. La segunda es la que efectivamente es lanzada por el analizador sintáctico. Esta cuenta con una cola de SyntacticException. De esta forma, el analizador sintáctico es capaz de realizar el análisis completo del archivo fuente proporcionado, encolando excepciones del tipo SyntacticException y finalmente lanzando una excepción del tipo SyntacticExceptions con todos los errores encontrados en el análisis.

El módulo Main es el encargado de manejar la excepción SyntacticExceptions, desencolando todas las excepciones que allí se encuentran e imprimiendo un mensaje de error acorde por pantalla.

**Gramática**

Para su mejor lectura, una copia de todas las gramáticas aquí especificadas podrá ser encontrada en archivos de texto entre los archivos entregados.

Observación: el carácter es utilizado como la cadena vacía.

Para obtener una gramática LL(1) a partir de la gramática que especifica las reglas de sintaxis de miniJava, primero se extendió a esta para cumplir con los logros deseados.

La gramática obtenida es la siguiente (copia en el archivo gramaticaExpandida.txt):

<Inicial> ::= <ListaClasesEInterfaces> eof

<ListaClasesEInterfaces> ::= <Clase> <ListaClasesEInterfaces> | <Interfaz> <ListaClasesEInterfaces> |

<Clase> ::= <ModificadorOpcional> class idClase <GenericidadOpcional> <HerenciaOpcional> { <ListaMiembros> }

<Interfaz> ::= <ModificadorOpcional> interface idClase <GenericidadOpcional> <ExtendsOpcional> { <ListaMiembrosInterfaz> }

<ModificadorOpcional> ::= abstract | static | final |

<HerenciaOpcional> ::= extends idClase <GenericidadOpcional>| implements idClase <GenericidadOpcional>|

<ExtendsOpcional> ::= extends idClase <GenericidadOpcional>|

<GenericidadODiamanteOpcional> ::= <GenericidadOpcional> | <>

<GenericidadOpcional> ::= < idClase > |

<ListaMiembros> ::= <VisibilidadOpcional> <Miembro> <ListaMiembros> |

<ListaMiembrosInterfaz> ::= <VisibilidadOpcional> <MiembroInterfaz> <ListaMiembrosInterfaz> |

<Miembro> ::= <Atributo> | <Metodo> | <Constructor>

<MiembroInterfaz> ::= <AtributoInicializado> | <Metodo>

<Atributo> ::= <Tipo> idMetVar <AsignacionOpcional> ;

<AsignacionOpcional> ::= = <Expresion> |

<AtributoInicializado> ::= <Tipo> idMetVar = <Expresion> ;

<Metodo> ::= <ModificadorOpcional> <TipoMetodo> idMetVar <ArgsFormales> <BloqueOpcional>

<Constructor> ::= public idClase <ArgsFormales> <Bloque>

<TipoMetodo> ::= <Tipo> | void

<Tipo> ::= <TipoPrimitivo> | idClase <GenericidadOpcional>

<TipoPrimitivo> ::= boolean | char | int

<ArgsFormales> ::= ( <ListaArgsFormalesOpcional> )

<ListaArgsFormalesOpcional> ::= <ListaArgsFormales> |

<ListaArgsFormales> ::= <ArgFormal>

<ListaArgsFormales> ::= <ListaArgsFormales> , <ArgFormal>

<ArgFormal> ::= <Tipo> idMetVar

<BloqueOpcional> ::= <Bloque> | ;

<Bloque> ::= { <ListaSentencias> }

<ListaSentencias> ::= <Sentencia> <ListaSentencias> |

<Sentencia> ::= ;

<Sentencia> ::= <Asignacion> ;

<Sentencia> ::= <Llamada> ;

<Sentencia> ::= <VarLocal> ;

<Sentencia> ::= <Return> ;

<Sentencia> ::= <If>

<Sentencia> ::= <While>

<Sentencia> ::= <Bloque>

<Sentencia> ::= <For>

<For> ::= for ( <Expresion> <CondicionFor> ) <Sentencia> | for( <VarLocal> <CondicionFor>) <Sentencia>

<CondicionFor> ::= ; <ExpresionCompuesta> ; <Expresion> | : <Expresion>

<Asignacion> ::= <Expresion>

<Llamada> ::= <Expresion>

<VarLocal> ::= var idMetVar = <ExpresionCompuesta> ; | <Tipo> idMetVar <DeclaracionMultiple> <AsignacionOpcional> ;

<DeclaracionMultiple> ::= , idMetVar <DeclaracionMultiple> |

<Return> ::= return <ExpresionOpcional>

<ExpresionOpcional> ::= <Expresion> |

<If> ::= if ( <Expresion> ) <Sentencia>

<If> ::= if ( <Expresion> ) <Sentencia> else <Sentencia>

<While> ::= while ( <Expresion> ) <Sentencia>

<Expresion> ::= <ExpresionCompuesta> <OperadorAsignacion> <ExpresionCompuesta> <OperadorTernarioOpcional>

<Expresion> ::= <ExpresionCompuesta> <OperadorTernarioOpcional>

<OperadorTernarioOpcional> ::= ? <Expresion> : <ExpresionCompuesta> <OperadorTernarioOpcional> |

<OperadorAsignacion> ::= = | += | -=

<ExpresionCompuesta> ::= <ExpresionCompuesta> <OperadorBinario> <ExpresionBasica>

<ExpresionCompuesta> ::= <ExpresionBasica>

<OperadorBinario> ::= || | && | == | != | < | > | <= | >= | + | - | \* | / | %

<ExpresionBasica> ::= <OperadorUnario> <Operando>

<ExpresionBasica> ::= <Operando>

<OperadorUnario> ::= + | ++ | − | −− | !

<Operando> ::= <Primitivo>

<Operando> ::= <Referencia>

<Primitivo> ::= true | false | intLiteral | charLiteral | null

<Referencia> ::= <Primario>

<Referencia> ::= <Referencia> <VarEncadenada>

<Referencia> ::= <Referencia> <MetodoEncadenado>

<Primario> ::= this

<Primario> ::= stringLiteral

<Primario> ::= <AccesoVar>

<Primario> ::= <LlamadaConstructor>

<Primario> ::= <LlamadaMetodo>

<Primario> ::= <LlamadaMetodoEstatico>

<Primario> ::= <ExpresionParentizada>

<AccesoVar> ::= idMetVar

<LlamadaConstructor> ::= new idClase <GenericidadODiamanteOpcional> <ArgsActuales>

<ExpresionParentizada> ::= ( <Expresion> )

<LlamadaMetodo> ::= idMetVar <ArgsActuales>

<LlamadaMetodoEstatico> ::= idClase . idMetVar <ArgsActuales>

<ArgsActuales> ::= ( <ListaExpsOpcional> )

<ListaExpsOpcional> ::= <ListaExps> |

<ListaExps> ::= <Expresion>

<ListaExps> ::= <Expresion> , <ListaExps>

<VarEncadenada> ::= . idMetVar

<MetodoEncadenado> ::= . idMetVar <ArgsActuales>

Luego, se eliminaron las recursiones a izquierda, y se resolvieron problemas de ambigüedad y decisión, mediante factorización de ser posible, o empleando algún otro método para resolver estos problemas.

La gramática final utilizada es la siguiente (copia en el archivo gramaticaFinal.txt):

<Inicial> ::= <ListaClasesEInterfaces> eof

<ListaClasesEInterfaces> ::= <ModificadorOpcional> <ListaClasesEInterfaces2> |

<ListaClasesEInterfaces2> ::= <Clase> <ListaClasesEInterfaces> | <Interfaz> <ListaClasesEInterfaces>

<Clase> ::= class idClase <GenericidadOpcional> <HerenciaOpcional> { <ListaMiembros> }

<Interfaz> ::= interface idClase <GenericidadOpcional> <ExtendsOpcional> { <ListaMiembrosInterfaz> }

<ModificadorOpcional> ::= abstract | static | final |

<Modificador> ::= abstract | static | final

<VisibilidadOpcional> ::= public | private |

<PrivateOpcional> ::= private |

<HerenciaOpcional> ::= extends idClase <GenericidadOpcional>| implements idClase <GenericidadOpcional>|

<ExtendsOpcional> ::= extends idClase <GenericidadOpcional>|

<GenericidadODiamanteOpcional> ::= < <IdClaseOpcional> > |

<IdClaseOpcional> ::= idClase |

<GenericidadOpcional> ::= < idClase > |

<ListaMiembros> ::= <PrivateOpcional> <AtributoOMetodo> <ListaMiembros> |

<ListaMIembros> ::= public <Miembro> <ListaMiembros>

<ListaMiembrosInterfaz> ::= <VisibilidadOpcional> <MiembroInterfaz> <ListaMiembrosInterfaz> |

<Miembro> ::= <TipoPrimitivo> idMetVar <CierreAtributoMetodo> | <Modificador> <TipoMetodo> idMetVar <ArgsFormalesYBloqueOpcional>

<Miembro> ::= void idMetVar <ArgsFormalesYBloqueOpcional> | idClase <ConstructorUOtroMiembro>

<MiembroInterfaz> ::= <Tipo> idMetVar = <Expresion> ; | <Modificador> <TipoMetodo> idMetVar <ArgsFormalesYBloqueOpcional>

<MiembroInterfaz> ::= void idMetVar <ArgsFormalesYBloqueOpcional>

<CierreAtributoMetodo> ::= <AsignacionOpcional> ; | <ArgsFormalesYBloqueOpcional>

<ArgsFormalesYBloqueOpcional> ::= <ArgsFormales> <BloqueOpcional>

<Atributo> ::= <Tipo> idMetVar <AsignacionOpcional> ;

<AsignacionOpcional> ::= = <Expresion> |

<AtributoInicializado> ::= <Tipo> idMetVar = <Expresion> ;

<AtributoOMetodo> ::= <Tipo> idMetVar <CierreAtributoMetodo>

<AtributoOMetodo> ::= <Modificador> <TipoMetodo> idMetVar <ArgsFormalesYBloqueOpcional> | void idMetVar <ArgsFormalesYBloqueOpcional>

<ConstructorUOtroMiembro> ::= <ArgsFormales> <Bloque> | idMetVar <CierreAtributoMetodo>

<TipoMetodo> ::= <Tipo> | void

<Tipo> ::= <TipoPrimitivo> | idClase <GenericidadOpcional>

<TipoPrimitivo> ::= boolean | char | int

<ArgsFormales> ::= ( <ListaArgsFormalesOpcional> )

<ListaArgsFormalesOpcional> ::= <ListaArgsFormales> |

<ListaArgsFormales> ::= <ArgFormal> <ListaArgsFormales2>

<ListaArgsFormales2> ::= , <ArgFormal> <ListaArgsFormales2> |

<ArgFormal> ::= <Tipo> idMetVar

<BloqueOpcional> ::= <Bloque> | ;

<Bloque> ::= { <ListaSentencias> }

<ListaSentencias> ::= <Sentencia> <ListaSentencias> |

<Sentencia> ::= ;

<Sentencia> ::= <AsignacionLlamadaOVarLocal> ;

<Sentencia> ::= <Return> ;

<Sentencia> ::= <If>

<Sentencia> ::= <While>

<Sentencia> ::= <Bloque>

<Sentencia> ::= <For>

<AsignacionLlamadaOVarLocal> ::= <VarLocalConVar>

<AsignacionLlamadaOVarLocal> ::= <VarLocalConTipoPrimitivo>

<AsignacionLlamadaOVarLocal> ::= idClase <MetodoEstativoOVarLocal>

<AsignacionLlamadaOVarLocal> ::= <ExpresionSinLlamadaMetodoEstatico>

<MetodoEstativoOVarLocal> ::= . idMetVar <ArgsActuales> <Referencia2> <ExpresionCompuesta2> <Expresion2>

<MetodoEstativoOVarLocal> ::= <GenericidadOpcional> idMetVar <DeclaracionMultiple> <AsignacionOpcional>

<ExpresionSinLlamadaMetodoEstatico> ::= <ExpresionCompuestaSinLLamadaMetodoEstatico> <Expresion2>

<ExpresionCompuestaSinLLamadaMetodoEstatico> ::= <ExpresionBasica> <ExpresionCompuesta2>

<VarLocalConVar> ::= var idMetVar = <Expresion>

<VarLocalConTipoPrimistivo> ::= <TipoPrimitivo> idMetVar <DeclaracionMultiple> <AsignacionOpcional>

<For> ::= for ( <For2> <CondicionFor ) <Sentencia>

<For2> ::= <VarLocalConVar> | <TipoPrimitivo> idMetVar <DeclaracionMultiple> <AsignacionOpcional>

<For2> ::= idClase <MetodoEstaticoOVarLocalInicializada> | <ExpresionSinLlamadaMetodoEstatico> |

<CondicionFor> ::= ; <ExpresionCompuestaOpcional> ; <ExpresionOpcional> | : <Expresion>

<MetodoEstaticoOVarLocalInicializada> ::= . idMetVar <ArgsActuales> <Referencia2> <ExpresionCompuesta2> <Expresion2>

<MetodoEstaticoOVarLocalInicializada> ::= <GerenicidadOpcional> idMetVar <DeclaracionMultiple> <AsignacionOpcional>

<AsignacionOLlamada> ::= <Expresion>

<VarLocal> ::= var idMetVar = <Expression> | <Tipo> idMetVar <DeclaracionMultiple> <AsignacionOpcional>

<DeclaracionMultiple> ::= , idMetVar <DeclaracionMultiple> |

<Return> ::= return <ExpresionOpcional>

<ExpresionOpcional> ::= <Expresion> |

<If> ::= if ( <Expresion> ) <Sentencia> <SentenciaElse>

<SentenciaElse> ::= else <Sentencia> |

<While> ::= while ( <Expresion> ) <Sentencia>

<Expresion> ::= <ExpresionCompuesta> <Expresion2>

<Expresion2> ::= <OperadorAsignacion> <ExpresionCompuesta> <OperadorTernarioOpcional> | <OperadorTernarioOpcional>

<OperadorTernarioOpcional> ::= ? <Expresion> : <ExpresionCompuesta> <OperadorTernarioOpcional> |

<OperadorAsignacion> ::= = | += | -=

<ExpresionCompuesta> ::= <ExpresionBasica> <ExpresionCompuesta2>

<ExpresionCompuesta> ::= <LlamadaMetodoEstatico> <Referencia2> <ExpresionCompuesta2>

<ExpresionCompuesta2> ::= <OperadorBinario> <ExpresionCompuesta> |

<OperadorBinario> ::= || | && | == | != | < | > | <= | >= | + | - | \* | / | %

<ExpresionBasica> ::= <OperadorUnario> <Operando>

<ExpresionBasica> ::= <Operando>

<OperadorUnario> ::= + | ++ | − | −− | !

<Operando> ::= <Primitivo>

<Operando> ::= <Referencia>

<Primitivo> ::= true | false | intLiteral | charLiteral | null

<Referencia> ::= <Primario> <Referencia2>

<Referencia2> ::= <VarMetodoEncadenado> <Referencia2> |

<Primario> ::= this

<Primario> ::= stringLiteral

<Primario> ::= <AccesoVarLlamadaMetodo>

<Primario> ::= <LlamadaConstructor>

<Primario> ::= <ExpresionParentizada>

<AccesoVarLlamadaMetodo> ::= idMetVar <ArgsActualesOpcional>

<ArgsActualesOpcional> ::= <ArgsActuales> |

<LlamadaConstructor> ::= new idClase <GenericidadODiamanteOpcional> <ArgsActuales>

<ExpresionParentizada> ::= ( <Expresion> )

<LlamadaMetodoEstatico> ::= idClase . idMetVar <ArgsActuales>

<ArgsActuales> ::= ( <ListaExpsOpcional> )

<ListaExpsOpcional> ::= <ListaExps> |

<ListaExps> ::= <Expresion> <ListaExps2>

<ListaExps2> ::= , <Expresion> <ListaExps2> |

<VarMetodoEncadenado> ::= . idMetVar < ArgsActualesOpcional>

**Logros**

Se intentaron cumplir los siguientes logros:

* Variables locales Clásicas E2
* Recuperación de Errores Modo Panico
* Genericidad E2
* Operador Ternario E2
* Visibilidad Mejorada E2
* Fors! E2
* Interfaces E2
* Atributos Inicializados
* Interfaces Genericas E2