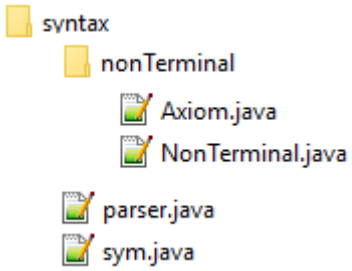
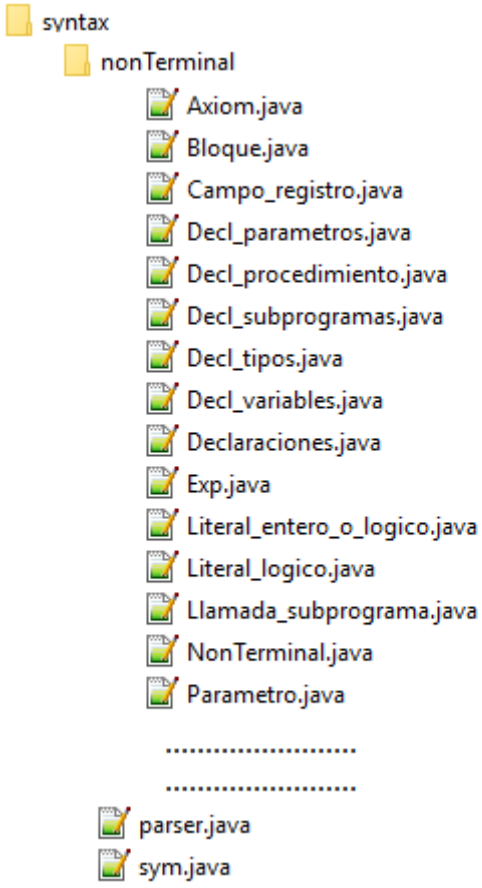
Clases de símbolos no terminales

C:\..\ArquitecturaPLII-cursoXX\src\compiler\syntax\nonTerminal



En la carpeta “nonTerminal” iremos creando archivos.java para cada no terminal del lenguaje que contenga o pueda contener algún atributo “se recomienda que tengan el mismo nombre que los no terminales declarados en C:\..\ArquitecturaPLII-cursoXX \doc\specs\parser.cup variando el primer carácter de minúscula a mayúscula o viceversa.

```
// Declaración de no terminales
// no modificar los propuestos

non terminal                    program;
non terminal    Axiom           axiom;

non terminal    Bloque          bloque;
non terminal    Campo_registro  campo_registro;
non terminal    Decl_parametros decl_parametros;
non terminal    Decl_procedimiento  decl_procedimiento;
non teminal     Decl_subprogramas   decl_subprogramas;
non terminal    Decl_tipos      decl_tipos;
non terminal    Decl_variables  decl_variables;
non terminal    Declaraciones   declaraciones;
non terminal    Exp             exp;
non terminal    Literal_entero_o_logico  literal_entero_o_logico;
non terminal    Literal_logico  literal_logico;
non terminal    Llamada_subprograma  llamada_suprograma;
……
……
```



Las clases que se crean para cada símbolo no terminal del lenguaje son herederas de la clase “NonTeminal”, incluida la clase del no terminal “Axiom”, aunque esta clase es abstracta, y las otras clases que creemos para los no terminales no deben ser abstractas, para poder instanciarlas.

```java
package compiler.syntax.nonTerminal;

import java.util.ArrayList;
import java.util.List;

import es.uned.lsi.compiler.intermediate.QuadrupleIF;
import es.uned.lsi.compiler.syntax.nonTerminal.NonTerminalIF;

/**
 * Abstract class for non terminals.
 */
public abstract class NonTerminal
    implements NonTerminalIF
{
    private List<QuadrupleIF> intermediateCode;

    /**
     * Constructor for NonTerminal.
     */
    public NonTerminal ()
    {
        super ();
        this.intermediateCode = new ArrayList<QuadrupleIF> ();
    }

    /**
     * Returns the intermediateCode.
     * @return Returns the intermediateCode.
     */
    public List<QuadrupleIF> getIntermediateCode ()
    {
        return intermediateCode;
    }

    /**
     * Sets The intermediateCode.
     * @param intermediateCode The intermediateCode to set.
     */
    public void setIntermediateCode (List<QuadrupleIF> intermediateCode)
    {
        this.intermediateCode = intermediateCode;
    }

}
```

```java
package compiler.syntax.nonTerminal;


/**
 * Abstract Class for Axiom non terminal.
 */
public abstract class Axiom
    extends NonTerminal
{
    /**
     * Constructor for Axiom.
     */
    public Axiom ()
    {
        super ();
    }
}
```

```java
package compiler.syntax.nonTerminal;

import java.util.*;

/**
 * Class for Expresion non terminal.
 */
public class Expresion
    extends NonTerminal
{
    private boolean logica;
    private String tipo;
    private String id;

    /**
     * Constructor for Expresion.
     */
    public Expresion()
    {
        super ();
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public boolean getLogica() {
        return logica;
    }

    public void setLogica(boolean logica) {
        this.logica = logica;
    }

}
```

Propagar atributos

Consultar sección 3.1.3 del documento “DirectricesImplementacionPracticaPL2_2020-2021”

declaracionVariable ::= IDENTIFICADOR DOSPUNTOS **tipoPrimitivo ;**

tipoPrimitivo ::= INTEGER | BOOLEAN ;

Introducir atributos, propagar estos por medio de acciones semánticas:

declaracionVariable ::= IDENTIFICADOR**:id** DOSPUNTOS tipoPrimitivo**:tp**

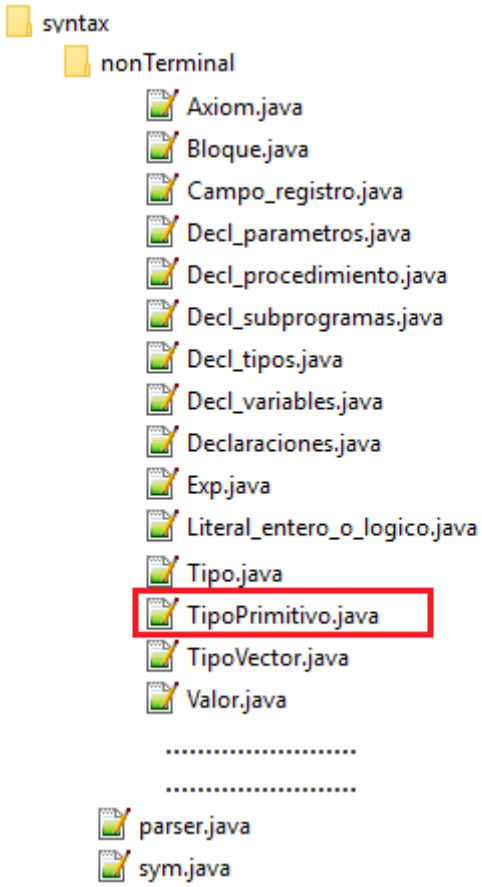
```
{:
//recuperamos el ámbito donde nos encontramos y obtenemos la tabla de símbolos
ScopeIF scope = scopeManager.getCurrentScope();
SymbolTableIF tablaSimbolos =scope.getSymbolTable();
//comprobamos que la variable no este ya contenida en la tabla de símbolos
if (tablaSimbolos.containsSymbol(id.getLexema())) {
        semanticErrorManager.semanticFatalError("Variable " + nombre + " ya declarada");
} else {
        Sytem.out.println ("Variable " + nombre + " aun No declarada");
        //Obtenemos el tipo que correspondiente
        TypeIF tipo = scopeManager.searchType(tp.getTipo());
        //si el tipo retornado es nulo se emite un error fatal
        if (tipo == null ) {
                semanticErrorManager.semanticFatalError("Variable Tipo nulo = " + tipo);
        } else {
                //Creamos la variable indicando su ámbito, nombre, tipo.  ¿valor?
                SymbolVariable sV = new SymbolVariable(scope, nombre, tipo);
                //Registramos la variable en la tabla de símbolos
                tablaSimbolos.addSymbol(sV);
        } // fin if
} //fin if
:}
```

```
tipoPrimitivo ::= INTEGER

{:
        TipoPrimitivo tP=new TipoPrimitivo();
        tP.setTipo("ENTERO");

        RESULT = tP;
:} | BOOLEAN
{:
        TipoPrimitivo tP=new TipoPrimitivo();
        tP.setTipo("LOGICO");

        RESULT = tP;
:};
```

Deberemos de crear una clase que extienda de **nonTerminal** para el no terminal **tipoPrimitivo** e incluir en ella los atributos que sean necesarios.

C:\..\ArquitecturaPLII-cursoXX\src\compiler\syntax\nonTerminal

```
package compiler.syntax.nonTerminal;

public class TipoPrimitivo extends NonTerminal {

        private String tipo;

        public TipoPrimitivo () {
                super ();
         }

        public String getTipo() {
                return tipo;
        }

        public void setTipo(String tipo) {
                this.tipo = tipo;
        }
}
```



También deberemos declarar la clase correspondiente al no terminal en el fichero parser.cup

```
// Declaración de no terminales
// no modificar los propuestos
non terminal program;
non terminal    Axiom                   axiom;
non terminal    Campo_registro          campo_registro;
non terminal    Decl_parametros         decl_parametros;
non terminal    Decl_procedimiento      decl_procedimiento;
non terminal    Decl_subprogramas       decl_ subprogramas;
……
……
non terminal    Tipo_Primitivo          tipo_primitivo;
……
……
```

```
secuencia_parametros ::= secuencia_parametros:sp PUNTO_Y_COMA parametro:param
{:
        Secuencia_parametros sParam = new Secuencia_parametros();
        sParam.setListaCampos(sp.getListaCampos());
        Parametro p = new Parametro(param.getId(), param.getTipo());
        sParam.addParametro(p);

        RESULT = sParam;
:} | parametro:param
{:
        Secuencia_parametros sParam = new Secuencia_parametros();
        Parametro p = new Parametro(param.getId(), param.getTipo());
        sParam.addParametro(p);

        RESULT = sParam;
:};



parametro ::= IDENTIFICADOR:id DOS_PUNTOS tipo_primitivo:tipo
{:
        Parametro param = new Parametro();
        param.setId(id.getLexema());
        param.setTipo(tipo.getNombre());

        RESULT = param;
:};
```

Deberemos de crear una clase que extienda de **nonTerminal** para el no terminal **secuencia_parametros** y **parametro** e incluir en ellas los atributos que sean necesarios.

C:\..\ArquitecturaPLII-cursoXX\src\compiler\syntax\nonTerminal

```
package compiler.syntax.nonTerminal;

import java.util.*;
/**
 * Class for Secuencia_parametros non terminal.
 */
public class Secuencia_parametros extends NonTerminal
{
	private List<Parametro> listaCampos=null;

	/**
	 * Constructor for Secuencia_parametros.
	*/
	public Secuencia_parametros ()
	{
		super ();
	}



	public void addParametro(Parametro param) {
		if (listaCampos == null) {
			listaCampos = new ArrayList<Parametro>();
		}
		if(param!=null) {
			listaCampos.add(param);
		}
	}
	public List<Parametro> getListaCampos() {
		return listaCampos;
	}
	public void setListaCampos(List<Parametro> listaCampos) {
		this.listaCampos = listaCampos;
	}
	public int getSize(){
		 return this.listaCampos.size();
	 }
}
```

```java
package compiler.syntax.nonTerminal;

import java.util.*;
/**
 * Class for Parametro non terminal.
 */
public class Parametro extends NonTerminal
{
	private String id;
	private String tipo;

	 /**
	 * Constructor for Parametro.
	*/
	public Parametro ()
	{
		super ();
	}
	public Parametro (String id, String tipo)
	 {
		super ();
		this.id = id;
		this.tipo = tipo;
	}
	public String getTipo() {
		return tipo;
	}
	public void setTipo(String tipo) {
		this.tipo = tipo;
	}
	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}
}
```