Tutorial para agregar un nuevo parser a la aplicación "ParserTester"

A continuación son mostrados los pasos básicos a seguir para agregar un nuevo parser a la aplicación "ParserTester"

1. En primera instancia se define el nuevo Parser en la clase MundoParsers, localizada en el paquete "uniandes.teolen.parserJavaCC.mundoParser" (ver Figura 1). Para hacer esto, debemos:
    a. Agregar en el constructor de la clase el nombre del nuevo parser (ver Figura 2)
    b. Crear un nuevo método en la clase que retorne una instancia de este (ver Figura 3)
    c. Crear un nuevo caso en el método "procesarCadena" de la clase, en el que se cree una nueva instancia del nuevo parser y se especifiquen los pasos a seguir para procesar una cadena. Tenga en cuenta que la respuesta del nuevo parser debe ser almacenada en la variable de tipo String, "resp" (ver Figura 4)

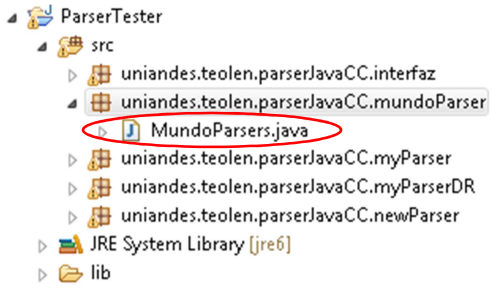


Figura 1. Ubicación de la clase MundoParsers

```
public  MundoParsers () {

    // Agreguen al final de esta lista los nombres del nuevo parser

    parsers.add("ParserDR");
    parsers.add("JavaCC");
    parsers.add("Nuevo Parser");

    currentParser =  0;

}
```

Figura 2. Constructor de la clase MundoParsers

```
public ParserCAML getParserCAML(){
    return new ParserCAML(System.in);
}

public ParserDR getParserDR(){
    return new ParserDR();
}

public NuevoParser getNuevoParser(){
    return new NuevoParser(System.in);
}

public String getStringCurrentParser(){
    return parsers.get(currentParser);
}

public int getCurrentParser() {
    return currentParser;
}
```

Figura 3. Creación de un nuevo método en la clase MundoParsers, que retorna una instancia del nuevo parser

```
public String  procesarCadena(String texto) {
    String resp = "";

    if(parsers.get(currentParser).equals("ParserDR")){
        ParserDR parserDR = getParserDR();
        parserDR.ReInit(new java.io.StringReader(texto));
        try {
            parserDR.Lexp();
            resp = new String("OK    \n");
        }catch (Exception e) {
            resp = new String ("Error de Sintaxis: "+e.getMessage());
         } catch (Error e) {
             resp = new String ("Error Lexico: "+e.getMessage());
    }
    }
    else if(parsers.get(currentParser).equals("JavaCC")){
        ParserCAML parserCAML = getParserCAML();
        parserCAML.ReInit(new java.io.StringReader(texto));
        ArrayList <Integer>  myList = new ArrayList <Integer> ();
        try {
            myList = parserCAML.camlList();
            resp = new String("OK    "+ myList + "\n");


        }catch (Exception e) {
                resp = new String ("Error de Sintaxis: "+e.getMessage());
        } catch (Error e) {
             resp = new String ("Error Lexico: "+e.getMessage());
        }
    }
    else if(parsers.get(currentParser).equals("Nuevo Parser")){
        NuevoParser nuevoParser = getNuevoParser();
        nuevoParser.ReInit(new java.io.StringReader(texto));
        try {
            nuevoParser.one_line();
            resp = new String("OK    \n");
        }catch (Exception e) {
            resp = new String ("Error de Sintaxis: "+e.getMessage());
         } catch (Error e) {
             resp = new String ("Error Lexico: "+e.getMessage());
         }
    }
    return "\n SISTEMA " + parsers.get(currentParser) + ": " + resp + "\n";
}
```

Figura 4. Inserción de un nuevo caso para el procesamiento del nuevo parser, en el método “procesarCadena” de la clase MundoParsers

2. Creamos un nuevo paquete en el proyecto, asociado al nuevo parser (ver Figura 5), de manera similar a parserCAML y parserDR. En este paquete se debe crear un nuevo archivo javaCC (tenga en cuenta que para hacer esto debe tener instalado el plugin de javaCC), para lo cual damos click derecho sobre el paquete y elegimos la opción “New”, en el submenú que aparece se debe elegir la opción “other” (ver Figura 6)

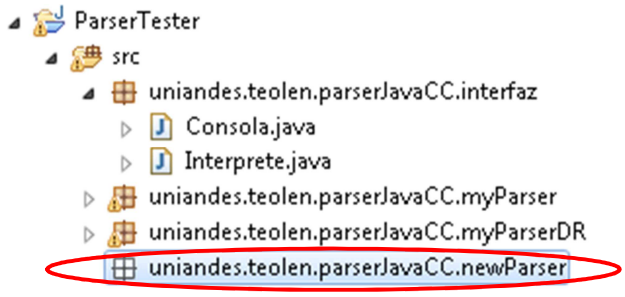


Figura 5. Paquete asociado al nuevo parser

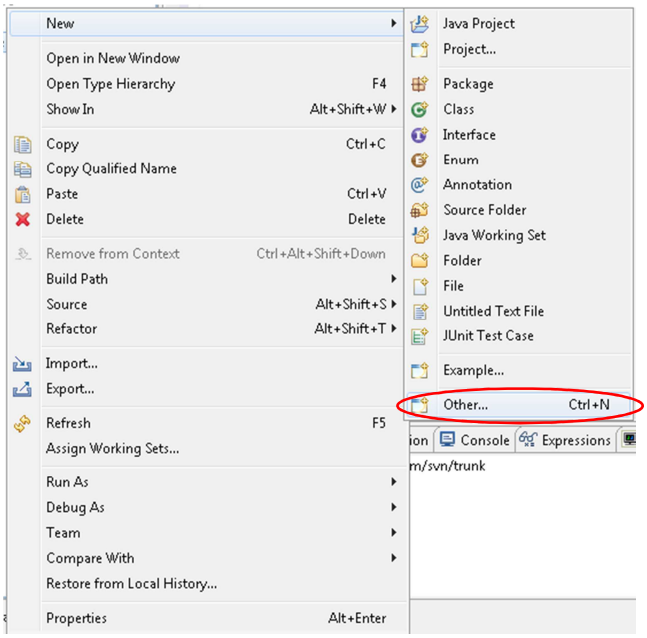


Figura 6. Creación de un nuevo archivo javaCC

3. Aparecerá una nueva ventana en la que se nos pide elegir el tipo de archivo que deseamos crear, allí elegimos la opción “Java CC Template File”, que se encuentra en la carpeta de JavaCC (ver Figura 7) y damos click en “Next”

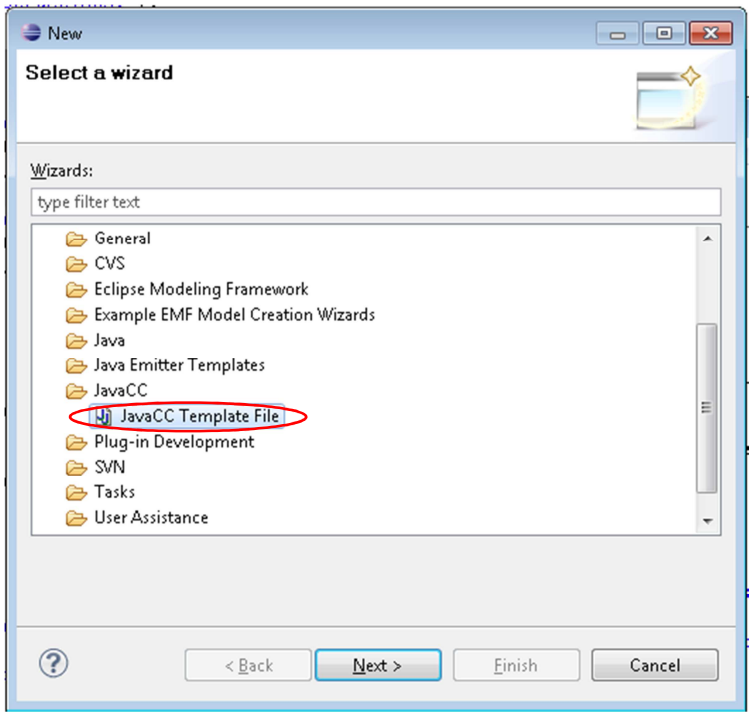


Figura 7. Creación de un nuevo archivo javaCC

4. En el siguiente cuadro de diálogo, se define el nombre del archivo javaCC como “NuevoParser” y damos click en “Finish” (ver Figura 8). De esta manera son creados en el paquete del nuevo parser una serie de archivos, dentro de los que se encuentra un archivo con extensión “.jj”, correspondiente al archivo javaCC dentro del cual se definirá la gramática del nuevo parser

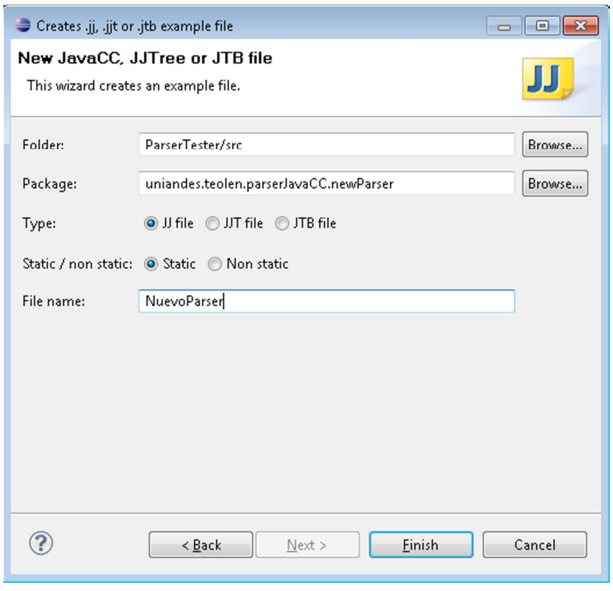


Figura 8. Creación de un nuevo archivo javaCC

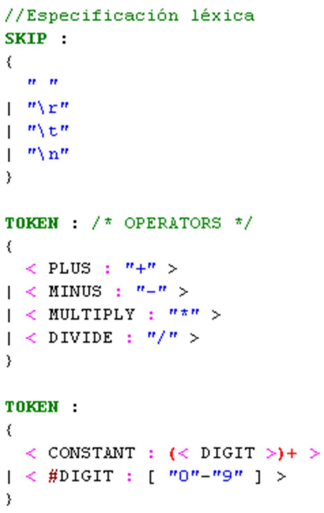
5. La estructura de los archivos javaCC se divide en 4 partes[1]:
    a. En la primera parte son definidas las opciones generales como: si se va a distnguir entre mayúsculas y minúsculas (estableciendo el valor de la cláusula IGNORE_CASE en true o en false, según el caso), si se van a generar métodos estáticos o no (a través de la clausula STATIC), etc.
    b. La segunda parte permite definir el nombre del analizador, en este caso "NuevoParser", así como el código Java que será añadido directamente a este último, como el nombre del paquete en el que se incluye, las cláusulas import, la cabecera de la definición de la clase, el(los) constructor(es), las variables de instancia o los métodos que pueden ser utilizados en la definición de la semántica de la gramática
    c. La tercera sección corresponde a la especificación léxica de la gramática
    d. Finalmente, la cuarte sección, corresponde a la descripción de las reglas sintácticas de la gramática
6. Al archivo javaCC creado en el paso 6 se le realizarán una serie de modificaciones. Primero, se define el nombre del analizador, como "NuevoParser", ya que el nombre establecido por defecto es "eg1", y se borra la función main que aparece en la plantilla inicialmente. De esta manera se obtiene algo similar a lo mostrado en la Figura 9. Si guardamos los cambios efectuados se generarán los siguientes archivos: "NuevoParser.java", "NuevoParserConstants.java", "NuevoParserTokenManager.java"; por lo que ya no serán necesarios los archivos creados inicialmente por el compilador: "eg1.java", "eg1Constants.java" y "eg1TokenManager.java", de manera que pueden ser borrados

```
/**
 * JavaCC template file created by SF JavaCC plugin 1.5.17+ wizard for JavaCC 1.5.0+
 */
options
{
  JDK_VERSION = "1.5";
  static = true;
}

PARSER_BEGIN(NuevoParser)
package uniandes.teolen.parserJavaCC.newParser;

public class NuevoParser
{

}

PARSER_END(NuevoParser)
```

Figura 9. Archivo javaCC asociado al nuevo parser

7. A continuación se define la especificación léxica del a gramática (ver Figura 10). Aquí se especifican los tokens que harán parte de la gramática (empleando declaraciones de tipo TOKEN), así como aquellos caracteres que serán filtrados por el analizador léxico, como los espacios y los finales de línea (esto se hace empleando declaraciones de tipo SKIP)

[1] Fuente: Universidad de Huelva, http://www.uhu.es/470004004/practicas/practica04.htm



```
//Especificación léxica
SKIP :
{
  " "
| "\r"
| "\t"
| "\n"
}

TOKEN : /* OPERATORS */
{
  < PLUS : "+" >
| < MINUS : "-" >
| < MULTIPLY : "*" >
| < DIVIDE : "/" >
}

TOKEN :
{
  < CONSTANT : (< DIGIT >)+ >
| < #DIGIT : [ "0"-"9" ] >
}
```

Figura 10. Especificación léxica, ubicada en el archivo javaCC del nuevo parser

8. Se define ahora la especificación sintáctica de la gramática (ver Figura 11). En esta sección son definidos todos los símbolos no terminales de la gramática (incluido el símbolo distinguido). Aquí se concluye la definición del archivo javaCC. Si se observa con detalle la definición léxica y sintáctica establecida por defecto al crear el archivo javaCC, se puede ver que corresponde a la gramática de una calculadora sencilla

```
int one_line() :
{}
{
  sum() ";"
  {
    return 0;
  }
| ";"
  {
    return 1;
  }
}

void sum() :
{}
{
  term()
  (
    (
      < PLUS >
    | < MINUS >
    )
    term()
  )*
}

void term() :
{}
{
  unary()
  (
    (
      < MULTIPLY >
    | < DIVIDE >
    )
    unary()
  )*
}

void unary() :
{}
{
  < MINUS > element()
| element()
}

void element() :
{}
{
  < CONSTANT >
| "(" sum() ")"
}
```

Figura 11. Especificación sintáctica de la gramática