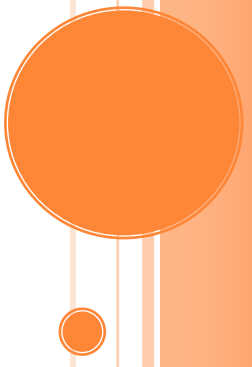
GRAPHIK Y HASKELL++

Introducción

Primer Proyecto 2S2017



Karen Melissa Lima Sandoval

201020210

Graphik y Haskell++

Introducción

Este manual está orientado a público con conocimientos técnicos de informática.

Actualmente existe una diversidad de paradigmas de programación, entre los cuales aparece la programación funcional, basado en el uso de funciones como objetos y que se ayuda también del paradigma de programación declarativa. Con el fin de aprender ambos paradigmas (funcional e imperativo) se requiere que el estudiante realice un conjunto de soluciones que proporcionen a un usuario la capacidad de realizar una unión entre ambos paradigmas, de modo que se implemente una programación funcional para crear módulos matemáticos y que luego por medio de una programación imperativa se generen diversas salidas haciendo uso de gráficas y texto.

El nombre del lenguaje que utiliza el paradigma funcional es Haskell+ + y el nombre del lenguaje que utiliza el paradigma imperativo es GraphiK. Para ambos lenguajes se requiere un mismo IDE que permita la facilidad de desarrollar el conjunto de funciones que un usuario desea. GraphiK podrá realizar llamadas a funciones de Haskell++, mientras que Haskell++ será únicamente para definir funciones propias.
Dentro del IDE se tendrá una terminal que funcionará como un intérprete que contendrá funciones predefinidas de Haskell++ y que podrá también manipular las funciones definidas por los usuarios mediante archivos. La terminal mencionada anteriormente también será la salida de los resultados obtenidos por GraphiK. Por el contrario de Haskell++, el lenguaje GraphiK será compilado al momento de su ejecución.

> Paradigma funcional es Haskell y el paradigma imperativo es Graphik

LIBRERIAS:

Las librerías utilizadas en el proyecto se explican a continuación:

- Jflex

  No es más que un generador de un analizador léxico parecido a LEX, el cual toma una cadena como entrada una cadena de caracteres, y lo convierte en una secuencia de tokens.

- Cup

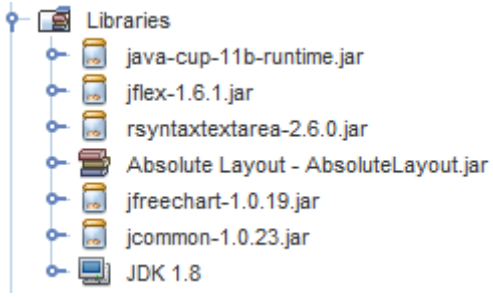
  Cup es un generador de analizadores sintácticos LALR en Java el cual recibe de entrada un archivo con la estructura de la gramática y su salida es un parser escrito en Java listo para usarse.

- JFreeChart

  Es un marco de software open source para el lenguaje de programación Java, el cual permite la creación de gráficos complejos de forma simple.

- RsyntaxtextArea

  Su función principal es darle características destacables a un área de texto, como colores a palabras y caracteres especiales, autocompletado, números de líneas, etc.



IDE

La herramienta que se utilizó como plataforma para este proyecto fue Netbeans en su versión 8.1

- Netbeans:

  es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

- Lenguaje Java:

  Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes.

GRAMÁTICA HASKELL++

```
Precedence left or;

Precedence left and;

Precedence left igualigual, diferente,
mayorq, menorq, mayorigualq,
menorigualq;

Precedence left mas, menos;

Precedence left por, div, residuo;

Precedence right pot, sqrt;


Start with INICIO;


INICIO ::= {:

        System.out.println("Empezo
Haskell\n");

      :}

       FUNCION:nodo

       {:

        System.out.println("Finalizo
Haskell\n");

        Parser.raiz = nodo;

       :}

  ;


FUNCION ::= FUNCION:nodo
L_FUNCION:fun

       {:

            Nodo.Hijo(fun);

            RESULT = nodo;

       :}

     | L_FUNCION:fun

       {:

          Nodo nuevo = new
Nodo("cuerpoarchivo");

            Nuevo.Hijo(fun);

            RESULT = nuevo;

       :}

     ;


L_FUNCION ::= id:name P:param igual
CUERPO:corpo fin

            {:

              Nodo nuevo = new
Nodo("Funcion");

            Nodo nombre = new
Nodo(name);

            Nodo cuerpo = new
Nodo("cuerpofuncion");

            Cuerpo.Hijo(corpo);

            Nuevo.Hijo(nombre);

            If(param!=null){

              Nuevo.Hijo(param);

            }
```

```
                Nuevo.Hijo(cuerpo);
                RESULT = nuevo;
            :}
        ;

P ::= PARAMETROS:nodo {:RESULT = nodo;:}
    |
    ;

PARAMETROS ::= PARAMETROS:param coma L_PARAM:p
            {:
                Param.Hijo(p);
                RESULT= param;
            :}
        | L_PARAM:exp
            {:
                Nodo nuevo = new Nodo("Parametros");
                Nuevo.Hijo(exp);
                RESULT = nuevo;
            :}
        ;

L_PARAM ::= EXP:exp
        {:
            RESULT = exp;
        :}
      | LISTA:lista
        {: RESULT = lista; :}
        ;

CUERPO ::= CUERPO:sent L_CUERPO:cuerpo
        {:
            Sent.Hijo(cuerpo);
            RESULT=sent;
        :}
      | L_CUERPO:sentencia
        {:
            Nodo nuevo = new Nodo("Sentencias");
            Nuevo.Hijo(sentencia);
            RESULT = nuevo;
        :}
      ;

L_CUERPO ::= IF:nodo
        {: RESULT = nodo; :}
       | CASE:nodo
        {: RESULT = nodo; :}
       | FUNCIONES_PROPIAS:nodo
        {: RESULT = nodo; :}
       ;

IF ::= si EXP:exp entonces CUERPO:corpoif sino CUERPO:corposino fin
      {:
        Nodo nuevo = new Nodo("If");
        Nodo novo = new Nodo("Else");
```

```
        Nodo cuerpo = new
Nodo("cuerpoif");

        Nodo cuerpo2 = new
Nodo("cuerposino");

        Nodo cond = new Nodo("Exp");

        Cond.Hijo(exp);

        Cuerpo.Hijo(corpoif);

        Cuerpo2.Hijo(corposino);

        Nuevo.Hijo(cond);

        Nuevo.Hijo(cuerpo);

        Nuevo.Hijo(cuerpo2);

        RESULT = nuevo;

      :}
   ;


CASE ::= caso EXP_CASO:exp
CASOS:corpo fin

      {:

        Nodo nuevo = new
Nodo("Case");

        Nuevo.Hijo(exp);

        Nuevo.Hijo(corpo);

        RESULT = nuevo;

      :}
    ;


CASOS ::= CASOS:caso L_CASOS:exp

      {:

        Caso.Hijo(exp);

        RESULT = caso;

      :}

      | L_CASOS:exp

      {:

        Nodo nuevo = new
Nodo("Lista_Casos");

        Nuevo.Hijo(exp);

        RESULT = nuevo;

      :}

     ;


L_CASOS ::= EXP_CASO:v dospuntos
CUERPO:corpo puntoycoma

        {:

          Nodo nuevo = new
Nodo("Caso");

          Nuevo.Hijo(v);

          Nuevo.Hijo(corpo);

          RESULT = nuevo;

        :}

      ;


EXP_CASO ::= EXP:exp

          {:  Nodo nuevo = new
Nodo("Exp");

            Nuevo.Hijo(exp);

            RESULT = nuevo;

          :}

        | LISTA:lista

          {:RESULT = lista;:}

        ;
```

```
FUNCIONES_PROPIAS ::=
DECLARA_LISTA:nodo
                {: RESULT = nodo; :}
              | CALCULAR:nodo
                {: RESULT = nodo; :}
              | SUCC:nodo
                {: RESULT = nodo; :}
              | DECC:nodo
                {: RESULT = nodo; :}
              | MIN:nodo
                {: RESULT = nodo; :}
              | MAX:nodo
                {: RESULT = nodo; :}
              | SUM:nodo
                {: RESULT = nodo; :}
              | PRODUCT:nodo
                {: RESULT = nodo; :}
              | REVERS:nodo
                {: RESULT = nodo; :}
              | IMPR:nodo
                {: RESULT = nodo; :}
              | PAR:nodo
                {: RESULT = nodo; :}
              | ASC:nodo
                {: RESULT = nodo; :}
              | DESC:nodo
                {: RESULT = nodo; :}
              | LENGTH:nodo
                {: RESULT = nodo; :}
              | CONCATENA:nodo
                {: RESULT = nodo; :}
              | INDICE:nodo
                {: RESULT = nodo; :}
              | LLAMA_FUNCION:nodo
                {: RESULT = nodo; :}
              ;

DECLARA_LISTA ::= dolar let id:nombre
igual CONCATENA:exp dolar
            {:
              Nodo nuevo = new
Nodo("D_Lista");
              Nodo name = new
Nodo(nombre);
              Nuevo.Hijo(name);
              Nuevo.Hijo(exp);
              RESULT = nuevo;
            :}
            ;

CALCULAR ::= dolar calcular:calcula
EXP:exp dolar
          {:
            Nodo cal = new
Nodo("Calcular");
            Nodo expresion = new
Nodo("Exp");
            Expresion.Hijo(exp);
            Cal.Hijo(expresion);
            RESULT = cal;
          :}
```

```
        ;

SUCC ::= dolar succ EXP:exp dolar
      {:
         Nodo nuevo = new
Nodo("Succ");
         Nodo lista = new Nodo("Exp");
         Lista.Hijo(exp);
         Nuevo.Hijo(lista);
         RESULT = nuevo;
      :}
    ;

DECC ::= dolar decc EXP:exp dolar
      {:
         Nodo nuevo = new
Nodo("Decc");
         Nodo lista = new Nodo("Exp");
         Lista.Hijo(exp);
         Nuevo.Hijo(lista);
         RESULT = nuevo;
      :}
    ;

MIN ::= dolar min M:exp dolar
      {:
         Nodo nuevo = new Nodo("Min");
         Nuevo.Hijo(exp);
         RESULT = nuevo;
      :}
```

```
    ;

MAX ::= dolar max M:exp dolar
      {:
         Nodo nuevo = new
Nodo("Max");
         Nuevo.Hijo(exp);
         RESULT = nuevo;
      :}
    ;

M ::= id:id
     {: Nodo nuevo = new Nodo("id");
        Nodo i = new Nodo(id);
        Nuevo.Hijo(i);
        RESULT = nuevo; :}
   | LISTA:lista
     {: RESULT = lista; :}
   ;

SUM ::= dolar sum CONCATENA:exp
dolar
      {:
         Nodo nuevo = new
Nodo("Sum");
         Nuevo.Hijo(exp);
         RESULT = nuevo;
      :}
    ;
```

```
PRODUCT ::= dolar product CONCATENA:exp dolar
        {:
            Nodo nuevo = new Nodo("Product");
            Nuevo.Hijo(exp);
            RESULT = nuevo;
        :}
        ;

REVERS ::= dolar revers CONCATENA:exp dolar
        {:
            Nodo nuevo = new Nodo("Revers");
            Nuevo.Hijo(exp);
            RESULT = nuevo;
        :}
        ;

IMPR ::= dolar impr CONCATENA:exp dolar
        {:
            Nodo nuevo = new Nodo("Impr");
            Nuevo.Hijo(exp);
            RESULT = nuevo;
        :}
      ;

PAR ::= dolar par CONCATENA:exp dolar
        {:
            Nodo nuevo = new Nodo("Par");
            Nuevo.Hijo(exp);
            RESULT = nuevo;
        :}
      ;

ASC ::= dolar asc CONCATENA:exp dolar
        {:
            Nodo nuevo = new Nodo("Asc");
            Nuevo.Hijo(exp);
            RESULT = nuevo;
        :}
      ;

DESC ::= dolar desc CONCATENA:exp dolar
        {:
            Nodo nuevo = new Nodo("Desc");
            Nuevo.Hijo(exp);
            RESULT = nuevo;
        :}
      ;

LENGTH ::= dolar tam CONCATENA:exp dolar
        {:
            Nodo nuevo = new Nodo("Length");
```

```
            Nuevo.Hijo(exp);
            RESULT = nuevo;
        :}
        ;

CONCATENA ::= CONCATENA:c concatena C_L:l
            {:
                C.Hijo(l);
                RESULT = c;
            :}
            | C_L:l
            {:
                Nodo lista = new Nodo("Concatena");
                Lista.Hijo(l);
                RESULT = lista;
            :}
            ;

C_L ::= LISTA:lista
        {:RESULT = lista;:}
      | id:id
        {:  Nodo nuevo = new Nodo("id");
            Nodo i = new Nodo(id);
            Nuevo.Hijo(i);
            RESULT = nuevo; :}
      | cadena:cad
        {:  Nodo nuevo = new Nodo("cadena");
            Nodo n = new Nodo(cad.tostring().replace("\"", ""));
            Nuevo.Hijo(n); RESULT = nuevo; :}
      ;

INDICE ::= id:id indice EXP_INDICE:exp L_INDICE:otro
            {:
                Nodo nuevo = new Nodo("Indice");
                Nodo nombre = new Nodo(id);
                Nodo e = new Nodo("Exp");
                E.Hijo(exp);
                Nuevo.Hijo(nombre);
                Nuevo.Hijo(e);
                If (otro!=null){
                    Nuevo.Hijo(otro);
                }
                RESULT = nuevo;
            :}
        ;

L_INDICE ::= indice EXP_INDICE:exp
            {:
                Nodo nuevo = new Nodo("Exp");
                Nuevo.Hijo(exp);
                RESULT = nuevo;
            :}
          | //{RESULT = null;}
```

```
        ;

EXP_INDICE ::= EXP_INDICE:j mas EXP_INDICE:m
      {:
         Nodo nuevo = new Nodo("+");
         Nuevo.Hijo(j);
         Nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP_INDICE:j menos EXP_INDICE:m
      {:
         Nodo nuevo = new Nodo("-");
         Nuevo.Hijo(j);
         Nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP_INDICE:j por EXP_INDICE:m
      {:
         Nodo nuevo = new Nodo("*");
         Nuevo.Hijo(j);
         Nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP_INDICE:j div EXP_INDICE:m
      {:
         Nodo nuevo = new Nodo("/");
         Nuevo.Hijo(j);
         Nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP_INDICE:j residuo EXP_INDICE:m
      {:
         Nodo nuevo = new
Nodo("mod");
         Nuevo.Hijo(j);
         Nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP_INDICE:j sqrt EXP_INDICE:m
      {:
         Nodo nuevo = new Nodo("sqrt");
         Nuevo.Hijo(j);
         Nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP_INDICE:j pot EXP_INDICE:m
      {:
         Nodo nuevo = new Nodo("pot");
         Nuevo.Hijo(j);
         Nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | SUM:nodo
      {: RESULT = nodo; :}
    | PRODUCT:nodo
      {: RESULT = nodo; :}
    | LENGTH:nodo
```

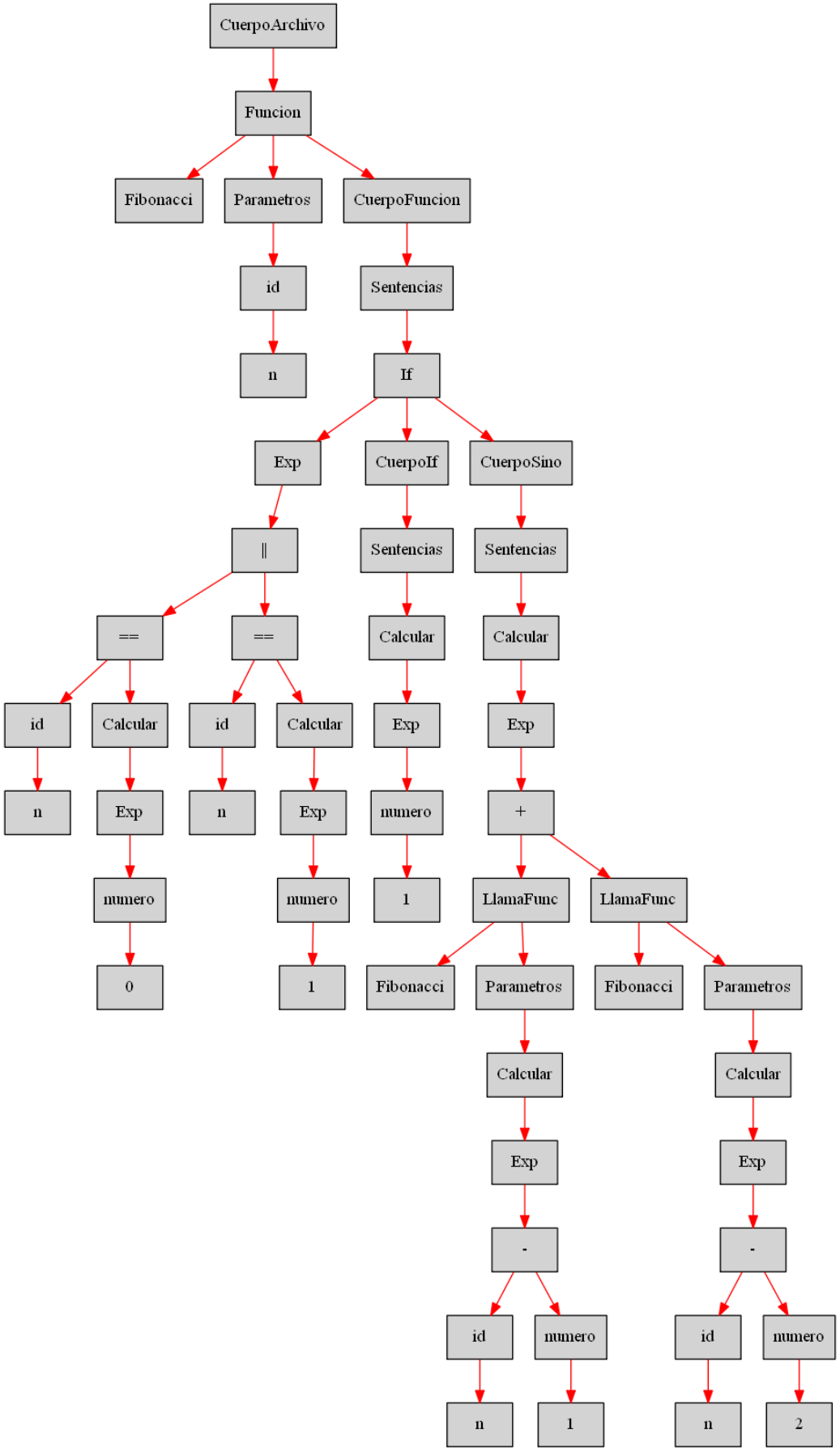
```
        {: RESULT = nodo; :}

    | menos EXP_INDICE:nodo

        {:

            Nodo nuevo = new
Nodo("Unario");

            Nuevo.Hijo(nodo);

            RESULT = nuevo;

        :}

    | parenabre EXP_INDICE:nodo
parencierra

        {: RESULT = nodo; :}

    | CALCULAR:nodo

        {: RESULT = nodo; :}

    | LLAMA_FUNCION:nodo

        {: RESULT = nodo; :}

    | SUCC:nodo

        {: RESULT = nodo; :}

    | DECC:nodo

        {: RESULT = nodo; :}

    | MIN:nodo

        {: RESULT = nodo; :}

    | MAX:nodo

        {: RESULT = nodo; :}

    | entero:num

        {:  Nodo nuevo = new
Nodo("numero");

            Nodo n = new Nodo(num);

            Nuevo.Hijo(n);

            RESULT = nuevo; :}

    | cadena:cad

        {:  Nodo nuevo = new
Nodo("cadena");

            Nodo n = new
Nodo(cad.tostring().replace("\"", ""));

            Nuevo.Hijo(n);

            RESULT = nuevo; :}


    | caracter:letra

        {:  Nodo nuevo = new
Nodo("caracter");

            Nodo l = new Nodo(letra);

            Nuevo.Hijo(l);

            RESULT = nuevo; :}

    | id:id

        {:  Nodo nuevo = new Nodo("id");

            Nodo i = new Nodo(id);

            Nuevo.Hijo(i);

            RESULT = nuevo; :}


LISTA ::= corabre LISTAS:listas corcierra

            {: RESULT = listas; :}

        ;


LISTAS ::= EXPRESIONES:exp

            {:RESULT = exp; :}

        | MAS_CORCH:lista

            {:RESULT = lista; :}


        ;
```

```
EXPRESIONES ::= EXPRESIONES:lista
coma EXP:exp
            {:
               Lista.Hijo(exp);
               RESULT = lista;
            :}
          | EXP:exp
            {:
               Nodo lista = new
Nodo("Lista");
               Lista.Hijo(exp);
               RESULT = lista;
            :}
          ;


MAS_CORCH ::= MAS_CORCH:nodo
coma DIM:dim
            {:
               Nodo.Hijo(dim);
               RESULT = nodo;
            :}
          | DIM:nodo
            {:Nodo nivel = new
Nodo("2Niveles");
               Nivel.Hijo(nodo);
               RESULT = nivel;
            :}
          ;


DIM ::= corabre EXPRESIONES:nodo
corcierra
            {:
               RESULT =nodo;
            :}
          ;
EXP ::= EXP:j mas EXP:m
            {:
               Nodo nuevo = new Nodo("+");
               Nuevo.Hijo(j);
               Nuevo.Hijo(m);
               RESULT = nuevo;
            :}
          | EXP:j menos EXP:m
            {:
               Nodo nuevo = new Nodo("-");
               Nuevo.Hijo(j);
               Nuevo.Hijo(m);
               RESULT = nuevo;
            :}
          | EXP:j por EXP:m
            {:
               Nodo nuevo = new Nodo("*");
               Nuevo.Hijo(j);
               Nuevo.Hijo(m);
               RESULT = nuevo;
            :}
          | EXP:j div EXP:m
            {:
               Nodo nuevo = new Nodo("/");
               Nuevo.Hijo(j);
```

```
            Nuevo.Hijo(m);
            RESULT = nuevo;
        :}
    | EXP:j residuo EXP:m
        {:
            Nodo nuevo = new
Nodo("mod");
            Nuevo.Hijo(j);
            Nuevo.Hijo(m);
            RESULT = nuevo;
        :}
    | EXP:j sqrt EXP:m
        {:
            Nodo nuevo = new Nodo("sqrt");
            Nuevo.Hijo(j);
            Nuevo.Hijo(m);
            RESULT = nuevo;
        :}
    | EXP:j pot EXP:m
        {:
            Nodo nuevo = new Nodo("pot");
            Nuevo.Hijo(j);
            Nuevo.Hijo(m);
            RESULT = nuevo;
        :}
    | EXP:j or EXP:m
        {:
            Nodo nuevo = new Nodo("||");
            Nuevo.Hijo(j);
            Nuevo.Hijo(m);
            RESULT = nuevo;
        :}
    | EXP:j and EXP:m
        {:
            Nodo nuevo = new Nodo("&&");
            Nuevo.Hijo(j);
            Nuevo.Hijo(m);
            RESULT = nuevo;
        :}
    | EXP:j menorq EXP:m
        {:
            Nodo nuevo = new Nodo("<");
            Nuevo.Hijo(j);
            Nuevo.Hijo(m);
            RESULT = nuevo;
        :}
    | EXP:j mayorq EXP:m
        {:
            Nodo nuevo = new Nodo(">");
            Nuevo.Hijo(j);
            Nuevo.Hijo(m);
            RESULT = nuevo;
        :}
    | EXP:j menorigualq EXP:m
        {:
            Nodo nuevo = new Nodo("<=");
            Nuevo.Hijo(j);
            Nuevo.Hijo(m);
```

```
        RESULT = nuevo;
    :}
  | EXP:j mayorigualq EXP:m
    {:
        Nodo nuevo = new Nodo(">=");
        Nuevo.Hijo(j);
        Nuevo.Hijo(m);
        RESULT = nuevo;
    :}
  | EXP:j igualigual EXP:m
    {:
        Nodo nuevo = new Nodo("==");
        Nuevo.Hijo(j);
        Nuevo.Hijo(m);
        RESULT = nuevo;
    :}
  | EXP:j diferente EXP:m
    {:
        Nodo nuevo = new Nodo("!=");
        Nuevo.Hijo(j);
        Nuevo.Hijo(m);
        RESULT = nuevo;
    :}
  | SUM:nodo
    {: RESULT = nodo; :}
  | PRODUCT:nodo
    {: RESULT = nodo; :}
  | LENGTH:nodo
    {: RESULT = nodo; :}
  | menos EXP:nodo
    {:
        Nodo nuevo = new
Nodo("Unario");
        Nuevo.Hijo(nodo);
        RESULT = nuevo;
    :}
  | parenabre EXP:nodo parencierra
    {: RESULT = nodo; :}
  | CALCULAR:nodo
    {: RESULT = nodo; :}
  | LLAMA_FUNCION:nodo
    {: RESULT = nodo; :}
  | INDICE:nodo
    {: RESULT = nodo; :}
  | SUCC:nodo
    {: RESULT = nodo; :}
  | DECC:nodo
    {: RESULT = nodo; :}
  | MIN:nodo
    {: RESULT = nodo; :}
  | MAX:nodo
    {: RESULT = nodo; :}
  | entero:num
    {:  Nodo nuevo = new
Nodo("numero");
        Nodo n = new Nodo(num);
        Nuevo.Hijo(n);
        RESULT = nuevo; :}
  | cadena:cad
```

```
        {:  Nodo nuevo = new
Nodo("cadena");

            Nodo n = new
Nodo(cad.tostring().replace("\"", ""));

            Nuevo.Hijo(n); RESULT = nuevo; :}

    | caracter:letra

        {:  Nodo nuevo = new
Nodo("caracter");

            Nodo l = new Nodo(letra);

            Nuevo.Hijo(l);

            RESULT = nuevo; :}

    | id:id

        {:  Nodo nuevo = new Nodo("id");

            Nodo i = new Nodo(id);

            Nuevo.Hijo(i);

            RESULT = nuevo; :}

    ;


LLAMA_FUNCION ::= DOLAR ID:ID LLAVEABRE
P:PARAM LLAVECIERRA DOLAR

            {:

                NODO NUEVO = NEW
NODO("LLAMAFUNC");

                NODO NOMBRE = NEW NODO(ID);

                NUEVO.HIJO(NOMBRE);

                NUEVO.HIJO(PARAM);

                RESULT = NUEVO
```


CuerpoArchivo
Funcion
Fibonacci
Parametros
CuerpoFuncion
id
Sentencias
n
If
Exp
CuerpoIf
CuerpoSino
||
Sentencias
Sentencias
==
==
Calcular
Calcular
id
Calcular
id
Calcular
Exp
Exp
n
Exp
n
Exp
numero
+
numero
numero
1
LlamaFunc
LlamaFunc
0
1
Fibonacci
Parametros
Fibonacci
Parametros
Calcular
Calcular
Exp
Exp
-
-
id
numero
id
numero
n
1
n
2

GRAMÁTICA GRAPHIK

```
INICIO ::= {:System.out.println("Empezo
Graphik \n");:}IM:a IN:b L_ALS:c

        {:Nodo nuevo = new
Nodo("Raiz");

            nuevo.Hijo(a);

            nuevo.Hijo(b);

            nuevo.Hijo(c);

            RESULT = nuevo;

            System.out.println("Finalizo
Graphik \n");

            parser.raiz=nuevo;

        :}

    ;

IM ::= IMPORTACIONES:nodo {:RESULT =
nodo; :}

  | {:RESULT = new Nodo("Importa");:}

  ;


IMPORTACIONES ::=
IMPORTACIONES:importa
L_IMPORTACIONES:nodo
{:importa.Hijo(nodo); RESULT =importa;:}

            | L_IMPORTACIONES:nodo

              {:Nodo nuevo = new
Nodo("Importa");
nuevo.Hijo(nodo);RESULT=nuevo;:}

            ;


L_IMPORTACIONES ::= importar
nombreArchivo:id fin {:RESULT = new
Nodo(id);:}

            ;


IN ::= INCLUYE_HK:nodo {:RESULT =
nodo; :}

  | {:RESULT = new Nodo("Incluye");:}

  ;


INCLUYE_HK ::= INCLUYE_HK:incluye
L_INCLUYE:nodo {:incluye.Hijo(nodo);
RESULT =incluye;:}

        | L_INCLUYE:nodo

          {:Nodo nuevo = new
Nodo("Incluye");
nuevo.Hijo(nodo);RESULT=nuevo;:}

        ;


L_INCLUYE ::= incluirHK id:id fin {:RESULT
= new Nodo(id);:}

        ;


L_ALS ::= L_ALS:als ALS:nodo {:
als.Hijo(nodo); RESULT = als; :}

      | ALS:nodo {:Nodo nuevo = new
Nodo("ListaAls"); nuevo.Hijo(nodo);
RESULT = nuevo;:}

      ;


ALS ::= Tals id:id VISIBILIDAD:v HEREDA:h
llaveAbre CUERPO_ALS:corpo
llaveCierra

      {:

        Nodo nuevo = new Nodo("Als");

        Nodo i = new Nodo(id);
```

```
            nuevo.Hijo(i);

            nuevo.Hijo(v);

            nuevo.Hijo(h);

            nuevo.Hijo(corpo);

            RESULT = nuevo;

        :}

    ;


HEREDA ::= hereda id:id

        {:

            Nodo nuevo = new
Nodo("Hereda");

            Nodo i = new Nodo(id);

            nuevo.Hijo(i);

            RESULT = nuevo;

        :}

        |{:RESULT = new Nodo("Hereda");:}

        ;


VISIBILIDAD ::= dospuntos publico
{:RESULT = new Nodo("Publico");:}

            | dospuntos privado {:RESULT =
new Nodo("Privado");:}

            | dospuntos protegido {:RESULT
= new Nodo("Protegido");:}

            | {:RESULT = new
Nodo("Publico");:}

            ;


CUERPO_ALS ::= CUERPO_ALS:als
L_CUERPO:nodo

                {:als.Hijo(nodo); RESULT = als;:}

            | L_CUERPO:nodo

                {:Nodo nuevo = new
Nodo("SentenciasAls");

                    nuevo.Hijo(nodo);

                    RESULT = nuevo;

                :}

            ;


L_CUERPO ::=
DECLARACION_GLOBAL:nodo
{:RESULT = nodo;:}

            | METODO_INICIO:nodo
{:RESULT = nodo;:}

            | DATOS:nodo                {:RESULT
= nodo;:}

            | METODOS:nodo
{:RESULT = nodo;:}

            ;


DATOS ::= Tvacio datos parenAbre
parenCierra llaveAbre
CUERPO_DATOS:corpo llaveCierra

            {:

                Nodo nuevo = new
Nodo("MetodoDatos");

                Nodo tipo = new
Nodo("TipoVacio");

                nuevo.Hijo(tipo);

                nuevo.Hijo(corpo);

                RESULT = nuevo;

            :}

        ;
```

```
CUERPO_DATOS ::=
CUERPO_DATOS:datos L_DATOS:nodo
                {:datos.Hijo(nodo); RESULT =
datos;:}
            | L_DATOS:nodo
             {:
                Nodo nuevo = new
Nodo("CuerpoDatos");
                nuevo.Hijo(nodo);
                RESULT = nuevo;
             :}
           ;
L_DATOS ::= PROCESAR:nodo    {:RESULT
= nodo;:}
        | DONDE:nodo       {:RESULT =
nodo;:}
        | DONDECADA:nodo   {:RESULT =
nodo;:}
        | DONDETODO:nodo   {:RESULT =
nodo;:}
          ;

COLUMNA ::= columna parenAbre
EXP:exp parenCierra
          {:
             Nodo nuevo = new
Nodo("Columna");
             Nodo e = new Nodo("Exp");
             e.Hijo(exp);
             nuevo.Hijo(e);
             RESULT = nuevo;
          :}
        ;

PROCESAR ::= procesar igual EXP:exp
fin
          {:
             Nodo nuevo = new
Nodo("Procesar");
             Nodo e = new Nodo("Exp");
             e.Hijo(exp);
             nuevo.Hijo(e);
             RESULT = nuevo;
          :}
        ;
DONDE ::= donde parenAbre EXP:exp
parenCierra igual EXP:exp2 fin
          {:
             Nodo nuevo = new
Nodo("Donde");
             Nodo e = new Nodo("Exp");
             Nodo e2 = new Nodo("Exp");
             e.Hijo(exp);
             e2.Hijo(exp2);
             nuevo.Hijo(e);
             nuevo.Hijo(e2);
             RESULT = nuevo;
          :}
      ;
DONDECADA ::= dondecada
parenAbre EXP:exp parenCierra fin
          {:
             Nodo nuevo = new
Nodo("DondeCada");
```

```
            Nodo e = new Nodo("Exp");
            e.Hijo(exp);
            nuevo.Hijo(e);
            RESULT = nuevo;
        :};


DONDETODO ::= dondetodo
parenAbre EXP:exp parenCierra fin
        {:
            Nodo nuevo = new
Nodo("DondeTodo");
            Nodo e = new Nodo("Exp");
            e.Hijo(exp);
            nuevo.Hijo(e);
            RESULT = nuevo;
        :}
        ;


DECLARACION_GLOBAL ::= var
TIPO_DATO:t id:id VISIBILIDAD:v
OPCION:op fin
            {: Nodo nuevo = new
Nodo("DeclaraGlobalVariable");
            nuevo.Hijo(t);
            if(op.valor.equals("ListaID")){
                Nodo l= new Nodo("L");
                Nodo i= new Nodo(id);
                l.Hijo(i);
                l.Hijo(v);
                op.Hijo(l);
                nuevo.Hijo(op);
            }else {
                Nodo m = new
Nodo("ListaID");
                Nodo mm = new Nodo("L");
                Nodo i = new Nodo(id);
                m.Hijo(mm);
                mm.Hijo(i);
                mm.Hijo(v);
                nuevo.Hijo(m);
                nuevo.Hijo(op);

            }
            RESULT = nuevo;
            :}
        | var TIPO_DATO:t id:id
ARREGLO:a VISIBILIDAD:v
ASIGNACION_ARREGLO:nodo fin
            {: Nodo nuevo = new
Nodo("DeclaraGlobalArreglo");
            Nodo l= new Nodo("ListaID");
            Nodo i= new Nodo(id);
            l.Hijo(i);
            l.Hijo(v);
            nuevo.Hijo(t);
            nuevo.Hijo(l);
            nuevo.Hijo(a);
            nuevo.Hijo(nodo);
            RESULT = nuevo;
            :}
        | var TIPO_DATO:t id:id
VISIBILIDAD:v igual nuevo
LLAMAR_INSTANCIA:nodo fin
```

```
            {: Nodo nuevo = new
Nodo("InstanciaGlobal");

            Nodo l= new Nodo("ListaID");

            Nodo as= new
Nodo("Objeto");

            Nodo i= new Nodo(id);

            l.Hijo(i);

            l.Hijo(v);

            as.Hijo(nodo);

            nuevo.Hijo(t);

            nuevo.Hijo(l);

            nuevo.Hijo(as);

            RESULT = nuevo;

            :} ;

OPCION ::= coma L_VAR:nodo {:RESULT
= nodo;:}

      | ASIGNACION:nodo {:Nodo n=
new Nodo("Asignacion");
n.Hijo(nodo);RESULT = n;:}

      | {:RESULT = new
Nodo("Asignacion");:}

      ;

L_VAR ::= L_VAR:nodo coma V:v

         {: v.setNombre("L");

         nodo.Hijo(v);

         RESULT = nodo;

         :}

      | V:v

      {:v.setNombre("L");

         Nodo nuevo = new
Nodo("ListaID");

      nuevo.Hijo(v);

         RESULT = nuevo;

         :}

      ;

V ::= id:id VISIBILIDAD:nodo

         {:Nodo nuevo = new Nodo("");

         Nodo i= new Nodo(id);

            nuevo.Hijo(i);

            nuevo.Hijo(nodo);

         RESULT = nuevo;

         :}

      ;

ARREGLO ::= ARREGLO:arr ARR:a

            {:

               arr.Hijo(a);

               RESULT=arr;

            :}

      | ARR:a

      {:Nodo nuevo = new
Nodo("Dimensiones");

      nuevo.Hijo(a);

      RESULT = nuevo;

      :}

      ;

ARR ::= corAbre EXP:exp corCierra

            {:Nodo nuevo = new
Nodo("Exp");

               nuevo.Hijo(exp);

               RESULT=nuevo;

            :}

      ;
```

```
TIPO_DATO ::= Tentero
{:      RESULT = new Nodo("numero");
        :}
        | Tdecimal
{:
        RESULT= new Nodo("decimal");
        :}
        | Tcaracter
{:
        RESULT = new Nodo("caracter");
        :}
        | Tcadena
{:
        RESULT = new Nodo("cadena");
        :}
        | Tbool
{:
        RESULT= new Nodo("bool");
        :}
        | Tvacio
{:
        RESULT = new Nodo("vacio");
        :}
        | id:id
        {:
        Nodo nuevo = new Nodo("AIs");
        Nodo i = new Nodo(id);
        nuevo.Hijo(i);
        RESULT = nuevo; :}
                ;
ASIGNACION_ARREGLO ::= igual
EXP:nodo {:RESULT = nodo;:}
                | {:RESULT = new
Nodo("Posiciones");:}
                ;
L_POSICIONES ::= L_POSICIONES:p
coma EXP:nodo
                {:p.Hijo(nodo); RESULT = p;:}
                | EXP:nodo
                {:Nodo nuevo = new
Nodo("Posiciones");
                nuevo.Hijo(nodo);
                RESULT = nuevo;:}
                ;
ASIGNACION ::= igual EXP:exp
   {:Nodo nuevo = new Nodo("Exp");
nuevo.Hijo(exp); RESULT = nuevo;:}
                ;
METODO_INICIO ::= Tvacio inicio
parenAbre parenCierra llaveAbre
SENTENCIAS:sent llaveCierra
                {:
                Nodo nuevo = new
Nodo("MetodoInicio");
                Nodo corpo = new
Nodo("CuerpoInicio");
                Nodo tipo = new
Nodo("TipoVacio");
                corpo.Hijo(sent);
                nuevo.Hijo(tipo);
                nuevo.Hijo(corpo);
```

```
            RESULT = nuevo;
            :}
            ;

METODOS ::= TIPO_DATO:t id:id
parenAbre L_PARAMETROS:p
parenCierra VISIBILIDAD:v llaveAbre
SENTENCIAS:s llaveCierra
        {:
            Nodo nuevo = new
Nodo("Metodo");
            Nodo corpo = new
Nodo("CuerpoMetodo");
            Nodo i = new Nodo(id);
            corpo.Hijo(s);
            nuevo.Hijo(t);
            nuevo.Hijo(i);
            nuevo.Hijo(p);
            nuevo.Hijo(v);
            nuevo.Hijo(corpo);
            RESULT = nuevo;
        :}
      ;

L_PARAMETROS ::= PARAMETROS:nodo
{:RESULT = nodo;:}
          | {:RESULT = new
Nodo("Parametros");:}
          ;

PARAMETROS ::= PARAMETROS:p coma
PP:pp
        {:
            pp.setNombre("P");
            p.Hijo(pp);
            RESULT = p;
        :}
        | PP:pp
        {:
            pp.setNombre("P");
            Nodo nuevo = new
Nodo("Parametros");
            nuevo.Hijo(pp);
            RESULT = nuevo;
        :}
        ;

PP ::= TIPO_DATO:t id:id
        {:
            Nodo nuevo = new Nodo("");
            Nodo i = new Nodo(id);
            nuevo.Hijo(t);
            nuevo.Hijo(i);
            RESULT = nuevo;
        :}
        ;

SENTENCIAS ::= SENTENCIAS:sent
L_SENT:nodo {:sent.Hijo(nodo); RESULT =
sent;:}
        | L_SENT:nodo
            {:Nodo n = new
Nodo("Sentencias"); n.Hijo(nodo);
RESULT = n;:}
        ;
```

```
L_SENT ::= DECLARACION_LOCAL:nodo
fin      {:RESULT = nodo;:}

       | LLAMAR:nodo fin
{:RESULT = nodo;:}

       | LLAMARHK:nodo fin
{:RESULT = nodo;:}

       | llamar datos parenAbre
parenCierra fin    {:RESULT = new
Nodo("LlamaDatos");:}

       | ASIGNA:nodo fin
{:RESULT = nodo;:}

       | RETORNA:nodo
{:RESULT = nodo;:}

       | SENTENCIA_SI:nodo
{:RESULT = nodo;:}

       | SENTENCIA_SELECCION:nodo
{:RESULT = nodo;:}

       | SENTENCIA_PARA:nodo
{:RESULT = nodo;:}

       | SENTENCIA_MIENTRAS:nodo
{:RESULT = nodo;:}

       | SENTENCIA_HMIENTRAS:nodo
{:RESULT = nodo;:}

       | CONTINUAR:nodo
{:RESULT = nodo;:}

       | TERMINAR:nodo
{:RESULT = nodo;:}

       | IMPRIMIR:nodo
{:RESULT = nodo;:}

       | GRAPHIKAR:nodo
{:RESULT = nodo;:}

       | LLAMADA_ARREGLO:nodo igual
EXP:exp fin

          {:

             Nodo nuevo = new
Nodo("AsignaPosicion");

             nuevo.Hijo(nodo);

             nuevo.Hijo(exp);

             RESULT = nuevo;

          :}

       | INCDEC:nodo incremento fin

          {:

             Nodo nuevo = new
Nodo("Incremento");

             nuevo.Hijo(nodo);

             RESULT = nuevo;

          :}

       | INCDEC:nodo decremento fin

          {:

          Nodo nuevo = new
Nodo("Decremento");

          nuevo.Hijo(nodo);

          RESULT = nuevo;

          :}

       ;

INCDEC ::= num:num

       {:  Nodo nuevo = new
Nodo("numero");

          Nodo n = new Nodo(num);

          nuevo.Hijo(n);

          RESULT = nuevo; :}

       | dec:num

       {:  Nodo nuevo = new
Nodo("decimal");

          Nodo n = new Nodo(num);

          nuevo.Hijo(n);

          RESULT = nuevo; :}

       | id:num
```

```
        {:  Nodo nuevo = new Nodo("id");
            Nodo n = new Nodo(num);
            nuevo.Hijo(n);
            RESULT = nuevo; :}
        | carac:num
        {:  Nodo nuevo = new
Nodo("caracter");
            Nodo n = new Nodo(num);
            nuevo.Hijo(n);
            RESULT = nuevo; :}
        ;

GRAPHIKAR ::= graphikar parenAbre
ARR_GRAPHIK:x coma ARR_GRAPHIK:y
parenCierra fin
            {:Nodo nuevo = new
Nodo("Graphikar");
                nuevo.Hijo(x);
                nuevo.Hijo(y);
                RESULT = nuevo;
            :}
            ;

ARR_GRAPHIK ::= EXP:exp {:Nodo
nuevo = new Nodo("Exp");
nuevo.Hijo(exp); RESULT = nuevo;:}
                ;

IMPRIMIR ::= imprimir parenAbre
EXP:exp parenCierra fin
            {:Nodo nuevo = new
Nodo("Imprimir");
            Nodo e = new Nodo("Exp");
            e.Hijo(exp);
            nuevo.Hijo(e);
            RESULT = nuevo;
            :}
            ;

CONTINUAR ::= continuar fin
{:RESULT = new Nodo("Continuar");:}
            ;

TERMINAR ::= terminar fin
{:RESULT = new Nodo("Terminar");:}
            ;

SENTENCIA_HMIENTRAS ::= hacer
llaveAbre SENTENCIAS:sent llaveCierra
mientras parenAbre EXP:cond
parenCierra fin
                {:Nodo nuevo = new
Nodo("SentenciaHacer");
                    Nodo e = new
Nodo("Exp");
                    Nodo s = new
Nodo("CuerpoHacer");
                    s.Hijo(sent);
                    e.Hijo(cond);
                    nuevo.Hijo(s);
                    nuevo.Hijo(e);
                    RESULT = nuevo;
                :}
                ;

SENTENCIA_MIENTRAS ::= mientras
parenAbre EXP:cond parenCierra
llaveAbre SENTENCIAS:sent llaveCierra
                {:Nodo nuevo = new
Nodo("SentenciaMientras");
                    Nodo e = new
Nodo("Exp");
```

```
                    Nodo s = new
Nodo("CuerpoMientras");

                    s.Hijo(sent);

                    e.Hijo(cond);

                    nuevo.Hijo(e);

                    nuevo.Hijo(s);

                    RESULT = nuevo;

                :}

            ;

SENTENCIA_PARA ::= para parenAbre
ASIGNA_PARA:a EXP:cond dospuntos
EXP:exp parenCierra llaveAbre
SENTENCIAS:s llaveCierra

          {:Nodo nuevo = new
Nodo("SentenciaPara");

                    Nodo e = new
Nodo("Exp");

                    Nodo e2 = new
Nodo("Exp");

                    Nodo sent = new
Nodo("CuerpoPara");

                    sent.Hijo(s);

                    e.Hijo(cond);

                    e2.Hijo(exp);

                    nuevo.Hijo(a);

                    nuevo.Hijo(e);

                    nuevo.Hijo(e2);

                    nuevo.Hijo(sent);

                    RESULT = nuevo;

                :}

          ;
```

```
ASIGNA_PARA ::=
DECLARACION_PARA:nodo dospuntos
{:RESULT = nodo;:}

          | ASIGNA:nodo dospuntos
{:RESULT = nodo;:}

          ;


DECLARACION_PARA ::= var
TIPO_DATO:t id:id igual EXP:exp

          {: Nodo nuevo = new
Nodo("DeclaraLocalVariable");

          Nodo l= new Nodo("ListaID");

          Nodo t2= new
Nodo("Publico");

          Nodo l2= new Nodo("L");

          Nodo e= new Nodo("Exp");

          Nodo i= new Nodo(id);

          l.Hijo(l2);

          l2.Hijo(i);

          l2.Hijo(t2);

          e.Hijo(exp);

          nuevo.Hijo(t);

          nuevo.Hijo(l);

          nuevo.Hijo(e);

          RESULT = nuevo;

          :}

            ;


SENTENCIA_SELECCION ::= seleccion
parenAbre EXP:exp parenCierra
llaveAbre CASOS:casos DEFECTO:def
llaveCierra
```

```
                {:Nodo nuevo = new
Nodo("SentenciaSeleccion");

                    Nodo e = new
Nodo("Exp");

                    e.Hijo(exp);

                    nuevo.Hijo(e);

                    nuevo.Hijo(casos);

                    nuevo.Hijo(def);

                    RESULT = nuevo;

                :}

            ;
```

```
CASOS ::= CASOS:caso L_CASO:nodo
{:caso.Hijo(nodo); RESULT = caso;:}

    | L_CASO:nodo {:Nodo nuevo =
new Nodo("ListaCasos");
nuevo.Hijo(nodo);RESULT = nuevo;:}

    ;
```

```
L_CASO ::= caso EXP:exp dospuntos
SENTENCIAS:sent

        {:Nodo nuevo = new
Nodo("Caso");

          Nodo e = new Nodo("Exp");

          Nodo c = new
Nodo("CuerpoCaso");

          e.Hijo(exp);

          c.Hijo(sent);

          nuevo.Hijo(e);

          nuevo.Hijo(c);

          RESULT = nuevo;

          :}
```

```
    ;


DEFECTO ::= defecto dospuntos
SENTENCIAS:sent

        {:Nodo nuevo = new
Nodo("CuerpoDefecto");
nuevo.Hijo(sent); RESULT = nuevo;:}

      | {:RESULT = new
Nodo("CuerpoDefecto");:}

      ;
```

```
SENTENCIA_SI ::= si parenAbre EXP:exp
parenCierra llaveAbre
SENTENCIAS:corpo llaveCierra
SENTENCIA_SINO:corposino

            {:

                Nodo nuevo = new
Nodo("SentenciaSi");

                Nodo e= new Nodo("Exp");

                Nodo si= new
Nodo("CuerpoSi");

                si.Hijo(corpo);

                e.Hijo(exp);

                nuevo.Hijo(e);

                nuevo.Hijo(si);

                nuevo.Hijo(corposino);

                RESULT = nuevo;

              :}

            ;
```

```
SENTENCIA_SINO ::= sino llaveAbre
SENTENCIAS:sent llaveCierra
```

```
            {:Nodo nuevo = new
Nodo("CuerpoSino");

              nuevo.Hijo(sent);

              RESULT = nuevo; :}

            | {:RESULT = new
Nodo("CuerpoSino");:}

            ;


RETORNA ::= retornar EXP:exp fin

      {:Nodo nuevo = new
Nodo("Retorno");

        Nodo e = new Nodo("Exp");

        e.Hijo(exp);

        nuevo.Hijo(e);

        RESULT = nuevo;

      :}

      ;


DECLARACION_LOCAL ::= var
TIPO_DATO:t id:id VISIBILIDAD:v
OPCION:op

            {: Nodo nuevo = new
Nodo("DeclaraLocalVariable");

              nuevo.Hijo(t);

              if(op.valor.equals("ListaID")){

                Nodo l= new Nodo("L");

                Nodo i= new Nodo(id);

                l.Hijo(i);

                l.Hijo(v);

                op.Hijo(l);

                nuevo.Hijo(op);

              }else {

                Nodo m = new
Nodo("ListaID");

                Nodo mm = new Nodo("L");

                Nodo i = new Nodo(id);

                m.Hijo(mm);

                mm.Hijo(i);

                mm.Hijo(v);

                nuevo.Hijo(m);

                nuevo.Hijo(op);


              }


              RESULT = nuevo;

              :}

            | var TIPO_DATO:t id:id
ARREGLO:a VISIBILIDAD:v
ASIGNACION_ARREGLO:nodo

              {: Nodo nuevo = new
Nodo("DeclaraLocalArreglo");

              Nodo l= new Nodo("ListaID");

              Nodo i= new Nodo(id);

              l.Hijo(i);

              l.Hijo(v);

              nuevo.Hijo(t);

              nuevo.Hijo(l);

              nuevo.Hijo(a);

              nuevo.Hijo(nodo);

              RESULT = nuevo;

              :}

            | var TIPO_DATO:t id:id
VISIBILIDAD:v igual nuevo
LLAMAR_INSTANCIA:nodo
```

```
            {: Nodo nuevo = new
Nodo("InstanciaLocal");

            Nodo l= new Nodo("ListaID");

            Nodo as= new
Nodo("Objeto");

            Nodo i= new Nodo(id);

            l.Hijo(i);

            l.Hijo(v);

            as.Hijo(nodo);

            nuevo.Hijo(t);

            nuevo.Hijo(l);

            nuevo.Hijo(as);

            RESULT = nuevo;

            :}

        ;


LLAMAR_INSTANCIA ::= id:id parenAbre
parenCierra

              {:RESULT = new Nodo(id);:}

             ;


ASIGNA ::= id:id igual EXP:exp

        {:

            Nodo nuevo = new
Nodo("Asignacion");

            Nodo i= new Nodo(id);

            Nodo e= new Nodo("Exp");

            nuevo.Hijo(i);

            e.Hijo(exp);

            nuevo.Hijo(e);

            RESULT = nuevo;

            :}

      | id:id igual nuevo
LLAMAR_INSTANCIA:ins

            {:

                Nodo nuevo = new
Nodo("Asignacion");

                Nodo i= new Nodo(id);

                Nodo e= new
Nodo("InstanciaLocal");

                nuevo.Hijo(i);

                e.Hijo(ins);

                nuevo.Hijo(e);

                RESULT = nuevo;

            :}

      | id:id L_ACCESO:nodo igual
EXP:exp

      {:

            Nodo nuevo = new
Nodo("AsignacionAcceso");

            Nodo i= new Nodo(id);

            Nodo e= new Nodo("Exp");

            nuevo.Hijo(i);

            nuevo.Hijo(nodo);

            e.Hijo(exp);

            nuevo.Hijo(e);

            RESULT = nuevo;

        :}

      ;


L_ACCESO ::= L_ACCESO:nodo A:a

      {:a.setNombre("L_Acceso");
nodo.Hijo(a); RESULT = nodo;:}
```

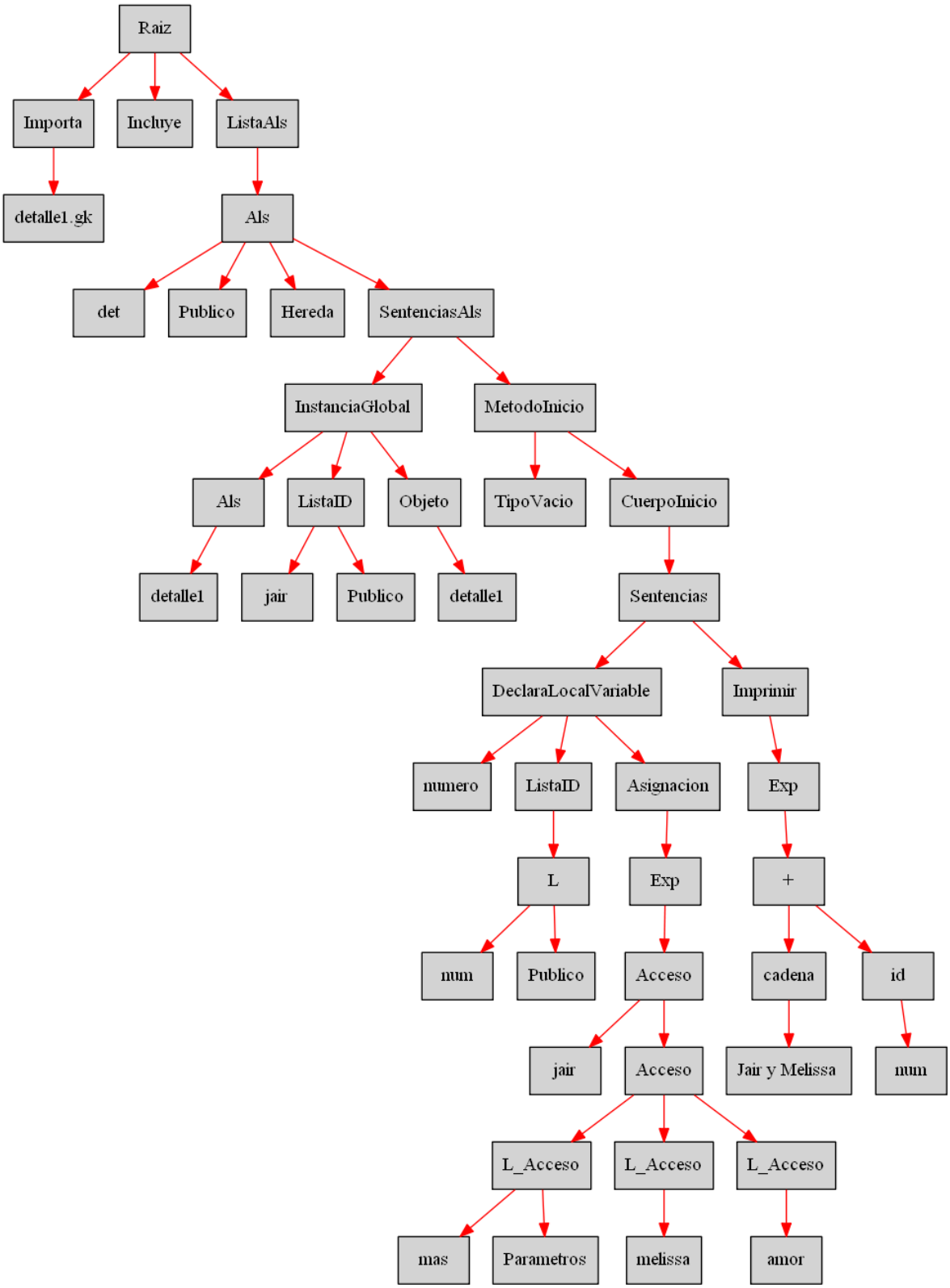
```
        | A:a

    {:a.setNombre("L_Acceso");

    Nodo nuevo = new
Nodo("Acceso");

    nuevo.Hijo(a);

      RESULT = nuevo;

    :}

        ;


A ::= punto id:id parenAbre
PARAM_LLAMA:nodo parenCierra

    {:Nodo nuevo  = new Nodo("");
Nodo i = new Nodo(id); nuevo.Hijo(i);
nuevo.Hijo(nodo); RESULT = nuevo;:}

  | punto id:id {:Nodo nuevo = new
Nodo(""); Nodo i = new Nodo(id);
nuevo.Hijo(i); RESULT = nuevo;:}

  | punto LLAMADA_ARREGLO:nodo
{:Nodo nuevo = new Nodo("");
nuevo.Hijo(nodo); RESULT = nuevo;:}

  ;


LLAMAR ::= llamar id:id parenAbre
PARAM_LLAMA:nodo parenCierra

        {:

        Nodo nuevo = new
Nodo("LlamaFun");

        Nodo i = new Nodo(id);

        nuevo.Hijo(i);

        nuevo.Hijo(nodo);

        RESULT = nuevo;

        :}

    | llamar id:id L_ACCESO:nodo

    {:

        Nodo nuevo = new
Nodo("Acceso");

        Nodo i = new Nodo(id);

        nuevo.Hijo(i);

        nuevo.Hijo(nodo);

        RESULT = nuevo;

    :}

    ;


LLAMARHK ::= llamarHK id:id parenAbre
PARAM_LLAMA:nodo parenCierra

        {:


        Nodo nuevo = new
Nodo("LlamarHK");

        Nodo i = new Nodo(id);

        nuevo.Hijo(i);

        nuevo.Hijo(nodo);

        RESULT = nuevo;

        :}

    ;


PARAM_LLAMA ::= PARAM:nodo
{:RESULT = nodo;:}

        | {:RESULT = new
Nodo("Parametros");:}

        ;


PARAM ::= PARAM:param coma P:p

        {: param.Hijo(p); RESULT =
param;:}
```

```
      | P:p
          {:Nodo nuevo = new
Nodo("Parametros"); nuevo.Hijo(p);
RESULT = nuevo;:}
      ;
P ::= EXP:exp
   {:Nodo nuevo = new Nodo("Exp");
nuevo.Hijo(exp); RESULT = nuevo;:}
   ;

EXP ::= EXP:j mas EXP:m
      {:
         Nodo nuevo = new Nodo("+");
         nuevo.Hijo(j);
         nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP:j menos EXP:m
      {:
         Nodo nuevo = new Nodo("-");
         nuevo.Hijo(j);
         nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP:j por EXP:m
      {:
         Nodo nuevo = new Nodo("*");
         nuevo.Hijo(j);
         nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP:j div EXP:m
      {:
         Nodo nuevo = new Nodo("/");
         nuevo.Hijo(j);
         nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP:j potencia EXP:m
      {:
         Nodo nuevo = new Nodo("^");
         nuevo.Hijo(j);
         nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP:j or EXP:m
      {:
         Nodo nuevo = new Nodo("||");
         nuevo.Hijo(j);
         nuevo.Hijo(m);
         RESULT = nuevo;
      :}
    | EXP:j xor EXP:m
      {:
         Nodo nuevo = new Nodo("&|");
         nuevo.Hijo(j);
         nuevo.Hijo(m);
         RESULT = nuevo;
```

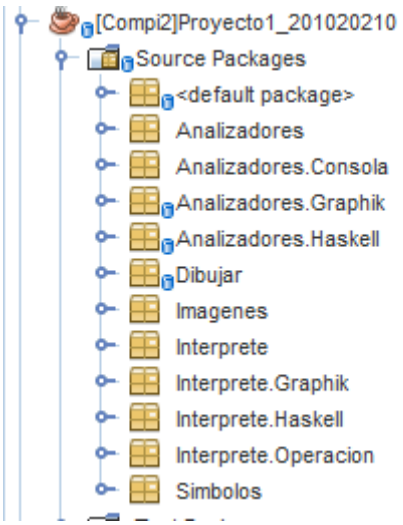
```
  :}
| EXP:j and EXP:m
  {:
     Nodo nuevo = new Nodo("&&");
     nuevo.Hijo(j);
     nuevo.Hijo(m);
     RESULT = nuevo;
  :}
| not EXP:m
  {:
  Nodo nuevo = new Nodo("!");
  nuevo.Hijo(m);
  RESULT = nuevo;
  :}
| EXP:j menorq EXP:m
  {:
     Nodo nuevo = new Nodo("<");
     nuevo.Hijo(j);
     nuevo.Hijo(m);
     RESULT = nuevo;
  :}
| EXP:j mayorq EXP:m
  {:
     Nodo nuevo = new Nodo(">");
     nuevo.Hijo(j);
     nuevo.Hijo(m);
     RESULT = nuevo;
  :}
| EXP:j menorigualq EXP:m
  {:
     Nodo nuevo = new Nodo("<=");
     nuevo.Hijo(j);
     nuevo.Hijo(m);
     RESULT = nuevo;
  :}
| EXP:j mayorigualq EXP:m
  {:
     Nodo nuevo = new Nodo(">=");
     nuevo.Hijo(j);
     nuevo.Hijo(m);
     RESULT = nuevo;
  :}
| EXP:j igualigual EXP:m
  {:
     Nodo nuevo = new Nodo("==");
     nuevo.Hijo(j);
     nuevo.Hijo(m);
     RESULT = nuevo;
  :}
| EXP:j diferente EXP:m
  {:
     Nodo nuevo = new Nodo("!=");
     nuevo.Hijo(j);
     nuevo.Hijo(m);
     RESULT = nuevo;
  :}

| EXP:nodo incremento
```

```
        {:
        Nodo nuevo = new
Nodo("Incremento");
        nuevo.Hijo(nodo);
        RESULT = nuevo;
        :}
    | EXP:nodo decremento
        {:
        Nodo nuevo = new
Nodo("Decremento");
        nuevo.Hijo(nodo);
        RESULT = nuevo;
        :}
    | menos EXP:nodo
        {:
        Nodo nuevo = new
Nodo("Unario");
        nuevo.Hijo(nodo);
        RESULT = nuevo;
        :}
    | COLUMNA:nodo
{:RESULT = nodo;:}
    | parenAbre EXP:nodo parenCierra
{:RESULT = nodo;:}
    | LLAMAR:nodo                {:RESULT
= nodo;:}
    | LLAMADA_ARREGLO:nodo
{:RESULT = nodo;:}
    | LLAMARHK:nodo
{:RESULT = nodo;:}
    | llaveAbre L_POSICIONES:nodo
llaveCierra {:RESULT = nodo;:}
    | id:id L_ACCESO:nodo
        {:
        Nodo nuevo = new
Nodo("Acceso");
        Nodo n=new Nodo(id);
        nuevo.Hijo(n);
        nuevo.Hijo(nodo);
        RESULT = nuevo;
        :}

    | num:num
        {: Nodo nuevo = new
Nodo("numero");
            Nodo n = new Nodo(num);
            nuevo.Hijo(n);
            RESULT = nuevo; :}
    | cad:cad
        {: Nodo nuevo = new
Nodo("cadena");
            Nodo n = new
Nodo(cad.toString().replace("\"", ""));
            nuevo.Hijo(n); RESULT = nuevo; :}
    | dec:num
        {: Nodo nuevo = new
Nodo("decimal");
            Nodo n = new Nodo(num);
            nuevo.Hijo(n);
            RESULT = nuevo; :}
    | carac:letra
        {: Nodo nuevo = new
Nodo("caracter");
            Nodo l = new Nodo(letra);
            nuevo.Hijo(l);
```

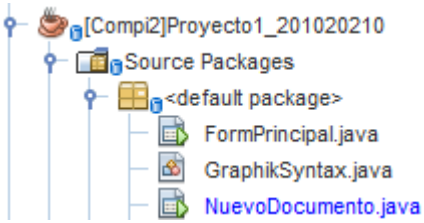
```
        RESULT = nuevo; :}
    | id:id
      {:  Nodo nuevo = new Nodo("id");
        Nodo i = new Nodo(id);
        nuevo.Hijo(i);
        RESULT = nuevo; :}
    | falso:id
      {:  Nodo nuevo = new
Nodo("falso");
        Nodo n = new Nodo("falso");
        nuevo.Hijo(n);
        RESULT = nuevo; :}
    | verdadero:id
      {:  Nodo nuevo = new
Nodo("verdadero");
        Nodo n = new
Nodo("verdadero");
        nuevo.Hijo(n);
        RESULT = nuevo; :}
    ;

LLAMADA_ARREGLO ::= id:id DIM:dim
            {:
              Nodo nuevo = new
Nodo("LlamaArreglo");
              Nodo n=new Nodo(id);
              nuevo.Hijo(n);
              nuevo.Hijo(dim);
              RESULT = nuevo;
            :}
                ;
DIM ::= DIM:dim D:d
      {:dim.Hijo(d);RESULT = dim;:}
      | D:dim
      {:
        Nodo d = new
Nodo("Dimensiones");
        d.Hijo(dim);
        RESULT = d;:} ;
D ::= corAbre EXP:exp corCierra
      {:Nodo nuevo = new Nodo("Exp");
nuevo.Hijo(exp); RESULT = nuevo;:};
```


Raiz
Importa
Incluye
ListaAls
detalle1.gk
Als
det
Publico
Hereda
SentenciasAls
InstanciaGlobal
MetodoInicio
Als
ListaID
Objeto
TipoVacio
CuerpoInicio
detalle1
jair
Publico
detalle1
Sentencias
DeclaraLocalVariable
Imprimir
numero
ListaID
Asignacion
Exp
L
Exp
+
num
Publico
Acceso
cadena
id
jair
Acceso
Jair y Melissa
num
L_Acceso
L_Acceso
L_Acceso
mas
Parametros
melissa
amor

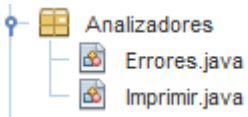
PAQUETES Y CLASES



DESCRIPCION DE CADA UNO:

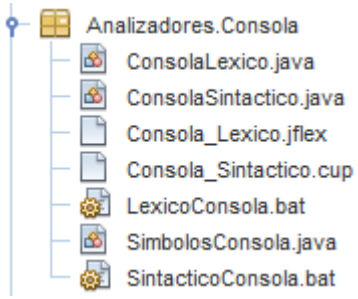


- Form Principal
  Contiene la interfaz grafica principal del proyecto
- GraphikSyntax
  Contiene las palabras de Graphik para pintar el texto
- NuevoDocumento
  Este Form se despliega cuando se quiere crear un nuevo documento



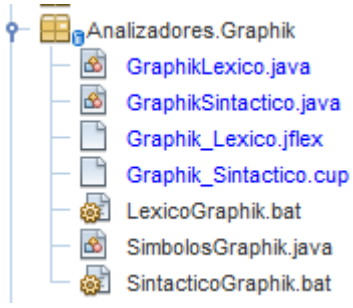
Analizadores:

- Errores:
  Clase para concatenar los errores.
- Imprimir:
  Clase para imprimir el resultado de la Sentencia Imprimir



Analizadores.Consola: todo el Analisis de las sentecias de Consola

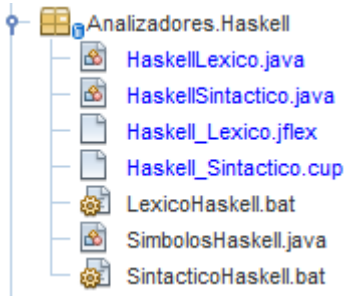
- ConsolaLexico
  Clase que genera el analizador lexico
- ConsolaSintactico
  Clase que genera el analizador sintactico
- Consola_Lexico
  Archivo que contiene el análisis lexico
- Consola_Sintactico
  Archivo que contiene la gramatica del análisis sintactico
- LexicoConsola
  Archivo para generar clases de analizadores lexicos
- SimbolosConsola
  Clase de símbolos que genera el analizador lexico
- SintacticoConsola
  Archivo para generar clases de analizadores sintácticos



Analizadores.Graphik: todo el Analisis de las sentecias de Graphik

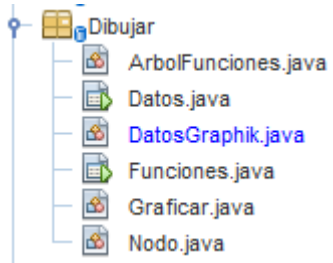
- GraphikLexico
  Clase que genera el analizador lexico
- GraphikSintactico
  Clase que genera el analizador sintactico
- Graphik_Lexico
  Archivo que contiene el análisis lexico
- Graphik_Sintactico
  Archivo que contiene la gramatica del análisis sintactico
- LexicoGraphik

Archivo para generar clases de analizadores lexicos
- SimbolosGraphik
  Clase de símbolos que genera el analizador lexico
- SintacticoGraphik
  Archivo para generar clases de analizadores sintácticos



Analizadores.Haskell: todo el Analisis de las sentecias de Haskell

- HaskellLexico
  Clase que genera el analizador lexico
- HaskellSintactico
  Clase que genera el analizador sintactico
- Haskell_Lexico
  Archivo que contiene el análisis lexico
- Haskell_Sintactico
  Archivo que contiene la gramatica del análisis sintactico
- HaskellGraphik
  Archivo para generar clases de analizadores lexicos
- SimbolosHaskell
  Clase de símbolos que genera el analizador lexico
- SintacticoHaskell
  Archivo para generar clases de analizadores sintácticos

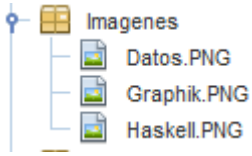


Dibujar:

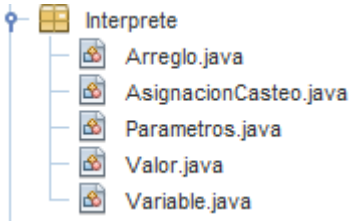
- ArbolFunciones
  Clase para generar la grafica de funciones
- Datos
  Frame que despliega el resultado de Datos
- DatosGraphik

Clase que contiene lo necesario para ejecutar Datos

- Funciones
  Frame que despliega la grafica de funciones
- Graficar
  Clase que contiene lo necesario para ejecutar Graphikar
- Nodo
  Clase nodo para generar los arboles

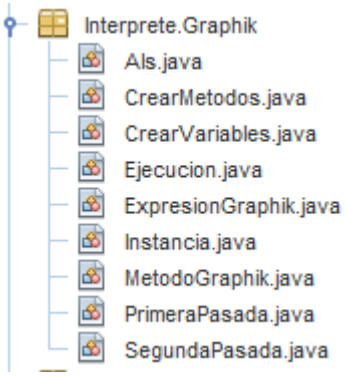


Imágenes: contiene las imágenes de los iconos del Frame NuevoDocumento



Interprete:

- Arreglo
  Clase que tiene métodos para hacer el mapeo lexicográfico de los arreglos
- AsignacionCasteo
  Clase que contiene el método para hacer el casteo implícito
- Parametros:
  Clase que contiene el constructor para los parámetros
- Valor:
  Una Clase que contiene el constructor de un valor
- Variable
  Clase que contiene el constructor para las variables



Interprete.Graphik:

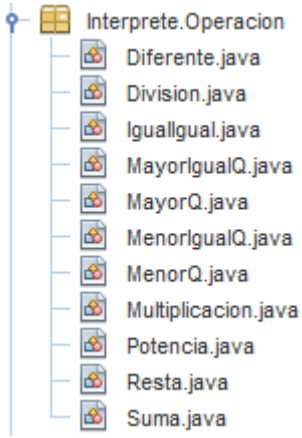
- Als
  Clase que contiene todo lo que guarda el Als cuando lo reconoce, contiene las variables, métodos, etc.
- CrearMetodos
  Clase que contiene los métodos para crear y agregar métodos
- Crear Variables
  Clase que contiene los métodos para crear variables globales y locales
- Ejecucion
  Clase que hace las dos pasadas para en análisis y ejecución
- ExpresionGraphik
  Clase que contiene todos los case de cuando viene una expresión
- Instancia
  Clase que contiene los métodos para el acceso de un objeto
- MetodoGraphik
  Clase que contiene el costructor de un método
- PrimeraPasada:
  Clase que hace el análisis de la entrada creando sus variables globales y guardando sus métodos
- SegundaPasada:
  Clase que realiza la ejecución de la entrada

Interprete.Haskell:

- Concatena
  Clase que contiene los métodos para manejar una lista en haskell
- ExpresionHaskell
  Clase que contiene todos los case de cuando viene una expresión
- FuncionHaskell

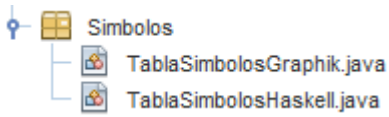
Clase que contiene el constructor para una función

- FuncionesPropiasHaskell
  Clase que contiene todas las funciones propias de haskell
- RecorreHaskell
  Clase que hace la ejecución de haskell



Interprete.Operacion

- Diferente
  Clase que hace las operaciones necesarias para la condición (!=)
- Division
  Clase que hace las operaciones necesarias para la condición (/)
- IgualIgual
  Clase que hace las operaciones necesarias para la condición (==)
- MayorIgualQ
  Clase que hace las operaciones necesarias para la condición (>=)
- MayorQ
  Clase que hace las operaciones necesarias para la condición (>)
- MenorIgualQ
  Clase que hace las operaciones necesarias para la condición (<=)
- MenorQ
  Clase que hace las operaciones necesarias para la condición (<)
- Multiplicación
  Clase que hace las operaciones necesarias para la condición (*)
- Potencia
  Clase que hace las operaciones necesarias para la condición (^)
- Resta
  Clase que hace las operaciones necesarias para la condición (-)
- Suma
  Clase que hace las operaciones necesarias para la condición (+)

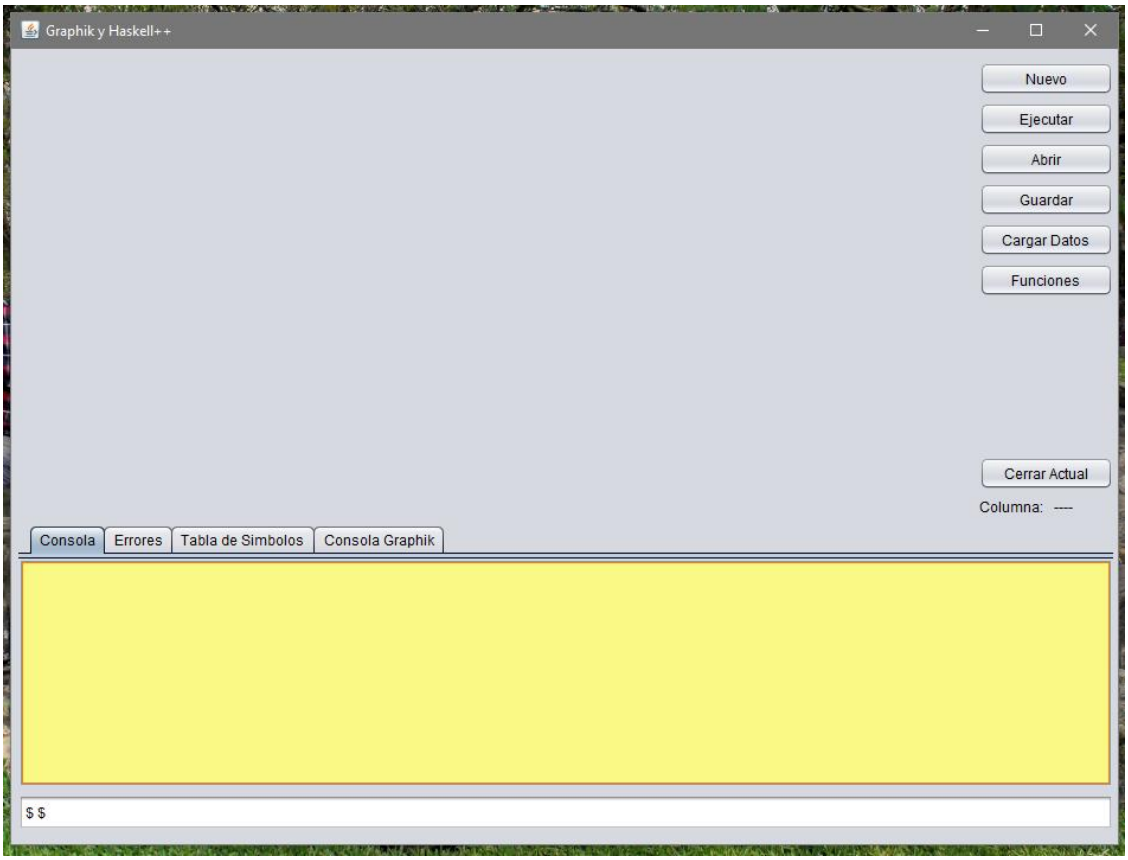


Simbolos:

- TablaSimbolosGraphik
  Contiene la tabla de símbolos de Graphik
- TablaSimbolosHaskell
  Contiene la tabla de símbolos de Haskell

Interfaz Gráfica

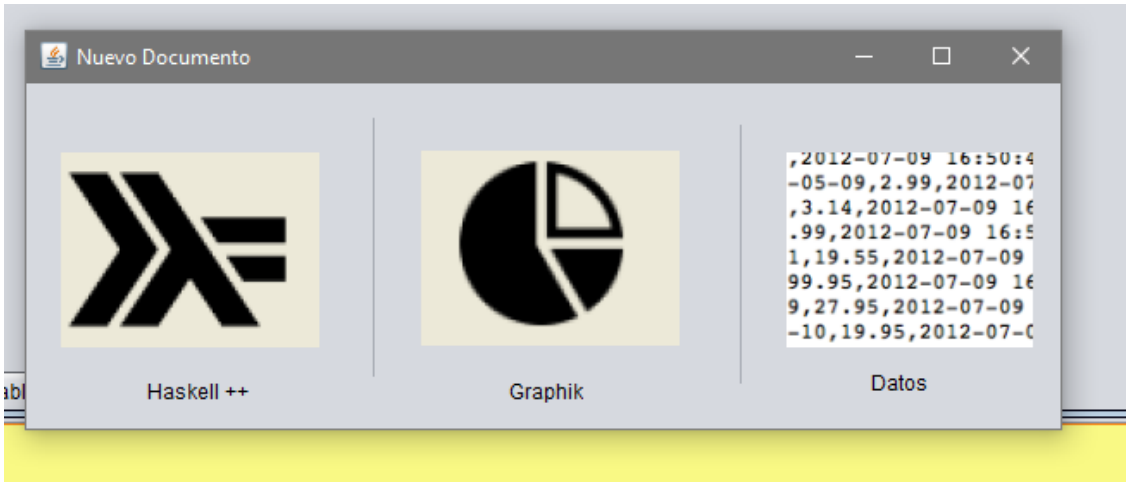
- FormPrincipal



- Nuevo: para crear un nuevo documento
- Ejecutar: para realizar el análisis del archivo
- Abrir: abrir un archivo nuevo
- Guardar: guardar el archivo de la pestaña actual
- Cargar datos: carga un archivo .csv
- Funciones: muestra el árbol de funciones
- Cerrar Actual: cierra la pestaña actual

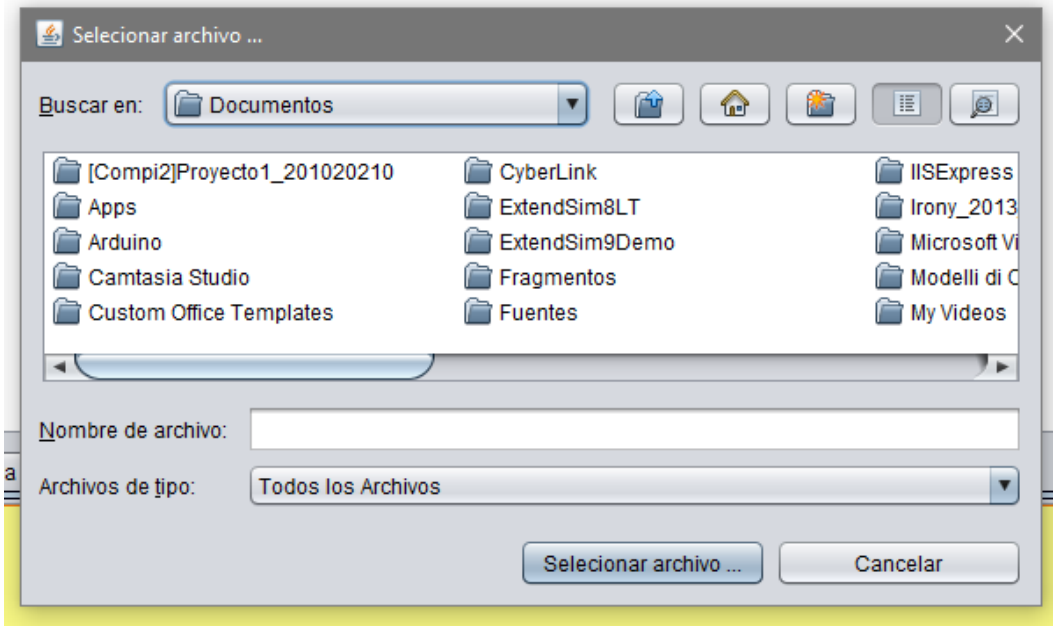
- Consola: muestra los resultados de Haskell
- Errores: muestra los errores del análisis
- Tabla de símbolos: muestra las variables y métodos de cada Als
- Consola Graphik: muestra los resultados de la sentencia Imprimir en Graphik

- NuevoDocumento

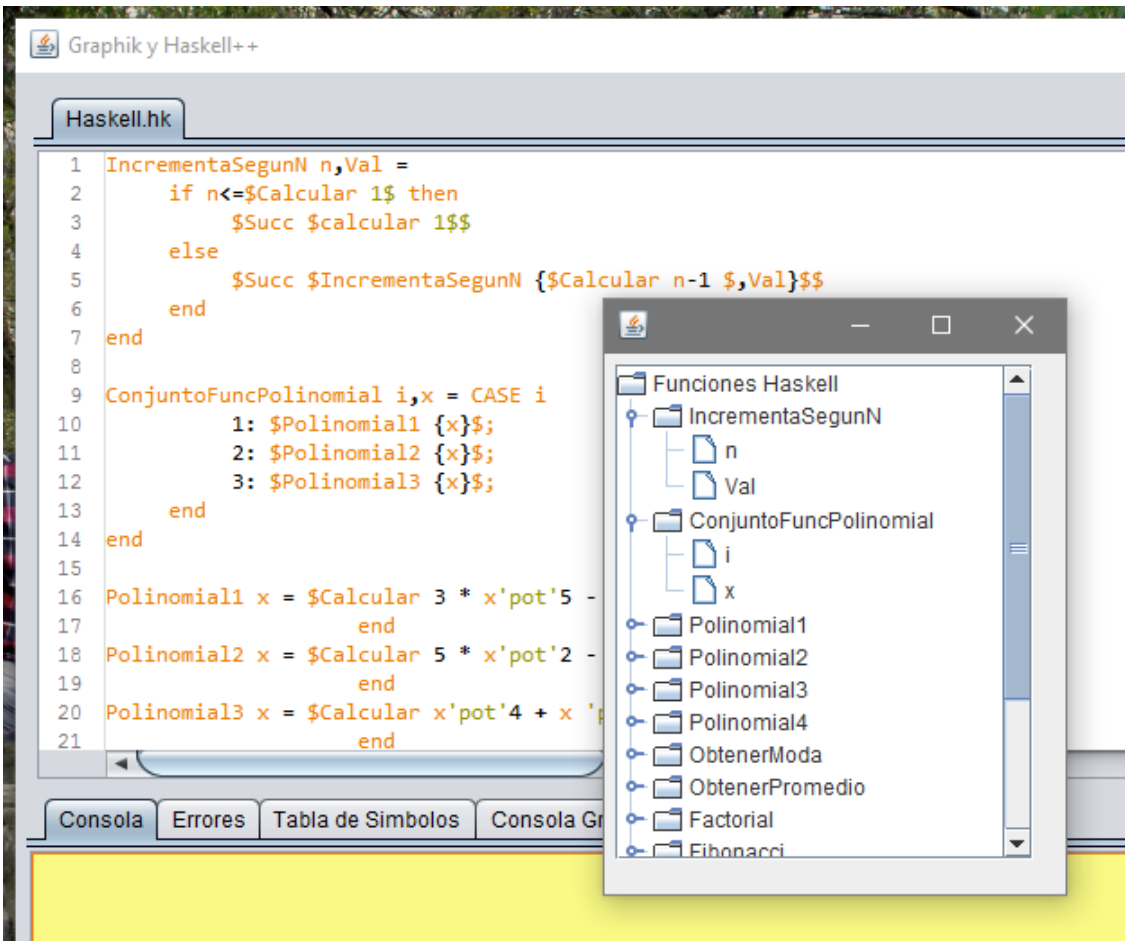


- Haskell++: crea un nuevo documento llamado Haskell.hk
- Graphik: crea un nuevo documento llamado Graphik.gk
- Datos: crea un nuevo documento llamado Datos.csv

- JFileChooser:

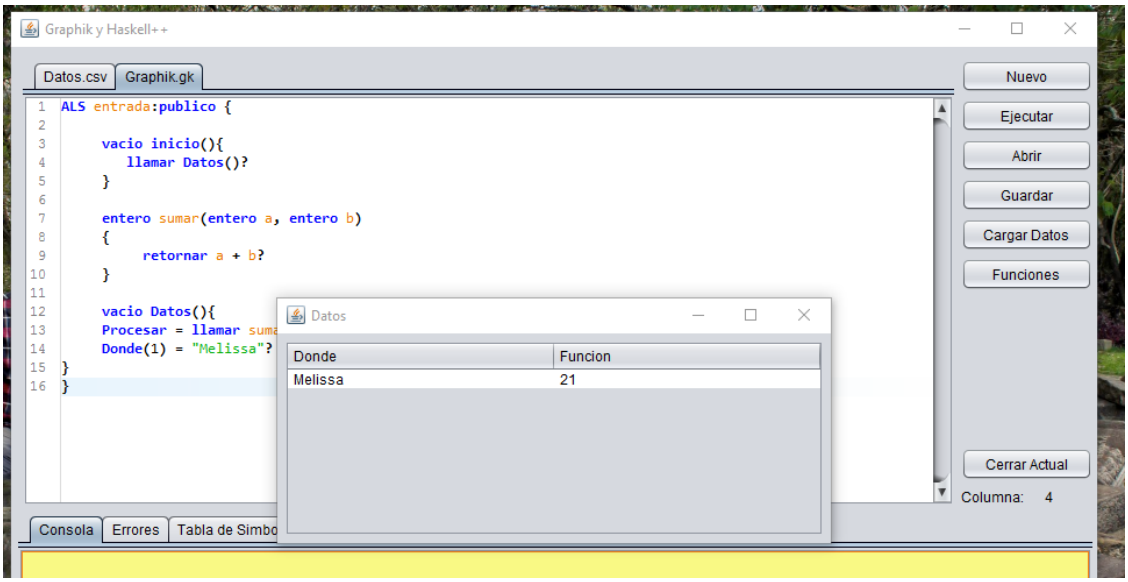


- Funciones



  - Muestra cada Función de Haskell cargada y dentro de cada una va el nombre de los parámetros que recibe

- Datos



- Grafica cuando se ejecutan las sentencias de graphikar_funcion