Diseño de Compiladores
21-Noviembre-2008

Código Fuente clasificado por paquete

Cristhian Parra
Fernando Mancía

Índice de contenido

1. Paquete traductor

```
package traductor;

import java.util.ArrayList;
import java.util.Iterator;

/**
 * Clase que implementa el contenedor del alfabeto sobre el cual se define la
 * expresión regular a traducir<br><br>
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class Alfabeto extends ArrayList<String> {

    public Alfabeto(String simbolos) {

        for (int i = 0; i < simbolos.length(); i++) {
            String tmp = "" + simbolos.charAt(i);

            /**
             * @TODO
             * 1. No incluir espacio en blanco en el alfabeto
             */
            if (! this.contains(tmp)) {
                this.add(tmp);
            }
        }

        this.ordenar();
    }

    /**
     * Método para obtener un iterador sobre el alfabeto.
     * @return Iterador sobre el alfabeto.
     */
    public Iterator getIterator() {
        return this.iterator();
    }

    /**
     * Método que permite obtener el tamaño del alfabeto .
     * @return Cantidad de símbolos del alfabeto.
```

```
     */
    public int getTamanho() {
        return this.size();
    }

    /**
     * Método para verificar la pertenencia de un símbolo al alfabeto.
     * @param simbolo Símbolo cuya pertenencia queremos verificar
     * @return <ul>
     *            <li><b>True</b> si el simbolo pertenece al alfabeto</li>
     *            <li><b>False</b> si el simbolo no pertenece al alfabeto</li>
     *         </ul>
     */
    public boolean contiene(String simbolo) {
        if ( this.contains(simbolo) ) return true;
        return false;
    }

    /**
     * Método que imprime el alfabeto.
     * @return Un String que contiene la representación en texto del alfabeto.
     */
    public String imprimir() {

        String result = "ALPHA = { ";
        for (int i = 0; i < this.size(); i++) {

            result += this.get(i);

            if (!(i == (this.size()-1))) {
                result += ", ";
            }
        }

        return result + " } ";

    }

    /**
     * Método privado que ordena las letras del alfabeto en orden ascendente.
     */
    private void ordenar() {
        String a[] = new String[1];
        a = this.toArray(a);
        java.util.Arrays.sort(a);

        this.removeAll(this);
        for(int i = 0; i < a.length; i++) {
            this.add(a[i]);
        }
    }
}
package traductor;

import afgenjava.*;
import exceptions.LexicalError;
import exceptions.SyntaxError;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * El traductor es el encargado de implementar los procedimientos necesarios
 * para llevar a cabo el proceso de traducción <br> <br>
 *
 * El traductor está basado en el siguiente BNF para definir un lenguaje de
 * expresiones regulares. <br><br>
 * <ol TYPE=i>
 *   <li>  RE => resimple A         </li>
 *   <li>  A  => "|" resimple A | Є </li>
 *   <li>  resimple => rebasico B   </li>
 *   <li>  B => rebasico B | Є      </li>
 *   <li>  rebasico => list op      </li>
 *   <li>  op => * | + | ? | Є      </li>
 *   <li>  list => grupo | leng     </li>
 *   <li>  grupo => "(" RE ")"      </li>
 *   <li>  leng => [alfabeto del lenguaje] </li>
 * </ol> <br><br>
 *
 * Se implementa un Traductor Dirigido por la Sintaxis que sigue este BNF y
 * produce el automata basándose en las construcciones de Thompson. <br><br>
```

```
 *
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancia ({@link fernandomancia@gmail.com})
 */
public class Analizador {

    /**
     * Analizador Lexico
     */
    private Lex lexico;

    /**
     * Expresión regular a traducir
     */
    private String regex;

    /**
     * Token que contiene el simbolo que se está procesando actualmente
     */
    private Token preanalisis;

    /**
     * Alfabeto sobre el cual está definida la expresión regular.
     */
    private Alfabeto alfabeto;

    /**
     * Automata en el cual se guardará el resultado final de la traducción.
     * Se trata de un Automata del tipo AFN.
     */
    private Automata automata;

    /**
     * Simbolo especial utilizado para guardar recordar el símbolo operador
     * consumido por una producción, cuando se deba aplicar la misma en una
     * producción superior
     */
    private String Special;

    /**
     * Contador de caracteres procesados
     */
    private int posicion;

    /**
     * Flag que indica la existencia o no de errores al final de la traducción
     */
    private boolean hayErrores = false;

    /**
     * Flag que indica la existencia o no de errores al final de la traducción
     */
    private String errMsg = "";
    /**
     * Constructor vacío de la clase <code>Analizador</code>
     */
    public Analizador() {
    }

    /**
     * Constructor del <code>Analizador</code> Sintáctico a partir de la
     * expresión regular y el alfabeto de entrada.
     *
     * @param regex Expresión regular cuyo AFN queremos generar
     * @param alfabeto Alfabeto sobre el cual está definida la expresión regular
     */
    public Analizador(String regex, String alfabeto) {
        this.setPosicion(0);
        this.regex = regex;
        this.alfabeto = new Alfabeto(alfabeto);
        this.lexico = new Lex(regex, alfabeto); // creamos el analizador léxico
        try {
            // creamos el analizador léxico
            this.preanalisis = nextSymbol(); // obtenemos el primer símbolo desde el analizador
léxico
        } catch (LexicalError ex) {

            this.hayErrores = true;
            this.errMsg =
```

```
                    "Se produjo un error FATAL en el traductor. La generación del AFN no puede
continuar\n"+
                    "--> "+ex.getMessage();

            System.out.println(this.getErrMsg());
            this.abort();
        }
        automata = new Automata();
        automata.setTipo(TipoAutomata.AFN);
    }

    /**
     * Implementación del procedimiento que se encarga de parear el símbolo de
     * preanálisis actual con la entrada esperada según la sintaxis del lenguaje
     *
     * @param tok Símbolo esperado
     * @throws exceptions.SyntaxError Error de Sintaxis
     */
    private void Match(String simbolo) throws SyntaxError, LexicalError {

        Token tok = new Token(simbolo); // se crea un Token temporal para
                                        // compararlo con preanalisis

        if ( getPreanalisis().compareTo(tok) == 0 ) {
            this.setPreanalisis(this.nextSymbol());
            this.Special = tok.getValor();
            this.incPosicion();
        } else {
            throw new SyntaxError(tok.getValor(),this.getPosicion());
        }
    }

    /**
     * Método que termina de manera instantánea el proceso de análisis y
     * traducción cuando se produce un error. <br><br>
     *
     * Inicialmente, el método solo consiste en llamar a la primitiva
     * <code>System.exit(0)</code>, pero permite encapsular el comportamiento
     * de esta acción para modificarla en el futuro de una sola vez.
     */
    private void abort() {
        // Do nothing
    }

    /**
     * Llamada al analizador léxico para obtener el siguiente caracter de la
     * cadena de entrada <br><br>
     *
     * Si el analizador léxico encuentra un error (como que el caracter no
     * pertenece al alfabeto) se atrapa la excepción, se informa en la salida y
     * se aborta el análisis. <br><br>
     * @return Token que contiene el símbolo siguiente a procesar
     */
    private Token nextSymbol() throws LexicalError {
        Token result = null;
        result = this.lexico.next();
        return result;
    }

    public Automata traducir() {
        this.automata = this.RE();

        if (!this.isHayErrores()) {
            if (preanalisis.getTipo() != TipoToken.FIN) {
                this.hayErrores = true;
                this.errMsg = "Quedaron caracteres sin analizar debido al siguiente Token no
esperado["+
                        this.getPosicion()+"]: "+preanalisis.getValor();
            }
        }

        return this.automata;
    }

    /**
     * Método correspondiente al símbolo inicial de la gramática de expresiones
     * regulares. <br><br>
     *
     * Las producciones que pueden ser vacío, retornan un valor null en ese caso.
     * Las demás producciones lanzan excepciones que se trasladan a los ámbitos
```

```
     * de llamada superiores
     *
     * @TODO
     * - Implementar Exception Management: Acciones a tomar a partir de los
     *   distintos tipos de errores
     *
     * @return Autoamata producido por la producción   RE => resimple A.
     *
     */
    private Automata RE() {

        // automatas auxiliares de producciones llamadas
        Automata Aux1 = null;
        Automata Aux2;

        try {

            Aux1 = this.resimple();
            Aux2 = this.A();

            if (Aux2 != null) {
                Aux1.thompson_or(Aux2);
            }
        } catch (SyntaxError ex) {

            this.hayErrores = true;
            this.errMsg =
                    "Se produjo un error FATAL en el traductor. La generación del AFN no puede
continuar\n"+
                    "--> "+ex.getMessage();

            System.out.println(this.getErrMsg());
            this.abort();
        } catch (LexicalError ex) {

            this.hayErrores = true;
            this.errMsg =
                    "Se produjo un error FATAL en el analizador léxico. La generación del AFN no
puede continuar\n"+
                    "--> "+ex.getMessage();
            System.out.println(this.getErrMsg());

            this.abort();
        } catch (Exception ex) {
            this.hayErrores = true;
            this.errMsg =
                    "Se produjo un error FATAL de diseño. La generación del AFN no puede
continuar\n"+
                    "--> "+ex.getMessage();
            System.out.println(this.getErrMsg());
            this.abort();
        }

        if (!(this.hayErrores) ){
            this.setAutomata(Aux1); // Actualizar el Automata Global
            Aux1.setAlpha(this.alfabeto);
            Aux1.setRegex(this.regex);
        }
        return Aux1;
    }

    /**
     * Producción A, que permite la recursión necesaria para producir cadenas
     * de expresiones regulares separadas por el operador "|" (disyunción) <br><br>
     *
     * @return null si derivó en vacío, en caso contrario, el automata generado
     * @throws exceptions.SyntaxError
     */
    private Automata A() throws SyntaxError, LexicalError {
        try {
            Token or = new Token("|");

            if (preanalisis.compareTo(or) == 0) {
                this.Match("|"); // si preanalisis es el esperado, consumimos,
                return RE();
            } else {
                return null;    // si es vacío se analiza en otra producción
            }
        } catch (SyntaxError ex) {
            this.hayErrores = true;
```

```
        throw new SyntaxError("se esperaba '|' en lugar de -> "
                        +this.preanalisis.getValor(),this.getPosicion());
    }
}

/**
 * Producción resimple
 *
 * @return Automata producido por la producción
 * @throws exceptions.SyntaxError
 * @throws exceptions.LexicalError
 */
private Automata resimple() throws SyntaxError, LexicalError {
    Automata Aux1 = this.rebasico();
    Automata Aux2 = this.B();

    if (Aux2 != null) {
        Aux1.thompson_concat(Aux2);
    }

    return Aux1;
}

/**
 * Producción rebasico.
 * @return Automata generado luego de derivar la producción
 */
private Automata rebasico() throws SyntaxError, LexicalError {

    Automata Aux1 = list();

    if (Aux1 != null) {
        char operator = op();

        switch (operator) {
            case '*':
                Aux1.thompson_kleene();
                break;
            case '+':
                Aux1.thompson_plus();
                break;
            case '?':
                Aux1.thompson_cerouno();
                break;
            case 'E':
                break;
        }
    } /*else if (preanalisis.) {
        throw new SyntaxError("se esperaba un símbolo del lenguaje y se encontró: "
                        +this.preanalisis.getValor(),this.getPosicion());
    }*/

    return Aux1;
}

/**
 * La producción B debe verificar si preanalisis está en el conjunto primero
 * de resimple, y si está, volver a ejecutar resimple. En caso contrario debe
 * retornar null. <br> <br>
 *
 * El conjunto Primero de resimple es {"(",[alpha]}.
 *
 * @return Automata el automata producido por la producción, o null si la
 *                  producción deriva en vacío.
 * @throws exceptions.SyntaxError
 * @throws exceptions.LexicalError
 */
private Automata B() throws SyntaxError, LexicalError {

    String current = preanalisis.getValor();
    Automata result = null;

    if ( (preanalisis.getTipo() != TipoToken.FIN) &&
         (this.alfabeto.contiene(current) || current.compareTo("(")==0)
       ) {
        result = this.resimple();
    }

    return result;
}
```

```
    private Automata list() throws SyntaxError, LexicalError {

        Token grupofirst = new Token("(");

        if(preanalisis.compareTo(grupofirst) == 0) {
            return this.grupo();
        } else {
            return this.leng();
        }
    }

    private char op() throws SyntaxError, LexicalError {
        char operador = 'E';

        if (preanalisis.getValor().compareTo("") != 0) {
            operador = preanalisis.getValor().charAt(0);

            switch (operador) {
                case '*':
                    this.Match("*");
                    break;
                case '+':
                    this.Match("+");
                    break;
                case '?':
                    this.Match("?");
                    break;
                default:
                    return 'E';
            }
        }
        return operador;
    }

    private Automata grupo() throws SyntaxError, LexicalError {
        try {
            this.Match("(");
        } catch (SyntaxError ex) {
            this.hayErrores = true;
            throw new SyntaxError("se esperaba el símbolo -> '('",this.getPosicion());
        }

        Automata Aux1 = this.RE();

        try {
            this.Match(")");
        } catch (SyntaxError ex) {
            this.hayErrores = true;
            throw new SyntaxError("se esperaba el símbolo -> ')'",this.getPosicion());
        }

        return Aux1;
    }

    /**
     *
     * @return
     */
    private Automata leng() throws LexicalError {
        Automata nuevo = null;
        try {
            if (preanalisis.getTipo() != TipoToken.FIN) {
                nuevo = new Automata(preanalisis.getValor(),TipoAutomata.AFN);
                this.Match(preanalisis.getValor());
            }
        } catch (LexicalError ex) {
            this.hayErrores = true;
            throw new LexicalError("Error Léxico en [" + this.getPosicion() + "]: el símbolo no
pertenece al alfabeto");
        } catch (Exception ex) {
            this.hayErrores = true;
            throw new LexicalError("Error Léxico en [" + this.getPosicion() + "]:
"+ex.getMessage());
        }

        return nuevo;
    }
```

```
    /* ----------------------- GETTERS Y SETTERS ------------------------ */
    public String getRegex() {
        return regex;
    }

    public void setRegex(String regex) {
        this.setPosicion(0);
        this.regex = regex;
        this.lexico = new Lex(regex, alfabeto); // creamos el analizador léxico

        try {
            // creamos el analizador léxico
            this.preanalisis = nextSymbol(); // obtenemos el primer símbolo desde el analizador
léxico
        } catch (LexicalError ex) {
            this.hayErrores = true;
            this.errMsg =
                    "Se produjo un error FATAL en el traductor. La generación del AFN no puede
continuar\n"+
                    "--> "+ex.getMessage();

            System.out.println(this.getErrMsg());
            this.abort();
        }
        automata = new Automata();
    }

    public Token getPreanalisis() {
        return preanalisis;
    }

    public void setPreanalisis(Token preanalisis) {
        this.preanalisis = preanalisis;
    }

    public Alfabeto getAlfabeto() {
        return alfabeto;
    }

    public void setAlfabeto(Alfabeto alfabeto) {
        this.alfabeto = alfabeto;
    }

    public void setAlfabetoString(String alpha) {
        this.alfabeto = new Alfabeto(alpha);
    }
    public Automata getAutomata() {
        return automata;
    }

    public void setAutomata(Automata Aut) {
        this.automata = Aut;
    }

    public int getPosicion() {
        return posicion;
    }

    public void setPosicion(int posicion) {
        this.posicion = posicion;
    }

    public void incPosicion() {
        this.setPosicion(this.posicion+1);
    }

    public boolean isHayErrores() {
        return hayErrores;
    }

    public String getErrMsg() {
        return errMsg;
    }
}
package traductor;

import exceptions.LexicalError;

/**
 *
```

```
 * Analizador Léxico del traductor dirigido por sintaxis de expresiones regulares
 * a AFNs
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class Lex {

    /**
     * Buffer de String que contiene la expresión regular a analizar
     */
    private StringBuffer regex;

    /**
     * Lista de caracteres que conforman el alfabeto de la expresión regular<br><br>
     * En conjunto con la propiedad "specials" forman la "Tabla de Símbolos"
     * del traductor
     */
    private Alfabeto Alpha;

    /**
     * Símbolos especiales del lenguaje
     */
    private String specials;

    /**
     * Constructor de la clase del analizador léxico
     * @param regex Expresión regular que se quiere analizar
     * @param alfabeto Cadena de símbolos que constituyen el alfabeto
     */
    public Lex(String regex, String alfabeto) {
        this.regex = new StringBuffer(regex);
        this.Alpha = new Alfabeto(alfabeto);
        this.specials = "*+?|()";
    }


    /**
     * Constructor de la clase del analizador léxico, con Alfabet ya creado
     * en una ámbito superior
     * @param regex Expresión regular que se quiere analizar
     * @param alfabeto Objeto Alfabeto que contiene la lista completa de símbolos del mismo
     */
    public Lex(String regex, Alfabeto alfabeto) {
        this.regex = new StringBuffer(regex);
        this.Alpha = alfabeto;
        this.specials = "*+?|()";
    }

    /**
     * Consume la entrada y devuelve el siguiente a procesar. Si no se trata de
     * un token que pertenezca al alfabeto, entonces se lanza una Excepción.
     * <br><br>
     *
     * @return El siguiente caracter de la expresión regular
     * @throws java.lang.Exception Se lanza una excepción si el siguiente
 símbolo
     *                             no pertenece al alfabeto o a alguno de los
     *                             símbolos conocidos
     */
    public Token next() throws LexicalError {

        String s = consume();
        Token siguiente;

        if (s.equalsIgnoreCase(" ") || s.equalsIgnoreCase("\t")) {
            siguiente = next();         // Los espacios y tabuladores se ignoran

        } else if (this.specials.indexOf(s) >= 0 || this.Alpha.contiene(s) || s.length() == 0) {
            siguiente = new Token(s);   // se procesan los simbolos del alfabeto o especiales

        } else {
            String except = "El símbolo "+s+" no es válido";
            throw new LexicalError(except);
        }

        return siguiente;
    }
```

```
    /**
     * Método que consume un carácter de la expresión regular. Si retorna la
     * cadena vacía es porque ya no hay nada que consume. <br> <br>
     *
     * Consume consiste en extraer la primera letra de la expresión regular
     * y devolverla como un String.
     *
     * @return El siguiente caracter en la expresión regular
     */
    private String consume() {

        String consumido = "";

        if (this.regex.length() > 0) {
            consumido = Character.toString( this.regex.charAt(0) );
            this.regex.deleteCharAt(0);
        }

        return consumido;
    }


    /**
     * Obtener el Alfabeto utilizado
     * @return Alpha El Alfabeto completo utilizado
     */
    public Alfabeto getAlpha() {
        return Alpha;
    }

    /**
     * Obtener la expresión regular
     * @return regex Expresión regular
     */
    public StringBuffer getRegex() {
        return regex;
    }


    /**
     * Obtener la expresión regular (en String)
     * @return regex Expresión regular, como un String
     */
    public String getRegexString() {
        return regex.toString();
    }

    /**
     * Obtener caracteres especiales
     * @return specials Los operadores y simbolos especiales del lenguaje
     */
    public String getSpecials() {
        return specials;
    }
}
package traductor;

/**
 * Tipo "enum" que enumera los diferentes tipos de de token que se pueden
 * manipular. <br><br>
 *
 * <ul>
 *  <li> <b>'*': </b>   cerradura de kleene </li>
 *  <li> <b>'+': </b>   cerradura positiva de kleene </li>
 *  <li> <b>'?': </b>   cero o una instancia </li>
 *  <li> <b>'|': </b>   disyunción </li>
 *  <li> <b>'(': </b>   paréntesis izquierdo </li>
 *  <li> <b>')': </b>   paréntesis derecho </li>
 *  <li> <b>'ALFA': </b>   cualquier letra del alfabeto </li>
 * </ul>
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public enum TipoToken {
    NONE,       // token erróneo
    KLEENE,     // '*' --> cerradura de kleene
    PLUS,       // '+' --> cerradura positiva de kleene
    CEROUNO,    // '?' --> Cero o una instancia
    OR,         // '|' --> Disyunción
```

```
    PARI,       // '(' --> Paréntesis izquierdo
    PARD,       // ')' --> Paréntesis derecho
    ALFA,       // Cualquier letra del alfabeto
    FIN         // Fin de la expresión regular
}
package traductor;

/**
 * Clase que encapsula a cada componente enviado desde el analizador léxico
 * al analizador sintáctico para su procesamiento. <br> <br>
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class Token implements Comparable<Token> {


    private TipoToken tipo;


    private String valor;


    /**
     * Constructor principal del Token a partir del símbolo que se le pasa. Se
     * asume que el símbolo es válido ya que se deja la validcación al analizador
     * léxico.
     * @param tipo Indica el tipo de token definidas por el enum TipoToken.
     */
    public Token(String simbolo) {
        this.valor = simbolo;
        this.setTipo(simbolo);
    }

    /**
     * Función que retorna el tipo de token actual
     * @return Retorna el tipo de token
     */
    public TipoToken getTipo() {
        return tipo;
    }

    /**
     * Método que retorna el valor (char) del token actual.
     * @return
     */
    public String getValor() {
        return valor;
    }

    /**
     * Establece el tipo de token
     * @param tipo Tipo del token actual
     */
    public void setTipo(TipoToken tipo) {
        this.tipo = tipo;
    }

    /**
     * Valor (en char) del tipo de token actual
     * @param valor Caracter que representa el tipo de token
     */
    public void setValor(String valor) {
        this.valor = valor;
        this.setTipo(valor);
    }

    /**
     * Método abstracto de la clase Comparable implementado por Token para poder
     * utilizar el operador == para las comparaciones <br><br>
     *
     * @param t Token con el que se comparará el actual.
     * @return <ul> <li><b>0 (Cero)</b> si son  iguales         </li>
     *              <li><b>-1 (Menos Uno)</b> si no son iguales </li>
     *         </ul>
     */
    public int compareTo(Token t) {
        if (this.getTipo() == t.getTipo()
                && this.getValor().compareTo(t.getValor()) == 0 ) {
            return 0;
```

```
        } else {
            return -1;
        }
    }

    private void setTipo(String simbolo) {

        if (simbolo.isEmpty()) {
            this.tipo = TipoToken.FIN;
        } else {

            switch (simbolo.charAt(0)) {
                case '*':
                    this.tipo = TipoToken.KLEENE;
                    break;
                case '+':
                    this.tipo = TipoToken.PLUS;
                    break;
                case '?':
                    this.tipo = TipoToken.CEROUNO;
                    break;
                case '|':
                    this.tipo = TipoToken.OR;
                    break;
                case '(':
                    this.tipo = TipoToken.PARI;
                    break;
                case ')':
                    this.tipo = TipoToken.PARD;
                    break;
                default:
                    this.tipo = TipoToken.ALFA;
                    this.valor = simbolo;
                    break;
            }
        }
    }
}
```

2. Paquete afgenjava

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package afgenjava;

import exceptions.AutomataException;
import java.util.*;

/**
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancia ({@link fernandomancia@gmail.com})
 */
public class AlgMinimizacion {

    Automata AFD;
    public AlgMinimizacion(Automata a){
        this.AFD = a;
    }


/** ALGORITMO DE MINIMIZACION
 * Los siguientes metodos son usados en el algoritmo de minimizacion.
 *
 **/


   /***
    * Este método realiza el algoritmo de minimización definido en el
    * libro en la sección 3.9.6.
    * Retorna un nuevo Automata que acepta el mismo lenguaje (del automata
    * pasado en el constructor.)y tiene el menor número de estados posible.
    * Pasos:
    *   1) Se empieza con una particion inicial P = {NO_FINALES, FINALES}
    *   2) Pnew = Por c/ grupo G en P particionarG en subgrupos de forma que s y t
    *       se encuentren en el mismo subgrupo, si y solo si para todo el
    *       alfabeto s y t tienen transiciones hacia los mismos grupos de P.
    *   3) Si Pnew = P, entonces Pfinal = Pnew, e ir al paso 4. Sino, ir al paso 2
    *   4) Elegir un estado de c/ grupo como representante y actualizar
    *      los enlaces
    *
    * @return Automata (Un nuevo automata minimizado)
    * @throws exceptions.AutomataException
    */
   public Automata minimizar() throws AutomataException{
       ArrayList<ListaEstados> anterior = new ArrayList<ListaEstados>();
       ArrayList<ListaEstados> actual = new ArrayList<ListaEstados>();

       int nro_est = 0;
       ListaEstados nofinales = AFD.getNoFinales();
       ListaEstados finales = AFD.getFinales();

       if(nofinales != null && nofinales.cantidad() > 0){
            nofinales.setId(nro_est++);
            anterior.add(nofinales);
       }

       if(finales != null && finales.cantidad() > 0){
            finales.setId(nro_est++);
            anterior.add(finales);
       }

       boolean seguir = true;
       while(seguir){

           int cant =0;
           for(ListaEstados cadaLista: anterior){
                Iterator it = separarGrupos(anterior, cadaLista);
                while(it != null && it.hasNext()){
                    ListaEstados list= (ListaEstados)it.next();
                    list.setId(cant++);
                    actual.add(list);
                }
```

```
        }

        if(anterior.size() == actual.size()){
            seguir = false;
        }else{
            anterior = actual;
            actual = new ArrayList<ListaEstados>();
        }
    }
    //Fin del Algoritmo de Minimizacion.


    //Ahora se convierte "actual"  en "Automata"
    //Primero creamos los estados
    Automata AFDM = new Automata();
    Iterator it = actual.iterator();
    while(it.hasNext()){
         ListaEstados lest = (ListaEstados) it.next();
         Estado nuevo = new Estado(lest.getId() , false, false,false);

         //Es estado inicial
         try{
             lest.getEstadoInicial();
             nuevo.setEstadoinicial(true);
             AFDM.setInicial(nuevo);
         }catch(Exception ex){
             nuevo.setEstadoinicial(false);
         }

         //Es estado final
         if(lest.getEstadosFinales().cantidad() > 0){
             nuevo.setEstadofinal(true);
             AFDM.getFinales().insertar(nuevo);
         }else{
             nuevo.setEstadofinal(false);
         }
         AFDM.addEstado(nuevo);
    }

    //Segundo, creamos los enlaces
    it = actual.iterator();
    while(it.hasNext()){
         ListaEstados lest = (ListaEstados) it.next();
         Estado estado_afdm  = AFDM.getEstadoById(lest.getId());
         Estado representante = lest.get(0);

         Iterator itenlaces = representante.getEnlaces().getIterator();
         while (itenlaces.hasNext()){
             Enlace e = (Enlace) itenlaces.next();
             ListaEstados lista_destino = enqueLista(actual, e.getDestino());
             Estado est_destino = AFDM.getEstadoById(lista_destino.getId());
             Enlace nuevo_enlace = new Enlace(estado_afdm, est_destino, e.getEtiqueta());
             estado_afdm.addEnlace(nuevo_enlace);
         }
    }

    return AFDM;
}


/***
 *   Método para separar una "lista" en varios grupos.
 * Para cada estado de la lista, se itera sobre todos sus enlaces y a partir
 * de eso se obtiene información para crear un nuevo subgrupo o agragar a
 * uno existente.
 *
 * @param ListasActuales (Todas las listas actuales)
 * @param laLista (la lista que será separa en grupos)
 * @return Iterador de las sublistas en que se dividió laLista
 */
public Iterator separarGrupos(ArrayList<ListaEstados> todas,
                              ListaEstados lista){
     Hashtable listasNuevas = new Hashtable();
     for(Estado estado : lista){
         String claveSimbolos = "";
         String claveEstados = "";

         for(Enlace enlace : estado.getEnlaces()){
             Estado dest = enlace.getDestino();
             ListaEstados tmp = enqueLista(todas, dest);
```

```
            claveSimbolos += enlace.getEtiqueta().trim();
            claveEstados += tmp.getId();

        }
        String clave = generarClaveHash(claveSimbolos, claveEstados);
        if(listasNuevas.containsKey(clave)){
            ((ListaEstados)listasNuevas.get(clave)).insertar(estado);
        }else{
            ListaEstados nueva = new ListaEstados();
            nueva.insertar(estado);
            listasNuevas.put(clave, nueva);
        }
     }
     return listasNuevas.values().iterator();
}



/**
 * Método mágico que genera una clave que sera la clave de un hash
 * que tendrá las sublistas que pertenecen a un mismo grupo.
 * Todas las listas que  generen el mismo hash tendrán la misma clave y por
 * ende estarán en la misma lista dentro del hash.
 *
 * @param simbolos
 * @param estados
 * @return
 */
public String generarClaveHash(String simbolos, String estados ){
    String cadenaFinal = "";

     char est[] = estados.toCharArray();
     char c[] = simbolos.toCharArray();
     boolean hayCambios = true;
     for (int i = 0; hayCambios ; i++) {
         hayCambios = false;
         for (int j = 0; j < c.length - 1; j++) {
           if (c[j] > c[j + 1]) {

               //intercambiar(arreglo, j, j+1);
               //ini intercambiar
               char tmp = c[j+1];
               c[j+1] = c[j];
               c[j] = tmp;

               char tmpEst = est[j+1];
               est[j+1] = est[j];
               est[j] = tmpEst;
               //fin intercambiar

               hayCambios = true;
           }
         }
     }
    cadenaFinal = String.copyValueOf(c) + String.copyValueOf(est);
    return cadenaFinal;
}



/***
 * Método que retorna una lista de estado de entre las "listas", en la que
 * se encuentra un "estado" en particular.
 *
 * @param listas
 * @param estado
 * @return
 */
public ListaEstados enqueLista(ArrayList<ListaEstados> listas, Estado estado){
     for(ListaEstados lista : listas){
         try{
             lista.getEstadoById(estado.getId());
             return lista;
         }catch(Exception ex){}
     }
     return null;
}
```

```
}
/*
 * AlgSubconjuntos.java
 *
 * Created on 8 de noviembre de 2008, 04:42 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package afgenjava;

import exceptions.AutomataException;
import graphviz.GraphViz;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Vector;
import traductor.Analizador;
import traductor.Token;

/**
 *
 * @author Administrador
 */
public class AlgSubconjuntos {
    Automata AFN;
    /**
     *  AFD, Matriz final que representa el AFD.
     **/
    private Dtrans dtrans;


    /**
     *  Lista de estados que se ira formando para el AFD
     */
    ArrayList<ListaEstados> Destados;



    /** Creates a new instance of AlgSubconjuntos */
    public AlgSubconjuntos(Automata AFN) {
        this.AFN = AFN;
        dtrans = new Dtrans();
        Destados = new ArrayList();
    }



    /**
     * Implementación del algoritmo de Subconjuntos.
     * Retorna un objeto Dtrans que es la matriz de transiciones.
     * Este Dtrans se puede convertir tambien a un "Automata"
     *
     * @return
     * @throws exceptions.AutomataException
     */
    public Dtrans ejecutar() throws AutomataException{
        Iterator it;
        Token simbolo;
        ListaEstados U;

        Estado est_inicial = AFN.getEstados().getEstadoInicial();
        ListaEstados list_est = e_cerradura(est_inicial, new ListaEstados());
        list_est.setId(0);
        Destados.add(list_est);

        while (hayEstadosSinMarcar()){
            DtransClave clave;
            ListaEstados T = estadoSinMarcar();
            T.setMarcado(true);

            it = AFN.getAlpha().iterator();
            while(it.hasNext()){
                simbolo = new Token((String)it.next());
                U = e_cerradura(mover(T, simbolo));
                if(U == null){
```

```
                continue;
            }
            int id_U = estaEnDestados(U);
            if(id_U == -1){
                U.setMarcado(false);
                U.setId(Destados.size());
                Destados.add(U);
            }else{
                U.setId(id_U);
            }
            clave = new DtransClave(T,simbolo);
            dtrans.setValor(clave, U);
        }
    }
    return this.dtrans;
}


/**
 * Ejecuta el algoritmo "e_cerradura(s)"
 * En donde a partir de un estado s, retorna una lista de estados
 * que se forma de recorrer desde el estado s por transiciones vacias.
 * Implementación recursiva, ya que debe recorrer los nodos por donde
 * exista enlaces vacios de la misma forma.
 *
 *
 * @param s Estado que se agrega y recorre por sus vacios.
 * @param listaActual (lista de estados donde se van agregando. Al inicio
 *          está vacia
 * @return La lista de estados por los que se recorre  mediante vacio desde
 *          el estado "s"
 */
public ListaEstados e_cerradura(Estado s, ListaEstados listaActual) {
    Iterator it = s.getEnlaces().getIterator();
    ListaEstados listaNueva = null;
    while(it.hasNext()){
        Enlace e = (Enlace) it.next();
        if(e.getEtiqueta().compareTo(CONSTANS.getVacio()) == 0){
            listaNueva = e_cerradura(e.getDestino(), listaActual);
            listaActual = concatListas(listaActual, listaNueva );

        }
    }
    listaActual.insertar(s);
    return listaActual;
}


/***
 *  Implementacion de e_cerradura(ListaEstados) del Algoritmo de Subconjuntos.
 *  Recibe una lista de estados y por cada estado aplica el
 * e_cerradura(estado, new ListaEstados()).
 *  Es decir, por cada estado de la lista recibida se recorre recursivamente por
 * los enlaces "vacio" y se genera una nueva lista.
 *
 * @param T
 * @return
 */
public ListaEstados e_cerradura(ListaEstados T){
    if(T == null){
        return null;
    }

    ListaEstados lista_ret = new ListaEstados();
    Iterator it = T.getIterator();
    Estado act;

    while(it.hasNext()){
        act = (Estado) it.next();
        lista_ret = concatListas(lista_ret, e_cerradura(act, new ListaEstados()));
    }

    return lista_ret;
}


/***
 *  Realiza el algoritmo mover que se propone en el capítulo 3.
 * Dado una lista de estados "T" y un símbolo "a" del alfabeto, mover
 * retorna una lista con los estados en donde existe una transición por "a"
```

```
     * desde alguno de los estados que hay en "T".
     *
     * @param T Lista de Estados.
     * @param a Símbolo del alfabeto.
     * @return Lista de Estados alos que se puede ir por a desde c/ estado en T
     */
    public ListaEstados mover(ListaEstados T, Token a){
        Iterator itEstados = null;
        Iterator itEnlaces = null;
        Estado estado = null;
        Enlace enlace = null;
        ListaEstados lista = new ListaEstados();

        itEstados = T.getIterator();
        while(itEstados.hasNext()){
            estado = (Estado) itEstados.next();
            itEnlaces = estado.getEnlaces().getIterator();

            while(itEnlaces.hasNext()){
                enlace = (Enlace) itEnlaces.next();
                if(enlace.getEtiqueta().compareTo(a.getValor()) == 0){
                    lista.insertar(enlace.getDestino());
                }
            }
        }
        if(lista.size() == 0){
            return null;
        }else{
            return lista;
        }
    }


    /**
     * Verifica si existe algún estado sin marcar en Destados.
     * Es la condición de parada del algoritmo de subconjuntos.
     *
     * @return
     */
    private boolean hayEstadosSinMarcar(){
        Iterator it = Destados.iterator();
        ListaEstados list_est;
        while (it.hasNext()){
            list_est = (ListaEstados) it.next();
            if(!list_est.isMarcado()){
                return true;
            }
        }
        return false;
    }

    /**
     * Retorna el primer estado sin marcar que encuentra en Destados.
     * Si no existe ninguno sin marcar, lanza una excepción.
     *
     * @return
     * @throws exceptions.AutomataException
     */
    private ListaEstados estadoSinMarcar() throws AutomataException{
        Iterator it = Destados.iterator();
        ListaEstados list_est;
        while (it.hasNext()){
            list_est = (ListaEstados) it.next();
            if(!list_est.isMarcado()){
                return list_est;
            }
        }
        throw new AutomataException("No hay Lista de Estados sin marcar en Destados.");
    }


    /***
     *  Metodo que retorna el id de la lista de estados U dentro de
     * Destados, si es que U no esta en la lista de estados retorna -1.
     *
     * @param U Lista de estados
     * @return El id de la lista U dentro de Destados
     */
    private int estaEnDestados(ListaEstados U){
        Iterator it = Destados.iterator();
```

```
    ListaEstados tmp;
    while(it.hasNext()){
        tmp = (ListaEstados)it.next();
        if(tmp.compareTo(U) == 0){
            return tmp.getId();
        }
    }
    return -1;
}


public static ListaEstados concatListas(ListaEstados A, ListaEstados B){
    ListaEstados ret = new ListaEstados();
    Iterator it;
    Estado est_tmp, test;

    if(A != null){
        it = A.getIterator();
        while(it.hasNext()){
            est_tmp = (Estado) it.next();
            try{
                ret.getEstadoById(est_tmp.getId());
            }catch(Exception ex){
                ret.insertar(est_tmp);
            }
        }
    }

    if(B != null){
        it = B.getIterator();
        while(it.hasNext()){
            est_tmp = (Estado) it.next();
            try{
                ret.getEstadoById(est_tmp.getId());
            }catch(Exception ex){
                ret.insertar(est_tmp);
            }
        }
    }

    return ret;
}



/**
 * Eliminación de los estados inalcanzables.
 * Método estatico que recibe un AFD y retorna un nuevo AFD sin los estados
 * inalcanzables. Necesita del metodo estatico "recorrer"
 *
 * @param AFD
 * @return AFD sin estados inalcanzables
 */
public static Automata eliminar_estados_inalcanzables(Automata AFD){
    Estado inicial = AFD.getInicial();
    AFD.getEstados().resetVisitas();
    visitarRecursivo(inicial);

    Automata AFDNEW = new Automata();
    AFDNEW.setAlpha(AFD.getAlpha());
    AFDNEW.setRegex(AFD.getRegex());

    Iterator it = AFD.getEstados().getIterator();
    while(it.hasNext()){
        Estado e = (Estado)it.next();
        if(e.isVisitado()){

            if(e.isEstadoinicial()){
               AFDNEW.setInicial(e);
            }
            if(e.isEstadofinal()){
                AFDNEW.getFinales().insertar(e);
            }
            AFDNEW.addEstado(e);
        }

    }

    return AFDNEW;
}
```

```
    /**
     * Método que marca como visitado un nodo con sus respectivos
     * hijos, lo hace recursivamente.
     *
     * @param Estado actual a marcar como visitado
     */
    public static void visitarRecursivo(Estado actual){
        if(!actual.isVisitado()){
            actual.setVisitado(true);
            Iterator it = actual.getEnlaces().iterator();
            while(it.hasNext()){
                Enlace enlace = (Enlace)it.next();
                visitarRecursivo(enlace.getDestino());
            }
        }
    }

}
package afgenjava;
import java.lang.StringBuffer;
import java.util.*;


/**
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})<br>
 * @author Fernando Mancia ({@link fernandomancia@gmail.com})
 */
public class Automata {

/*------------------------ ATRIBUTOS ------------------------*/

    /**
     * Lista de Estados que componen el automata
     */
    private ListaEstados estados;

    /**
     * Apuntador al Estado inicial del mismo
     */
    private Estado inicial;

    /**
     * Lista de apuntadores a estados finales
     */
    private ListaEstados finales;

    /**
     * Identificador del tipo de Automata representado. Puede ser: <br>
     * <ul>
     *  <li>AFN (Automata Finito No-Determinístico)        </li>
     *  <li>AFD (Automata Finito Determinístico)          </li>
     *  <li>AFDMin (Automata Finito Determinístico Mínimo) </li>
     * </ul>
     */
    private TipoAutomata tipo;

    // Los siguientes atributos podrían ser eliminados

    /**
     * Expresion regular representada
     */
    private String regex;

    /**
     * Alfabeto sobre el cual se define la expresión regular
     */
    private ArrayList<String> alpha;

    private String empty = CONSTANS.getVacio();


    // VARIABLES AUXILIARES

    private int level = 0;

/*---------------------------- CONSTRUCTORES ----------------------------*/
```

```
    /**
     * Constructor Vacío
     */
    public Automata() {
        this.estados = new ListaEstados();
        this.finales = new ListaEstados();
    }

    /**
     * Constructor de un automata simple. Compuesto por dos estados y un solo
     * enlace a través del simbolo especificado.
     * @param simbolo Expresion regular simple (de un solo caracter)
     */
    public Automata(String simbolo) {
        this.estados = new ListaEstados();

        Estado e1 = new Estado(0, true, false, false);
        Estado e2 = new Estado(1, false, true, false);
        Enlace enlace = new Enlace(e1, e2, simbolo);
        e1.addEnlace(enlace);

        this.estados.insertar(e1);
        this.estados.insertar(e2);

        // Actualización de apuntadores auxiliares
        this.inicial = e1;
        this.finales = new ListaEstados();
        this.finales.add(e2);
    }

    /**
     * Constructor auxiliar para automatas simples con especificación del tipo
     * de automata a construir.
     * @param simbolo Expresion regular simple (de un solo caracter)
     * @param tipo Especificación del tipo de automata en construcción
     */
    public Automata(String simbolo, TipoAutomata tipo) {
        this(simbolo);
        this.tipo = tipo;
    }

/*----------------------- OPERACIONES DE THOMPSON -----------------------*/

    /**
     * Implementación de la generación de automatas según la definición de
     * Thompson para la operación "|"
     *
     * @param A2 Automata a seguir como camino alternativo al actual
     */
    public void thompson_or(Automata A2){

        Automata A1 = this;



        // Obtenemos las referencias a los finales e iniciales correspondientes
        Estado final_A1 = A1.getFinales().getEstado(0);
        Estado final_A2 = A2.getFinales().getEstado(0);
        Estado inicial_A1 = A1.getInicial();
        Estado inicial_A2 = A2.getInicial();

        final_A1.setEstadofinal(false);
        final_A2.setEstadofinal(false);

        // Se crean 2 nuevos estados
        Estado estado_inicial = new Estado(0, true, false, false);
        Estado estado_final = new Estado(A1.estados.size()+A2.estados.size()+1, false, true, false);


        // Actualizar estados iniciales de A1 y A2
        A1.inicial.setEstadoinicial(false);
        A2.inicial.setEstadoinicial(false);

        // Se incrementan los numeros de ambos automatas
        A1.renumerar(1);
        A2.renumerar(A1.estados.size()+1);

        // Se crean los enlaces vacios desde el nuevo estado inicial

        // 1. Nuevo Inicio --> Inicio del Automata Actual
```

```
    estado_inicial.addEnlace(new Enlace(estado_inicial,
                                        inicial_A1,
                                        this.empty));

    // 2. Nuevo Inicio --> Inicio del Automata Alternativo
    estado_inicial.addEnlace(new Enlace(estado_inicial,
                                        inicial_A2,
                                        this.empty));

    // Se crean los enlaces desde los finales del Actual (A1) y el
    // alternativo (A2) hacia el Nuevo Estado Final.

    // 3. Fin del Actual (A1) --> Nuevo Estado Final
    final_A1.addEnlace( new Enlace( final_A1, estado_final, this.empty) );

    // 4. Fin del Alternativo (A2) --> Nuevo Estado Final
    final_A2.addEnlace( new Enlace( final_A2, estado_final, this.empty) );

    // Agregamos a A1 todos los estados de A2
    Iterator it = A2.estados.getIterator();
    while(it.hasNext()){
        A1.estados.insertar((Estado)it.next());
    }

    // Agregamos a A1 los nuevos estados creados.
    A1.estados.insertar(estado_inicial);
    A1.estados.insertar(estado_final);

    // Actualizar referencias auxiliares al inicial y al final del actual
    A1.inicial=estado_inicial;
    A1.getFinales().set(0, estado_final);
}

/**
 * Implementación de la generación de automatas según la definición de
 * Thompson para la operación de concatenación
 *
 * @param A2 Automata siguiente al actual
 */
public void thompson_concat(Automata A2){
    Automata A1 = this; //se agrega a este automata quedando A1 A2 osea this A2.

    // Obtener referencias al final de A1 y al inicial de de A2
    Estado final_A1   = A1.getFinales().getEstado(0);
    Estado inicial_A2 = A2.getInicial();

    // Se actualiza al estado inicial del Automata Siguiente (A2) para
    // que deje de ser inicial
    inicial_A2.setEstadoinicial(false);
    final_A1.setEstadofinal(false);

    // Renumeramos los estados del Automata siguiente
    int a1_estado_final = A1.estados.size() - 1;
    A2.renumerar(a1_estado_final);

    // Se fusiona el enlace inicial de A2 con el final de A1
    // 1. Primero agregamos los enlaces del inicio de A2, al final de A1
    Iterator <Enlace> enlaces_a2_inicio = inicial_A2.getEnlaces().getIterator();

    while(enlaces_a2_inicio.hasNext()){
        Enlace current = enlaces_a2_inicio.next();
        current.setOrigen(final_A1);
        final_A1.addEnlace(current);
    }

    // 2. Agregar los demás estados de A2, excepto su inicial, al automata A1
    Iterator <Estado> estados_a2 = A2.estados.getIterator();

    while(estados_a2.hasNext()){
        Estado est_a2 = estados_a2.next();

        // 2.1 Actualizar en el estado, todos los enlaces que apuntaban al
        //     inicio de A2 para que apunten al nuevo inicio, que es el final
        //     de A1 y a
        Iterator <Enlace> enlaces = est_a2.getEnlaces().getIterator();

        while(enlaces.hasNext()){
            Enlace current = enlaces.next();
            Estado current_destino = current.getDestino();
```

```
                // Si el destino de este enlace
                if (current_destino.getId() == inicial_A2.getId()) {
                    current.setDestino(final_A1);
                }
            }

            // Agregar el estado al automata actual
            if(est_a2.getId() != inicial_A2.getId()){
                A1.estados.insertar(est_a2);
            }
        }

        A1.getFinales().set(0, A2.getFinales().getEstado(0));
    }


    /**
     * Parte de las operaciones de implementación de kleene (*), plus (+) y
     * cerouno (?) que es común entre las tres. <br>
     *
     * Modifica el automata actual de la siguiente manera: <br>
     * <ul>
     *   <li>Agrega dos nuevos estados (uno al inicio y otro al final) </li>
     *   <li>Agrega dos nuevos enlaces vacíos
     *       <ul>
     *          <li>Uno para unir el nuevo inicio con el viejo</li>
     *          <li>Uno para unir el viejo fin con el nuevo</li>
     *       </ul>
     *   </li>     *
     * </ul>     *
     */
    public void thompson_common() {

        // Se realiza la operacion sobre el mismo objeto.
        Automata A1 = this;


        // Se incrementan en 1 los estados
        A1.renumerar(1);

        // Se agregan 2 Estados nuevos (Un inicial y uno al final)
        Estado estado_inicial = new Estado(0, true, false, false);
        Estado estado_final   = new Estado(A1.estados.size()+1, false, true, false);

        Estado ex_estado_inicial = A1.getInicial();
        Estado ex_estado_final   = A1.getFinales().getEstado(0);

        ex_estado_inicial.setEstadoinicial(false);
        ex_estado_final.setEstadofinal(false);

        // Agregar vacíos al comienzo y al final
        estado_inicial.addEnlace(new Enlace(estado_inicial,
                                            ex_estado_inicial,
                                            this.empty));

        ex_estado_final.addEnlace(new Enlace(ex_estado_final,
                                             estado_final,
                                            this.empty));


        // Actualizar referencias auxiliares

this.inicial = estado_inicial;
        this.finales.set(0, estado_final);

        A1.estados.insertar(estado_inicial);
        A1.estados.insertar(estado_final);


    }

    /**
     * Implementación de la operación '?' sobre el automata actual. <br>
     *
     * Consiste en Agregar al automata actual enlaces vacios al comienzo y al
     * final ademas de un enlace vacio entre el inicio y el final para permitir
     * que se pueda recorrer o no el Automata actual, tal como lo especifica la
     * operación ? <br>
     *
     * Observación: La operación '?' no está prevista entre las operaciones
```

```
 * originales de Thompson por lo que implementamos nuestra propia versión
 *
 */
public void thompson_cerouno() {

    // Agrega dos nuevos estados al inicio y al final y los enlaza al
    // inicio y al final del automata original respectivamente,
    // por medio del símbolo vacío
    this.thompson_common();

    // Se agregan un enlace vacío entre el nuevo inicio y el nuevo fin
    this.inicial.addEnlace(new Enlace(this.inicial,
                                      this.finales.getEstado(0),
                                      this.empty));
}

/**
 *
 */
public void thompson_plus() {

    Estado inicio_original = this.inicial;
    Estado fin_original    = this.getFinales().getEstado(0);

    // Agrega dos nuevos estados al inicio y al final y los enlaza al
    // inicio y al final del automata original respectivamente,
    // por medio del símbolo vacío
    this.thompson_common();

    // Se agregan un enlace vacío entre el fin original y inicio original
    // para que se recorra el actual por lo menos una vez y pueda ser
    // recorrido más veces como lo especifica la operación '+'
    fin_original.addEnlace(new Enlace(fin_original,
                                      inicio_original,
                                      this.empty));
}

public void thompson_kleene(){
    Estado inicio_original = this.inicial;
    Estado fin_original    = this.finales.get(0);

    // Agrega dos nuevos estados al inicio y al final y los enlaza al
    // inicio y al final del automata original respectivamente,
    // por medio del símbolo vacío
    this.thompson_common();

    // Se agrega un enlace vacío entre el fin original y inicio original
    // para que se recorra el actual más veces como lo especifica
    // la operación *
    fin_original.addEnlace(new Enlace(fin_original,
                                      inicio_original,
                                      this.empty));

    // Se agregan un enlace vacío entre el nuevo inicio y el nuevo fin
    this.inicial.addEnlace(new Enlace(this.inicial,
                                      this.finales.getEstado(0),
                                      this.empty));
}


/* --------------- GETTERS Y SETTERS -------------------- */



/**
 * Obtener el estado referenciado por el índice correspondiente
 *
 * @param index indice en el listado donde se encuentra el estado.
 * @return Estado guardado en index
 */
public Estado getEstado(int index){
    return this.estados.getEstado(index);
}

public ListaEstados getEstados() {
    return this.estados;
}

public Estado getEstadoById(int id) {
    return this.estados.getEstadoById(id);
```

```
    }

    /**
     * Obtener la lista de estados finales.
     *
     * En el AFN, siempre hay un solo estado final, cuya referencia se guarda en
     * la primera posición de este listado.
     * @return ListaEstados Lista de Estados finales del Automata
     */
    public ListaEstados getFinales() {
        return finales;
    }

    /**
     * Obtiene la lista de estados no finales.
     **/
    public ListaEstados getNoFinales(){
        ListaEstados lista = new ListaEstados();
        for(Estado x : estados){
            if(!x.isEstadofinal()){
                lista.insertar(x);
            }
        }
        return lista;
    }


    /**
     * Obtener el estado inicial del automata.
     *
     * @return Estado inicial del automata
     */
    public Estado getInicial() {
        return inicial;
    }

    public void setInicial(Estado ini) {
        this.inicial = ini;
    }

    public ArrayList<String> getAlpha() {
        return this.alpha;
    }

    public String getRegex() {
        return this.regex;
    }


    public void setAlpha(ArrayList<String> alpha) {
        this.alpha = alpha;
    }

    public void setRegex(String regex) {
        this.regex = regex;
    }


    /**
     * Renumera los identificadores del Automata incrementando su valor según un
     * incremento dado.
     *
     * @param incremento para renumerar los estados del automata.
     */
    public void renumerar(int incremento){

        //Renumerar Estados
        Iterator it = this.estados.getIterator();
        while (it.hasNext()){
            Estado e = (Estado) it.next();
            e.setId(e.getId()+incremento);
        }

    }

/* TEST */

    public String imprimir(){

        String result ="";
```

```
        Iterator it = this.estados.getIterator();
        while (it.hasNext()){
            Estado e = (Estado) it.next();
            result += "\nE."+ e.getId();

            if (e.isEstadoinicial()) {
                result += "(ini)";
            }

            if (e.isEstadofinal()) {
                result += "(fin)";
            }


            result+="\n";

            Iterator itenlaces = e.getEnlaces().getIterator();
            while(itenlaces.hasNext()){
                Enlace enlace = (Enlace) itenlaces.next();
                result +="\t"+
                        enlace.getOrigen().getId() + " ---"+enlace.getEtiqueta()+"---> " +
enlace.getDestino().getId()+"\n";
            }
        }
        return result;
    }


    private void eliminarEstado(Estado e){

        for(Estado est: this.estados){
            for(Enlace enlace: est.getEnlaces()){
                if( e.getId() != est.getId() && enlace.getDestino().getId() == e.getId()){
                        est.eliminarEnlace(enlace);
                }
            }
        }
    }


    /***
     * Método que elimina de este Automata los estados muertos, es decir, los
     * estados en el que todos sus enlaces van a si mismo y no es estado final.
     */
    public void eliminar_estados_muertos(){

       for(Estado e : this.getEstados()){
           if(e.esEstadoMuerto()){
               eliminarEstado(e);
           }
       }
   }


    public ListaEnlaces getEnlaces(){
        ListaEnlaces ret = new ListaEnlaces();

        for(Estado est: getEstados()){
            for(Enlace enlace: est.getEnlaces()){
                ret.add(enlace);
            }
        }

        return ret;
    }

    /**
     * Genera un String que puede ser utilizado para graficar con el GraphViz<br><br>
     *
     * Ejemplo: <br><br>
     * <code>
     * digraph test123 {
     *         a -> b -> c;
     *         a -> {x y};
     *         b [shape=box];
     *         c [label="hello\nworld",color=blue,fontsize=24,
     *              fontname="Palatino-Italic",fontcolor=red,style=filled];
     *         a -> z [label="hi", weight=100];
     *         x -> z [label="multi-line\nlabel"];
```

```
 *         edge [style=dashed,color=red];
 *         b -> x;
 *         {rank=same; b x}
 * }
 * </code>
 *
 * @return String del grafo formateado para dot (GraphViz)
 */
public String imprimirGraphViz(){

    String result_header = "Digraph AFN {\n" +
            "\trankdir=LR;\n\toverlap=scale;\n";

    String result_nodes = "";
    String result_edges = "";

    Iterator it = this.estados.getIterator();
    while (it.hasNext()){
        Estado e = (Estado) it.next();
        String shape = "circle";

        if (e.isEstadofinal()) {
            shape = "doublecircle";
        }

        result_nodes+=e.getId() + " [shape="+shape+"];\n";

        shape="circle";

        Iterator itenlaces = e.getEnlaces().getIterator();
        while(itenlaces.hasNext()){

            Enlace enlace = (Enlace) itenlaces.next();

            Estado orig = enlace.getOrigen();
            Estado dest = enlace.getDestino();
            String label = enlace.getEtiqueta();

            result_edges += orig.getId() + " -> " + dest.getId() +
                            " [label = \""+label+"\" ];\n";

        }
    }
    String result = result_header + result_nodes + result_edges + "}";
    return result;
}


/**
 * Genera un automata sencillo de prueba.
 * @return
 */
public static Automata dameAutomata(){
    Automata A1 = new Automata();
    A1.estados.insertar(new Estado(0,true,false, false));
    A1.estados.insertar(new Estado(1,true,false, false));
    A1.estados.insertar(new Estado(2,true,false, false));
    A1.estados.insertar(new Estado(3,true,false, false));
    A1.estados.insertar(new Estado(4,true,false, false));
    A1.estados.insertar(new Estado(5,true,false, false));

    //Estado 0
    A1.estados.getEstadoById(0).addEnlace( new Enlace(A1.estados.getEstadoById(0),

                            A1.estados.getEstadoById(1), "a"));

    A1.estados.getEstadoById(0).addEnlace( new Enlace(A1.estados.getEstadoById(0),
                                    A1.estados.getEstadoById(2), "b"));


    //Estado 1 y 2
    A1.estados.getEstadoById(1).addEnlace( new Enlace(A1.estados.getEstadoById(1),
                                    A1.estados.getEstadoById(3), "a"));

    A1.estados.getEstadoById(2).addEnlace( new Enlace(A1.estados.getEstadoById(2),
                                    A1.estados.getEstadoById(4), "a"));


    //Estado 3 y 4
    A1.estados.getEstadoById(3).addEnlace( new Enlace(A1.estados.getEstadoById(3),
```

```
                                            A1.estados.getEstadoById(5), "b"));

        A1.estados.getEstadoById(4).addEnlace( new Enlace(A1.estados.getEstadoById(4),
                                            A1.estados.getEstadoById(5), "a"));
        return A1;
    }

    public TipoAutomata getTipo() {
        return tipo;
    }

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

    public void addEstado(Estado e){
        this.estados.insertar(e);
    }

    public int getLevel() {
        return level;
    }

    public void setLevel(int level) {
        this.level = level;
    }
}
/*
 * CONSTANS.java
 *
 * Created on 10 de noviembre de 2008, 05:01 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package afgenjava;

/**
 *
 * @author Operador
 */
public class CONSTANS {

    private static String vacio = "(vacio)";
    private static String graphViz = "/usr/bin/dot";

    public static String getVacio() {


        return vacio;
    }

    public static void setVacio(String aVacio) {
        vacio = aVacio;
    }

    public static String getGraphViz() {
        return graphViz;
    }

    public static void setGraphViz(String aGraphViz) {
        graphViz = aGraphViz;
    }


    /** Creates a new instance of CONSTANS */
    public CONSTANS() {
    }

}
/*
 * Dtrans.java
 *
 * Created on 11 de noviembre de 2008, 01:03 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
```

```
package afgenjava;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Vector;
import traductor.Token;

/**
 * Representa la matriz de transiciones.
 * Está implementada con una hashtable en donde,
 *  La clave es : (ListaEstados ID_LISTA, Token SIMBOLO_ALPHA)
 *  El valor es : (ListaEstados ID_LISTA)
 *
 * En esta Matriz se representa para todos los estados ("A", "B", "C", etc)
 * a que estado destino van por cada uno de los símbolos del alfabeto.
 *
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 */
public class Dtrans {
    Hashtable dtrans;

    /** Creates a new instance of Dtrans */
    public Dtrans() {
        dtrans = new Hashtable();
    }

    /**
     * Retorna el valor(ListaEstados) apartir de la clave (ListaEstados, Token)
     *
     * @param clave
     * @return
     */
    public ListaEstados obtenerValor(DtransClave clave){
        return obtenerValor(clave.getIndiceEstados(), clave.getIndiceToken());
    }

    public ListaEstados obtenerValor(ListaEstados lista, Token token){
        DtransClave comparar = new DtransClave(lista, token);
        Enumeration en = dtrans.keys();
        DtransClave clave;
        while(en.hasMoreElements()){
            clave = (DtransClave)en.nextElement();
            if(clave.compareTo(comparar) == 0){
                return (ListaEstados) dtrans.get(clave);
            }
        }
        return null;
    }


    public void setValor(DtransClave clave, ListaEstados valor){
        dtrans.put(clave, valor);
    }


    /**
     *  Método que convierte el Dtrans en un "Automata", ya que las listas
     * de estados A,B,C, etc son los estados del nuevo Automata creado.
     *
     * @return Automata convertido.
     */
    public Automata convertAutomata(){
        Automata a = new Automata();

        Enumeration en = dtrans.keys();
        while(en.hasMoreElements()){
            DtransClave clave = (DtransClave) en.nextElement();
            ListaEstados valor = obtenerValor(clave);

            int id_new_origen = clave.getIndiceEstados().getId();
            int id_new_dest = valor.getId();
            Estado st_new_origen, st_new_dest;

            try{
```

```
                st_new_origen = a.getEstadoById(id_new_origen);
            }catch(Exception ex){
                //No existe el estado entonces creamos
                st_new_origen = new Estado(id_new_origen,
                                           clave.getIndiceEstados().contieneInicial(),
                                           clave.getIndiceEstados().contieneFinal(),
                                           false);
                a.addEstado(st_new_origen);
                if(clave.getIndiceEstados().contieneInicial()){
                    a.setInicial(st_new_origen);
                }
                if(clave.getIndiceEstados().contieneFinal()){
                    a.getFinales().insertar(st_new_origen);
                }

            }


            try{
                 st_new_dest = a.getEstadoById(id_new_dest);
            }catch(Exception ex){
                //No existe el estado entonces creamos
                st_new_dest = new Estado(id_new_dest,
                                         valor.contieneInicial(),
                                         valor.contieneFinal(),
                                         false);
                a.addEstado(st_new_dest);
                if(valor.contieneInicial()){
                    a.setInicial(st_new_dest);
                }
                if(valor.contieneFinal()){
                    a.getFinales().insertar(st_new_dest);
                }
            }

            //Agregamos los enlaces.
            Enlace enlace_new = new Enlace( st_new_origen, st_new_dest,
                                           clave.getIndiceToken().getValor());

            st_new_origen.addEnlace(enlace_new);
        }

        return a;
    }


    public String imprimir(){
        String print = "";
        Enumeration en = dtrans.keys();
        while(en.hasMoreElements()){
            DtransClave clave = (DtransClave) en.nextElement();
            ListaEstados lista = obtenerValor(clave);

            print += "\n" + clave.getIndiceEstados().imprimir() +
                    " -#- " + clave.getIndiceToken().getValor() +
                    " = " + lista.imprimir();

        }
        return print;
    }

}

/*
 * DtransClave.java
 *
 * Created on 11 de noviembre de 2008, 12:58 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package afgenjava;

import traductor.Token;

/**
 * Clase DtransClave,
 *      Se utiliza como clave de el Dtrans, que es un hashtable.
 *      Consta de:
```

```
 *              1 (una) lista de estados
 *              1 (un) token
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancia ({@link fernandomancia@gmail.com})
 */
public class DtransClave {

    /**
     * Las filas, indicadas por una lista de estados
     */
    private ListaEstados indiceEstados;

    /**
     * La columna, indicada por un token del lenguaje
     */
    private Token indiceToken;




    /** Creates a new instance of DtransClave */
    public DtransClave(ListaEstados list, Token tok) {
        this.indiceEstados = list;
        this.indiceToken = tok;
    }


    /**
     * Getter for property indiceEstados.
     * @return Value of property indiceEstados.
     */
    public ListaEstados getIndiceEstados() {
        return this.indiceEstados;
    }

    /**
     * Setter for property indiceEstados.
     * @param indiceEstados New value of property indiceEstados.
     */
    public void setIndiceEstados(ListaEstados indiceEstados) {
        this.indiceEstados = indiceEstados;
    }


    /**
     * Getter for property indiceToken.
     * @return Value of property indiceToken.
     */
    public Token getIndiceToken() {
        return this.indiceToken;
    }

    /**
     * Setter for property indiceToken.
     * @param indiceToken New value of property indiceToken.
     */
    public void setIndiceToken(Token indiceToken) {
        this.indiceToken = indiceToken;
    }



    /**
     * Compara 2 claves del Dtrans.
     **/
    public int compareTo(Object otro){
        DtransClave o = (DtransClave) otro;
        if(indiceToken.getValor().compareTo(o.getIndiceToken().getValor()) == 0) {
            if(indiceEstados.compareTo(o.getIndiceEstados()) == 0){
                return 0;
            }else{
                return -1;
            }
        }else{
            return -1;
        }
    }
```

```
}
package afgenjava;

/**
 * La clase <b> Enlace </b> representa a los arcos que conectan los estados
 * en una Automata Finito. <br><br>
 *
 * Un enlace está definido por los siguientes componentes:<br><br>
 * <ul>
 *   <li>Estado Origen</li>
 *   <li>Estado Destino</li>
 *   <li>Etiqueta (símbolo del alfabeto)</li>
 * </ul>
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class Enlace implements Comparable<Enlace> {

    /**
     * Apuntador al estado de origen del Enlace
     * Obs.: No es necesario, pero se deja porque podría favorecer a la
     * reutilización de la clase.
     */
    private Estado origen;

    /**
     * Apuntador al estado de destino del Enlace
     */
    private Estado destino;

    /**
     * Simbolo de transición. Su valor no importa si vacio esta como true;
     */
    private String etiqueta;

    /**
     * Propiedad que indica si el enlace representa al símbolo vacío.
     */
    private boolean vacio;

     /**
      * Constructor de la Clase Enlace. Crea un nuevo enlace entre "origen" y
      * "destino" con la etiqueta "label"
      *
      * @param origen  Estado de origen del enlace.
      * @param destino Estado de destino del enlace.
      * @param label   Etiqueta del Enlace
      */
    public Enlace(Estado origen, Estado destino, String label) {
        this.origen = origen;
        this.destino = destino;
        this.etiqueta = label;

        if (label.compareTo(CONSTANS.getVacio())==0) {
            this.vacio = true;
        } else {
            this.vacio = false;
        }
    }

    // ------------------- GETTERS Y SETTERS DE LA CLASE -------------------//

    /**
     * Método para obtener el estado origen del enlace
     * @return El origen del enlace
     */
    public Estado getOrigen() {
        return origen;
    }

    /**
     * Método para establecer el estado origen del enlace
     * @param origen Origen a establecer
     */
    public void setOrigen(Estado origen) {
        this.origen = origen;
    }

    /**
```

```
     * Método para obtener el estado destino del enlace
     * @return El destino del enlace
     */
    public Estado getDestino() {
        return destino;
    }

    /**
     * Método para establecer el estado destino del enlace
     * @param destino Destino a establecer
     */
    public void setDestino(Estado destino) {
        this.destino = destino;
    }

    /**
     * Obtener la etiqueta del enlace
     * @return La etiqueta del enlace.
     */
    public String getEtiqueta() {
        return this.etiqueta;
    }

    /**
     * Establecer la etiqueta del enlace
     * @param label Etiqueta para el enlace
     */
    public void setEtiqueta(String label) {
        this.etiqueta = label;
    }

    public void setVacio(boolean vacio) {
        this.vacio = vacio;
    }

    public boolean isVacio() {
        return vacio;
    }

    /**
     * Implementación del método para comparar enlaces
     *
     * @param e Estado al cual queremos comparar el actual
     * @return <ul> <li><b>0 (Cero)</b> si son  iguales                          </li>
     *              <li><b>-1 (Menos Uno)</b> si son <b>distintos</b> </li>
     *         </ul>
     */
    public int compareTo(Enlace e) {

        Estado origi;
        Estado desti;
        String simbi;


        origi = e.getOrigen();
        desti = e.getDestino();
        simbi = e.getEtiqueta();


        if (origi == this.getOrigen()
                && desti == this.getDestino()
                && simbi.equals(this.getEtiqueta())
                ) {
            return 0;
        } else {
            return -1;
        }
    }

    public String toString(){
        return getEtiqueta();
    }
}
package afgenjava;

import java.util.ArrayList;
import java.util.HashMap;
import traductor.Token;

/**
```

```
 * La clase <b> Estado </b> representa a los nodos dentro de un Autómata
 * finito. <br><br>
 *
 * Un estado está definido por su nombre (identificador) y puede estar conectado
 * a otros estados por medio de símbolos en el alfabeto. Esta clase contiene
 * los dos componentes:<br><br>
 * <ul>
 *   <li>Identificador del Estado</li>
 *   <li>Su conjunto de enlaces asociados</li>
 * </ul>
 *
 * <br>
 * Además, se definine propiedades auxiliares que caracterizan al estado
 * en el automata correspondiente.
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class Estado implements Comparable<Estado> {
    private int id;
    private ListaEnlaces enlaces;

    /*
     * Otras propiedades del Estado que lo definen en el contexto de un
     * autómata
     */
    private boolean estadoinicial;  // establece si el Estado es un estado Inicial
    private boolean estadofinal;    // establece si el Estado es un estado Final
    private boolean visitado;       // establece si el Estado ya fue visitado en el
                                    // contexto de un recorrido por el autómata

    /**
     * Constructor del Estado. Inicializa todas sus características.
     *
     * @param id Identificador del Estado
     * @param esInicial Define si es un estado inicial
     * @param esFinal Define si es un estado final
     * @param visitado Define si ya fue visitado
     */
    public Estado(int id, boolean esInicial, boolean esFinal, boolean visitado) {
        this.id         = id;
        this.estadoinicial  = esInicial;
        this.estadofinal    = esFinal;
        this.visitado   = visitado;
        this.enlaces = new ListaEnlaces();
    }


    // ------------------------------ GETTERS ------------------------------ //

    /**
     * Obtener Id del Estado
     * @return Id del estado
     */
    public int getId() {
        return id;
    }

    /**
     * Obtener lista de enlaces del estado
     * @return ArrayList con los enlaces.
     */
    public ListaEnlaces getEnlaces() {
        return enlaces;
    }

    /**
     * Verifica si el estado es un estado final
     * @return Boolean que define si el estado es un estado final
     */
    public boolean isEstadofinal() {
        return estadofinal;
    }

    /**
     * Verifica si el estado es un estado inicial
     * @return Boolean que define si el estado es un estado inicial
     */
    public boolean isEstadoinicial() {
        return estadoinicial;
```

```
}

/**
 * Verifica si el estado ya fue visitado en un recorrido
 * @return Boolean que define si el estado ya fue visitado en un recorrido
 */
public boolean isVisitado() {
    return visitado;
}


// ------------------------------ SETTERS ------------------------------ //

/**
 * Establece un valor para el identificador del estado
 * @param id Identificador del Estado
 */
public void setId(int id) {
    this.id = id;
}

/**
 * Establece si el estado es Final
 * @param estadofinal Boolean que establece si el estado es o no Final
 */
public void setEstadofinal(boolean estadofinal) {
    this.estadofinal = estadofinal;
}

/**
 * Establece si el estado es inicial
 * @param estadoinicial Boolean que establece si el estado es o no Inicial
 */
public void setEstadoinicial(boolean estadoinicial) {
    this.estadoinicial = estadoinicial;
}

/**
 * Establece si el estado fue o no visitado en un recorrido
 * @param visitado Boolean que establece si el estado fue o no visitado en un recorrido
 */
public void setVisitado(boolean visitado) {
    this.visitado = visitado;
}

// --------------------------- OTROS MÉTODOS --------------------------- //

/**
 * Agrega un nuevo enlace que sale de este estado
 * @param e Enlace a agregar
 */
public void addEnlace(Enlace e) {
    // Insertar en la lista de enlaces para tener un método eficiente de
    // recorrido en el futuro
    enlaces.insertar(e);
}

/**
 * Retorna el estado destino buscando entre todos los enlaces de este estado.
 * @param a Token de la transicion.
 * @return El estado destino al que va desde este estado por el token a
 */
public Estado estadoDestino(Token a){
    return estadoDestinoString(a.getValor());
}

/**
 * Retorna el estado destino buscando entre todos los enlaces de este estado.
 * @param a String que es la etiqueta de la transicion.
 * @return El estado destino al que va desde este estado por el token a
 */
public Estado estadoDestinoString(String a){
    for(Enlace x: enlaces){
        if(x.getEtiqueta().compareTo(a)== 0){
            return x.getDestino();
        }
    }
    return null;
}
```

```
/**
 * Obtiene el primer enlace asociado al simbolo especificado que está
 * cargado en el Hash de enlaces
 * @param simbolo
 * @return
 */
public Estado getDestinoFromHash(String simbolo) {
    Enlace link = this.getEnlaceSimboloFromHash(simbolo);
    Estado result = null;

    if (link != null) {
        result =link.getDestino();
    }
    return result;
}

/**
 * Devuelve el enlace relacionado con el símbolo
 * @param simbolo
 * @return
 */
public Enlace getEnlaceSimboloFromHash(String simbolo) {
    return this.enlaces.getEnlaceSimbolo(simbolo);
}

/**
 * Si el automata es un AFN, devuelve los enlaces vacios asociado a este
 * estado.
 * @return
 */
public ArrayList<Enlace> getEnlacesVacios() {
    return this.enlaces.getVacios();
}


public void eliminarEnlace(Enlace e){
    this.enlaces.borrar(e);

}

public boolean esEstadoMuerto(){
    if(isEstadofinal()){
        return false;
    }

    boolean esMuerto = true;
    for(Enlace e: this.enlaces){
        if(e.getDestino().getId() != this.getId()){
            esMuerto = false;
        }
    }
    return esMuerto;
}



/**
 * Implementación del método para comparar estados
 *
 * @param e Estado al cual queremos comparar el actual
 * @return <ul> <li><b>0 (Cero)</b> si son  iguales                        </li>
 *              <li><b>1 (Uno)</b> si Estado es mayor que <b>e</b>         </li>
 *              <li><b>-1 (Menos Uno)</b> si Estado es menor que <b>e</b> </li>
 *         </ul>
 */
public int compareTo(Estado e) {
    if (this.getId() == e.getId()) {
        return 0;
    } else if (this.getId() > e.getId()) {
        return 1;
    } else {
        return -1;
    }
}

@Override
public String toString() {
    String result = ""+id;
    if (this.isEstadofinal()) {
        result = result + "(fin)";
```

```
        }

        if (this.isEstadoinicial()){
            result = result + "(ini)";
        }
        return result;
    }
}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package afgenjava;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

/**
 * Wrapper de un ArrayList en el que se almacenarán los enlaces que salen de un
 * Estado.
 *
 * Observación:
 * - Evaluar si no sería mejor un HashMap con clave, en el que podríamos utilizar
 *   el símbolo del alfabeto como clave de cada enlace.
 *
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class ListaEnlaces extends ArrayList<Enlace> {

    /* ---------------------- PROPIEDADES DE LA LISTA ---------------------- */

    /**
     * Identificador de la Lista de enlaces
     */
    private int id;

    /**
     * Se implementa una tabla Hash interna para la lista de enlaces que
     * permita indexar para cada símbolo del alfabeto, el índice del array list
     * con el enlace asociado. <br>
     *
     * Por cada nuevo enlace, se tendrá que agregar la Hash la entrada
     * correspondiente. Esta tabla será útil para buscar los enlaces asociados
     * a un símbolo cuando se requiera recorrer el Automata.
     *
     * En esta tabla solo se guardarán los índices de enlaces asociados a
     * símbolso no vacíos.
     *
     */
    private HashMap<String, Integer> TablaEnlaces;

    /**
     * Listado de enlaces cuya etiqueta es el simbolo vacio
     */
    private ArrayList<Enlace> vacios;



    public ListaEnlaces(){
        this.TablaEnlaces = new HashMap<String, Integer>();
        this.vacios       = new ArrayList<Enlace>();
    }


    /* ------------------- GETTERS Y SETTERS DE LA LISTA ------------------- */

    /**
     * Establecer el identificador de listado
     * @param id Identificador del conjunto de estados.
     */
    public void setId(int id) {
        this.id = id;
    }


    /**
```

```
 * Obtener el id del conjunto de estados.
 * @return Identificador del conjunto de estados.
 */
public int getId() {
    return this.id;
}

/**
 * Obtener la lista de enlaces asociados al símbolo vació
 * @return Lista de enlaces del simbolo vacío
 */
public ArrayList<Enlace> getVacios() {
    return vacios;
}


/**
 * Obtener un estado de la lista. Por convención, el index de cada estado
 * será igual a su Id.
 * @param index Indice del arraylist donde está almacenado el estado a obtener.
 * @return El estado almacenado en la posición index.
 */
public Enlace getEnlace(int index) {
    return this.get(index);
}

/**
 * Devuelve un iterador para recorrer el listado de estados.
 * @return Iterador sobre el conjunto de estados.
 */
public Iterator<Enlace> getIterator() {
    return this.iterator();
}


/* --------------------- OTROS MÉTODOS ---------------------- */

/**
 * Insertar un nuevo estado a la lista
 * @param e Estado a insertar.
 */
public void insertar(Enlace e) {

    int     indexToInsert   = this.cantidad();
    String  simbolo         = e.getEtiqueta();

    this.add(e);

    if (e.isVacio()) {
        this.agregarEnlaceVacio(e);
    } else {
        this.TablaEnlaces.put(simbolo, indexToInsert);
    }
}

/**
 * Insertar un nuevo estado a la lista, en la posición indicada
 * @param e Estado a insertar.
 * @param index posicion donde se insertara el elemento
 */
public void insertarAt(Enlace e, int index) {

    int     indexToInsert   = index;
    String  simbolo         = e.getEtiqueta();

    this.add(index,e);

    if (e.isVacio()) {
        this.agregarEnlaceVacio(e);
    } else {
        this.TablaEnlaces.put(simbolo, indexToInsert);
    }
}

public Enlace getEnlaceSimbolo(String symbol) {
    Integer index = this.TablaEnlaces.get(symbol);
    Enlace result = null;

    if (index != null) {
        result = this.get(index);
```

```
    }
    return result;
}

/**
 * Método que permite añadir al final de la lista de enlaces, otra lista de
 * enlaces. Será útil para la implementación de los algoritmos de thompson.
 *
 * @param l
 */
public void insertarListaEnlaces(ListaEnlaces l) {
    Iterator <Enlace> i = l.getIterator();
    Enlace current;

    while(i.hasNext()) {
        current = i.next();
        this.insertar(current);
    }
}

/**
 * Permite insertar un nuevo enlace cuya etiqueta es VACIO en la lista de
 * vacios
 * @param e
 */
private void agregarEnlaceVacio(Enlace e) {
    this.getVacios().add(e);
}

/**
 * Eliminar un estado del conjunto.
 * @param e Estado a eliminar
 */
public void borrar(Enlace e) {

    String simbolo = e.getEtiqueta();

    this.remove(e);

    if (e.isVacio()) {
        this.getVacios().remove(e);
    } else {
        TablaEnlaces.remove(simbolo);
    }
}

/**
 * Obtener la cantidad de estados de la lista
 * @return Número de estados de la lista
 */
public int cantidad() {
    return this.size();
}


/**
 * Método que permite verificar si el estado e pertenece o no
 * a la lista de estados.
 * @param e Estado para el cual queremos verificar la condición de pertenencia
 * @return True o False dependiendo de si el estado pertenece o no
 */
public boolean contiene(Estado e) {
    if (this.contains(e)) {
            return true;
    }
    return false;
}


/**
 * Método heredado reescrito para comparar dos listas de enlaces.
 *
 * Dos listas de estados son iguales si tienen la misma cantidad de elementos
 * y si los mismos son iguales en ambas listas.
 *
 * @param o ListaEstados con el que se comparará la lista actual.
 * @return <ul> <li><b>0 (Cero)</b> si son  iguales                        </li>
 *              <li><b>1 (Uno)</b> si Estado es mayor que <b>e</b>         </li>
 *              <li><b>-1 (Menos Uno)</b> si Estado es menor que <b>e</b> </li>
 *         </ul>.
```

```
     */
    public int compareTo(Object o) {

        int result = -1;

        ListaEnlaces otro = (ListaEnlaces) o;

        // comparación de cantidad de estados
        if (this.cantidad() == otro.cantidad()) {

            // comparación uno a uno
            for (int i = 0; i < this.cantidad(); i++) {

                Enlace a = this.getEnlace(i);
                Enlace b = otro.getEnlace(i);

                if (a.compareTo(b) != 0) {
                    return -1;
                }
            }

            result = 0; //Si llego hasta aqui es xq los elementos son iguales
        }

        return result;
    }

    /**
     * Imprime en una larga cadena toda la lista de estados.
     * @return Un String que contiene la representación en String de
     *         la lista de estados.
     */
    public String imprimir() {

        String result = " ";

        Estado origi;
        Estado desti;
        String simbi;
        Enlace current;


        result = result + this.getId() + " = { ";

        for (int i = 0; i < this.cantidad(); i++) {

            current = this.getEnlace(i);

            origi = current.getOrigen();
            desti = current.getDestino();
            simbi = current.getEtiqueta();

            if (current.isVacio()) {
                simbi = "EMPTY";
            }

            result = result + "("+ origi + "--|"+simbi+"|-->"+desti+ ")";

            if (!(i == (this.cantidad()-1))) {
                result = result + ", ";
            }
        }

        result = result + " } ";

        return result;
    }
}
package afgenjava;

import exceptions.AutomataException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Iterator;

/**
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancia ({@link fernandomancia@gmail.com})
```

```
 */
public
 class ListaEstados extends ArrayList<Estado>{

    /**
     * Identificador de la Lista de estados
     */
    private int id;

    /**
     * Establecer el identificador de listado
     * @param id Identificador del conjunto de estados.
     */
    public void setId(int id) {
        this.id = id;
    }

    /**
     * Obtener el id del conjunto de estados.
     * @return Identificador del conjunto de estados.
     */
    public int getId() {
        return this.id;
    }

    /**
     * Insertar un nuevo estado a la lista
     * @param e Estado a insertar.
     */
    public void insertar(Estado e) {
        this.add(e);
    }

    /**
     * Eliminar un estado del conjunto.
     * @param e Estado a eliminar
     */
    public void borrar(Estado e) {
        this.remove(this.getEstadoById(e.getId()));
    }

    /**
     * Obtener un estado de la lista. Por convención, el index de cada estado
     * será igual a su Id.
     * @param index Indice del arraylist donde está almacenado el estado a obtener.
     * @return El estado almacenado en la posición index.
     */
    public Estado getEstado(int index){
        return this.get(index);
    }

    public Estado getEstadoById(int index) {
        Iterator it = this.getIterator();
        while(it.hasNext()){
            Estado e = (Estado) it.next();
            if(e.getId() == index){
                return e;
            }
        }
        throw new IndexOutOfBoundsException(" No existe en esta lista un Estado con id = " + index);

    }

    /**
     * Obtener la cantidad de estados de la lista
     * @return Número de estados de la lista
     */
    public int cantidad() {
        return this.size();
    }

    /**
     * Devuelve un iterador para recorrer el listado de estados.
     * @return Iterador sobre el conjunto de estados.
     */
    public Iterator <Estado> getIterator() {
        return this.iterator();
    }

    /**
```

```
     * Con este método, se vuelven a marcar todos los estados de la lista
     * como no visitados.
     */
    public void resetVisitas() {
        for (int i = 0; i < cantidad(); i++) {
            getEstado(i).setVisitado(false);
        }
    }

    /**
     * Método que permite verificar si el estado e pertenece o no
     * a la lista de estados.
     * @param e Estado para el cual queremos verificar la condición de pertenencia
     * @return True o False dependiendo de si el estado pertenece o no
     */
    public boolean contiene(Estado e) {
        if (this.contains(e)) {
                return true;
        }
        return false;
    }

    public Estado getEstadoInicial() throws AutomataException{
        int indice_ini = 0;
        int cant_iniciales = 0;
        for (int i = 0; i < cantidad(); i++) {
            if(getEstado(i).isEstadoinicial()){
                indice_ini = i;
                cant_iniciales++;
            }
        }
        if(cant_iniciales == 1){
            return getEstado(indice_ini);
        }else{
            throw new AutomataException("Solo debe haber un estado incial, y en esta lista existen
"+ cant_iniciales);
        }
    }

    public Estado getEstadoFinal() throws AutomataException{
        int indice_fin = 0;
        int cant_finales = 0;
        for (int i = 0; i < cantidad(); i++) {
            if(getEstado(i).isEstadofinal()){
                indice_fin = i;
                cant_finales++;
            }
        }
        if(cant_finales == 1){
            return getEstado(indice_fin);
        }else{
            throw new AutomataException("Este metodo se usa cuando existe un solo " +
                    "estado final y en esta lista existen " + cant_finales +
                    ". Utilize el metodo getEstadosFinales");
        }
    }


    public ListaEstados getEstadosFinales() throws AutomataException{
        ListaEstados nuevaLista = new ListaEstados();
        for (int i = 0; i < cantidad(); i++) {
            if(getEstado(i).isEstadofinal()){
                nuevaLista.insertar(getEstado(i));
            }
        }
        return nuevaLista;
    }

    public boolean contieneInicial(){
        //verificar q contenga un estado inicial
        Estado ini = null;
        try{
            ini = getEstadoInicial();
            return true;
        }catch (Exception ex){
            return false;
        }
    }

    public boolean contieneFinal() {
```

```
    ListaEstados fin;
    try {
        fin = getEstadosFinales();
    } catch (AutomataException ex) {
        return false;
    }

    if(fin.cantidad() > 0){
        return true;
    }else{
        return false;
    }
}


/**
 * Método para ordenar los estados de la lista
 */
public void ordenar() {

    Estado a[] = new Estado[1];

    a = this.toArray(a);
    Comparator<Estado> c = null;

    Arrays.sort(a, c);

    this.removeAll(this);

    for(int i = 0; i < a.length; i++) {
        this.add(a[i]);
    }
}

/**
 * Método heredado reescrito para comparar dos listas de estados.
 *
 * Dos listas de estados son iguales si tienen la misma cantidad de elementos
 * y si los mismos son iguales en ambas listas.
 *
 * @param o ListaEstados con el que se comparará la lista actual.
 * @return <ul> <li><b>0 (Cero)</b> si son  iguales                        </li>
 *              <li><b>1 (Uno)</b> si Estado es mayor que <b>e</b>         </li>
 *              <li><b>-1 (Menos Uno)</b> si Estado es menor que <b>e</b> </li>
 *         </ul>.
 */
public int compareTo(Object o) {

    int result = -1;

    ListaEstados otro = (ListaEstados) o;

    //Se ordenan ambas Listas
    otro.ordenar();
    this.ordenar();

    // comparación de cantidad de estados
    if (this.cantidad() == otro.cantidad()) {

        // comparación uno a uno
        for (int i = 0; i < this.cantidad(); i++) {

            Estado a = this.getEstado(i);
            try{
                otro.getEstadoById(a.getId());
            }catch(Exception ex){
                return -1;
            }
        }

        result = 0; //Si llego hasta aqui es xq los elementos son iguales
    }

    return result;
}

/**
 * Imprime en una larga cadena toda la lista de estados.
 * @return Un String que contiene la representación en String de
 *         la lista de estados.
```

```
     */
    public String imprimir() {

        String result = " ";

        result = result + this.getId() + " = { ";

        for (int i = 0; i < this.cantidad(); i++) {

            result = result + ( this.get(i) ).getId();

            if (!(i == (this.cantidad()-1))) {
                result = result + ", ";
            }
        }

        result = result + " } ";

        return result;
    }

    /**
     * Holds value of property marcado.
     */
    private boolean marcado;

    /**
     * Getter for property marcado.
     * @return Value of property marcado.
     */
    public boolean isMarcado() {
        return this.marcado;
    }

    /**
     * Setter for property marcado.
     * @param marcado New value of property marcado.
     */
    public void setMarcado(boolean marcado) {
        this.marcado = marcado;
    }


}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package afgenjava;

import exceptions.AutomataException;
import traductor.Analizador;

/**
 *
 * @author cparra
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) throws AutomataException{
        String regex = "(a|b)*";
        String alpha = "ab";

        System.out.println("Testing: Analizador.java (testAfGen)");
        System.out.println("--> Generaci?n de un AFN simple con:\n-->   regex (sin
espacios)="+regex+"\n-->   alfabeto="+alpha);

        Analizador t = new Analizador(regex, alpha);
        Automata A = t.traducir();
        A.setAlpha(t.getAlfabeto());
        A.setRegex(t.getRegex());

        String salida_simple = A.imprimir();
        System.out.println(salida_simple);

        //Alg de Subconjuntos
```

```
        AlgSubconjuntos algSub = new AlgSubconjuntos(A);
        Automata AFD = algSub.ejecutar().convertAutomata();
        System.out.println("\nAFD\n____\n");
        System.out.println(AFD.imprimir());

        //Eliminar estados inalacanzables
        AFD = AlgSubconjuntos.eliminar_estados_inalcanzables(AFD);
        System.out.println("\nAFD sin estados inalcanzables\n___________\n");
        System.out.println(AFD.imprimir());

        //Alg de Minimizacion
        AlgMinimizacion algMin = new AlgMinimizacion(AFD);
        Automata AFDM = algMin.minimizar();
        System.out.println("\nAFDM\n_____\n");
        System.out.println(AFDM.imprimir());

        //Eliminar estados muertos
        AFDM.eliminar_estados_muertos();
        System.out.println("\nAFDM sin estados muertos\n_________________\n");
        System.out.println(AFDM.imprimir());

    }

    public static Automata unAutomata() throws AutomataException{
        String regex = "a*b?(ab|ba)*b?a*";
        String alpha = "ab";

        Analizador t = new Analizador(regex, alpha);
        Automata A = t.traducir();
        A.setAlpha(t.getAlfabeto());
        A.setRegex(t.getRegex());
        /**
        //Alg de Subconjuntos
        AlgSubconjuntos algSub = new AlgSubconjuntos(A);

   Automata AFD = algSub.ejecutar().convertAutomata();

        //Eliminar estados inalacanzables
        AFD = AlgSubconjuntos.eliminar_estados_inalcanzables(AFD);

        //Alg de Minimizacion
        AlgMinimizacion algMin = new AlgMinimizacion(AFD);
        Automata AFDM = algMin.minimizar();

        //Eliminar estados muertos
        AFDM.eliminar_estados_muertos();**/
        return A;
    }

}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package afgenjava;

import java.util.ArrayList;
import java.util.Stack;
import traductor.*;

/**
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancia ({@link fernandomancia@gmail.com})
 */
public class Simulacion {

     // variables utilizadas en la simulación y validación de cadenas de entrada
    private String validationString;

    private ArrayList<ListaEstados> estadosPathAFN;

    private ListaEstados estadosPath;

    private int currentIndex = -1;

    private Automata automata;
```

```
// variables auxiliares

Stack<Estado> estadosAnt;           // conjunto actual de estados

Stack<Estado> estadosNew;           // conjunto siguiente de estados

boolean[] yaEstaEn;                 // cuales estados ya están en EstadosNew

ArrayList<Enlace> enlacesVacios;         // lista utilizada para almacenar mover[s,vacio]

String currentCar;


public Simulacion() {
}

public Simulacion(String validationString, Automata automata) {
    this.validationString = validationString;
    this.automata = automata;

    this.estadosPathAFN = new ArrayList<ListaEstados>();
    this.estadosPath = new ListaEstados();
    this.yaEstaEn    = new boolean[automata.getEstados().size()];
    this.estadosAnt  = new Stack<Estado>();
    this.estadosNew  = new Stack<Estado>();

    /* Deprecated
    for (boolean b : this.yaEstaEn) {
        b = false;
    }*/
}

public Estado getEstadoFinal() {
    Estado result = null;

    if (this.automata.getTipo() == TipoAutomata.AFN) {
        // @TODO
    } else {
        if (estadosPath != null) {
            int cantidad = estadosPath.size();
            if (cantidad > 0) {
                result = estadosPath.get(cantidad-1);
            }
        }
    }

    return result;
}

public Estado getEstadoPreFinal() {
    Estado result = null;

    if (this.automata.getTipo() == TipoAutomata.AFN) {
        // @TODO
    } else {
        if (estadosPath != null) {
            int cantidad = estadosPath.size();
            if (cantidad > 1) {
                result = estadosPath.get(cantidad-2);
            }
        }
    }

    return result;
}


/**
 * Proceso que recorre el automata para verificar si la cadena de prueba
 * perteneces al lenguaje descrito por la expresión regular.
 * @param test cadena de prueba cuya pertenencia queremos verificar
 * @return boolean True si la cadena pertenece, false en caso contrario.
 */
public boolean validar() {
    boolean exito = true;

    if (this.automata.getTipo() == TipoAutomata.AFN) {
        exito = this.validar_AFN();
    } else {
        exito = this.validar_AFD();
```

```
    }

    return exito;
}

private void agregarEstado(Estado s) {
    this.estadosNew.push(s);
    this.yaEstaEn[s.getId()] = true;

    this.enlacesVacios = s.getEnlacesVacios(); // equivale a mover[s,(vacio)]

    for (Enlace e : this.enlacesVacios) {
        Estado t = e.getDestino();
        if (!this.yaEstaEn(t)) {
            this.agregarEstado(t);
        }
    }
}

private boolean contieneFinal(ListaEstados S) {
    boolean exito = false;
    for (Estado e : S ) {
        exito = e.isEstadofinal();
        if (exito) {
            break;
        }
    }
    return exito;
}

private Estado validar_AFN_Backtracking(Estado current_state) {

    String current = this.currentCar();
    Estado result = current_state;

    Enlace path = current_state.getEnlaceSimboloFromHash(current);

    // Si no hay ningún enlace al símbolo, buscamos algún vacío.
    // Solo se aplica a los AFNs
    if (path == null && this.automata.getTipo() == TipoAutomata.AFN) {
        ArrayList<Enlace> emptys = current_state.getEnlacesVacios();

        for (Enlace enlace : emptys) {
            Estado siguiente = enlace.getDestino();

            // se inserta el estado a seguir en el camino de validacion
            int indexEstado = this.estadosPath.cantidad();
            this.estadosPath.add(siguiente);
            result = this.validar_AFN_Backtracking(siguiente);

            if (result != null) {
                break;
            }
            this.estadosPath.remove(indexEstado);
        }
    } else {  // se encontró un enlace seguir por el símbolo y avanzamos

        Estado siguiente = path.getDestino();

        this.estadosPath.add(siguiente);
        this.sigCar();

        result = this.validar_AFN_Backtracking(siguiente);
    }

    return result;
}


private boolean validar_AFN() {
    boolean exito = true;

    AlgSubconjuntos subc = new AlgSubconjuntos(this.automata);
    ListaEstados S = new ListaEstados();
    S = subc.e_cerradura(this.automata.getInicial(), S);
    String c = this.sigCar();

    this.estadosPathAFN.add(S);

    while (c.compareToIgnoreCase("")!=0) {
```

```
            S = subc.mover(S, new Token(c));
            S = subc.e_cerradura(S);

            if (S == null || S.size() == 0) {
                exito = false;
                break;
            }

            this.estadosPathAFN.add(S);

            c = this.sigCar();
        }

        if (exito) {
            exito = this.contieneFinal(S);
        }

        return exito;
    }

    private boolean validar_AFD() {
        Estado s = this.automata.getInicial();
        String c = this.sigCar();
        boolean exito = true;

        // empezamos a cargar el camino de la simulación
        this.estadosPath.insertar(s);

        while (c.compareToIgnoreCase("")!=0) {
            s = this.mover(s, c);
            if (s == null) {
                exito = false;
                break;
            }

            this.estadosPath.insertar(s);
            c = this.sigCar();
        }

        if (s != null && !s.isEstadofinal()) {
           exito = false;
        }

        return exito;
    }

    private Estado mover(Estado s, String c) {
        Estado next = s.getDestinoFromHash(c);
        return next;
    }

    private String sigCar() {
        String siguiente = "";
        this.currentIndex++;
        if (this.currentIndex < this.validationString.length()) {
            siguiente = this.validationString.charAt(this.currentIndex)+"";
        } else {
            this.currentIndex = 0;
        }
        return siguiente;
    }

    private String currentCar() {
        String siguiente = this.validationString.charAt(this.currentIndex)+"";
        return siguiente;
    }


    public String getValidationString() {
        return validationString;
    }

    public void setValidationString(String validationString) {
        this.validationString = validationString;
        this.currentIndex = 0;
        this.estadosPath = new ListaEstados();
    }

    public ArrayList<ListaEstados> getEstadosPathAFN() {
        return estadosPathAFN;
```

```
    }

    public ListaEstados getEstadosPath() {
        return estadosPath;
    }

    public int getCurrentIndex() {
        return currentIndex;
    }

    public

    Automata getAutomata() {
        return automata;
    }

    public void setAutomata(Automata automata) {
        this.automata = automata;
    }

    private boolean yaEstaEn(Estado t) {
        return this.yaEstaEn[t.getId()];
    }

    public String getSimulationPath() {
        return this.estadosPath.toString();
    }

}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package afgenjava;

/**
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 */
public enum TipoAutomata {
    AFN,
    AFD,
    AFDMin
}
```

3. Paquete exceptions

```
/*
 * AutomataException.java
 *
 * Created on 8 de noviembre de 2008, 05:51 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package exceptions;

/**
 *
 * @author Administrador
 */
public class AutomataException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>AutomataException</code> without detail message.
     */
    public AutomataException() {
    }


    /**
     * Constructs an instance of <code>AutomataException</code> with the specified detail message.
     * @param msg the detail message.
     */
    public AutomataException(String msg) {
        super(msg);
    }
}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package exceptions;

/**
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 */
public class LexicalError extends Exception {

    /**
     * Creates a new instance of <code>LexicalError</code> without detail message.
     */
    public LexicalError() {
    }


    /**
     * Constructs an instance of <code>LexicalError</code> with the specified detail message.
     * @param msg the detail message.
     */
    public LexicalError(String msg) {
        super(msg);
    }
}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package exceptions;

/**
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 */
public class SyntaxError extends Exception {

    /**
     * Creates a new instance of <code>SyntaxError</code> without detail message.
     */
```

```
    public SyntaxError() {
    }


    /**
     * Constructs an instance of <code>SyntaxError</code> with the specified detail message.
     * @param msg the detail message.
     */
    public SyntaxError(String msg) {
        super(msg);
    }


    /**
     * Construye una instancia de <code>SyntaxError</code> con el mensage
     * detallada, anticipado por la información de la pos en la expresión
     * regular donde se produce el error
     *
     *
     * @param msg el mensaje detallado
     * @param pos la posicion en la cadena de entrada donde se produjo la excepción
     */
    public SyntaxError(String msg,int pos) {
        super("Error de sintaxis en el símbolo ["+pos+"]: " +msg);
    }
}
```

4. Paquete app

```
/*
 * About.java
 *
 * Created on 18 de noviembre de 2008, 10:03 PM
 */

package app;

/**
 *
 * @author  Cristhian Parra ({@link cdparra@gmail.com})
 */
public class About extends javax.swing.JFrame {

    /** Creates new form About */
    public About() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jLabel2 = new javax.swing.JLabel();
        jScrollPane2 = new javax.swing.JScrollPane();
        jTextPane2 = new javax.swing.JTextPane();
        okBtn = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jPanel1.setBorder(javax.swing.BorderFactory.createEtchedBorder());

        jLabel1.setFont(new java.awt.Font("Alien Encounters", 1, 36));
        jLabel1.setForeground(new java.awt.Color(0, 0, 102));
        jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel1.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/img/logo_afgen.png"))); // NOI18N

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(21, 21, 21)
                .addComponent(jLabel1)
                .addContainerGap(23, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );

        jLabel2.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/img/knockfamily.jpg"))); // NOI18N

        jTextPane2.setContentType("text/html");
        jTextPane2.setEditable(false);
        jTextPane2.setText("<html>\n  <head>\n\n  </head>\n  <body>\n    <p style=\"margin-top:
0\">\n<strong>AfGen</strong> - Finite Automaton Generator for Regular Expressions is developed
by:\n<ul>\n<LI>Cristhian D. Parra T. </LI>\n<LI>Fernando M. Mancia Z.</LI>\n</ul>\n<strong>Version:
</strong>1.0<br>\n    </p>\n  </body>\n</html>\n");
        jScrollPane2.setViewportView(jTextPane2);
```

```
        okBtn.setText("Close");
        okBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                okBtnActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                    .addGroup(jPanel2Layout.createSequentialGroup()
                        .addContainerGap()
                        .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.TRAILING, false)
                            .addComponent(jScrollPane2, javax.swing.GroupLayout.Alignment.LEADING,
0, 0, Short.MAX_VALUE)
                            .addComponent(jLabel2, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                    .addGroup(jPanel2Layout.createSequentialGroup()
                        .addGap(197, 197, 197)
                        .addComponent(okBtn)))
                .addContainerGap(15, Short.MAX_VALUE))
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel2)
                .addGap(27, 27, 27)
                .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 12,
Short.MAX_VALUE)
                .addComponent(okBtn)
                .addContainerGap())
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
                    .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

private void okBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_okBtnActionPerformed
this.setVisible(false);
}//GEN-LAST:event_okBtnActionPerformed

    /**
    * @param args the command line arguments
    *
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
```

```
            public void run() {
                new About().setVisible(true);
            }
        });
    }*/

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextPane jTextPane2;
    private javax.swing.JButton okBtn;
    // End of variables declaration//GEN-END:variables

}
/*
 * AutomataGrafico.java
 *
 * Created on 15 de noviembre de 2008, 02:47 PM
 */

package app;

import afgenjava.*;
import graphviz.GraphViz;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Font;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.geom.RoundRectangle2D;
import java.io.File;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;
import java.util.Stack;
import javax.swing.BorderFactory;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;


import org.jgraph.JGraph;
import org.jgraph.graph.DefaultEdge;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.DefaultGraphModel;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.GraphModel;


/**
 *
 * @author  Cristhian Parra ({@link cdparra@gmail.com})
 */
public class AutomataGrafico extends javax.swing.JFrame {


    private Automata automata;
    private AutomataGraph jgraph;
    DefaultGraphCell[] cells;
    private String library = "graphviz";
    private JLabel imageLabel;
    private String imgDir = "/tmp";
    private Simulacion simulacion;
    private boolean simulacionResult;
    private String imgUrl;
    private String graphvizbin="/usr/bin/dot";
    private boolean enSimulacion = false;
    private ListaEstados camino;
    private Estado EstadoActual;
    private int IndexActual;
    private boolean simulacionTerminada = false;
    private String simulationMessage = "";
    private String validationString="";
    private String CaracterActual="";
```

```
    private Estado EstadoSiguiente;
    private ArrayList<File> fileList;
    private File imagenOriginal;
    private boolean CargarOriginal=true;
    private boolean HayArchivos = false;
    private boolean primeraVez = true;

    /** Creates new form AutomataGrafico */
    public AutomataGrafico(String library, Automata a, Configuracion conf) {
        /*
         * Definir si usamos mxGraph o JGraph
         */
        this.automata = a;
        this.imageLabel = new javax.swing.JLabel();
        this.imageLabel.setFont(new java.awt.Font("Dialog", 1, 18)); // NOI18N
        this.imageLabel.setForeground(new java.awt.Color(100,
100, 255));
        this.imageLabel.setText("<Aquí va la Imagen>");
        this.library = library;

        this.imgDir = conf.getImgdir();
        this.graphvizbin = conf.getDotPath()+File.separator + "dot";
        // Construct Model and Graph
        DefaultGraphModel model = new DefaultGraphModel();
        this.jgraph = new AutomataGraph(a,model);

        initComponents();

        this.verificarSimulacion();
        this.cargarAutomata();
    }

    void setSimulacionResult(boolean simResult) {
        this.simulacionResult = simResult;
    }

    /**
     * Borra los archivos que están en la cache, incluyendo el original si así lo
     * indica el parámetro
     *
     * @param incluirOri
     */
    private void borrarArchivos(boolean incluirOri) {
        if (this.fileList != null) {
            for (File f : this.fileList) {
                if (f.exists()) {
                    f.delete();
                }
            }
        }

        this.fileList = null;

        if(incluirOri) {
            if (this.imagenOriginal != null && this.imagenOriginal.exists()) {
                this.imagenOriginal.delete();
            }
        }

        this.HayArchivos = false;
    }

    private void cargarAutomata() {
        if (this.library.compareTo("jgraph") == 0) {
            // Do nothing
        } else {
            this.cargarAutomataGraphViz();
        }
    }

    private void cargarAutomataGraphViz() {

        this.imgUrl = this.generateImageUrl();
        GraphViz gv = new GraphViz();

        if (gv.testGraphViz()) {
            if (this.imgUrl.compareTo("<NO>") == 0) {
                this.imageLabel.setText("<El Path al Directorio de imagen es incorrecto>");
                this.imageLabel.setFont(new Font("Verdana", Font.BOLD, 14));
                this.imageLabel.setForeground(Color.orange);
```

```
            } else {

                boolean dibujar = this.primeraVez;     // variable que indica que debemos generar el
dibujo

                ImageIcon i = null;

                if (dibujar) { // Usar las imágenes guardadas
                    i = this.dibujarNuevo();
                    this.imagenOriginal = new File (this.imgUrl);
                    this.primeraVez = false;
                } else {
                    if (this.fileList != null && ((this.fileList.size() - 1) < this.IndexActual)) {
                        dibujar = true;
                        i = this.dibujarNuevo();
                        File f = new File(this.imgUrl);
                        this.fileList.add(f);
                        this.HayArchivos = true;
                    } else if (this.fileList != null) {
                        dibujar = false; // no generar, utilizar uno ya creado.

                        File f = this.fileList.get(this.IndexActual);
                        i = new ImageIcon(f.getAbsolutePath());
                    }
                }

                if (i != null) {
                    i.setDescription("Automata Generado");
                    this.imageLabel.setIcon(i);
                }
            }
        } else {
            this.imageLabel.setText("<GraphViz no está instalado>");
            this.imageLabel.setFont(new Font("Verdana", Font.BOLD, 14));
            this.imageLabel.setForeground(Color.red);
        }
        this.imageLabel.setHorizontalAlignment(JLabel.CENTER);
        this.imageLabel.setVisible(true);
    }

    private Component ScrollPaneConstructor() {
        if (this.getLibrary().compareTo("jgraph") == 0) {
            return this.jgraph;
        } else if (this.getLibrary().compareTo("graphviz") == 0){
            return this.imageLabel;
        }

        return this.imageLabel;
    }

    private void cargarMensajeSimulacion(String msg, Color c) {
        if (this.simulacionTerminada) {
            this.simulationMessage = "La Cadena ";
            this.simulationMessage += this.simulacionResult ? "SI " : "NO ";
            this.simulationMessage += "pertenece al Lenguaje.##" + msg;
        } else {
            this.simulationMessage = msg;
        }

        Color finalColor = c;
        if(this.simulacionTerminada) {
            finalColor = this.simulacionResult?Color.blue:Color.red;
        }
        Font font = new Font("VERDANA", Font.BOLD, 12);

        this.jTextFieldOutput.setText(this.simulationMessage);
        this.jTextFieldOutput.setForeground(finalColor);
        this.jTextFieldOutput.setFont(font);
    }

    private ImageIcon dibujarNuevo() {
        GraphViz gv = new GraphViz();
        gv.dibujar(this.getDotSyntax(), this.imgUrl);
        this.imageLabel.setText("");
        return (new ImageIcon(this.imgUrl));
    }

    /**
     * Genera un URL para la imagen a utilizar en los gráficos.
     * Se utiliza un URL aleatorio debido a que existe un problema con la
```

```
     * cache de los ImageIcon que hace que no se actualize la imagen si el
     * nombre no cambio.
     * @return Strin nombre de la imagen a cargar en el panel de dibujo
     */
    private String generateImageUrl() {
        Random r = new Random();
        r.nextInt(100000);
        String rand = ""+r.nextInt(100000);

        String dir = this.getImgDir();

        File fileDir = new File(dir);
        String dibujo = "<NO>";

        System.out.println("ImageDir: "+fileDir);
        if (fileDir.isDirectory()) {
            // Crear la Imagen
            dibujo = this.getImgDir()+File.separator+"automata_"+rand+".gif";
        }

        return dibujo;
    }

    /**
     * Construye la cadena de atributos visuales para un estado en particular
     * @param e El estado que vamos a dibujar
     * @param marcado Si es true, significa que está marcado y tiene un color especial
     * @return
     */
    private String getColorEstado(Estado e, boolean marcado) {

        String style = "[";
        // Características gráficas de cada estado
        String shape = e.isEstadofinal()?"shape=doublecircle":"shape=circle";

        style += shape;

        /**
         * Estilos Especiales.
         *
         * Definen los estilos para estados finales e iniciales. Si el nodo está
         * marcado, define los atributos de un nodo marcado.
         */
        String coloresp     = marcado?"color=green4":"color=blue4";
        String fillcolor    = marcado?"style=filled,fillcolor=green":"style=filled,fillcolor=blue";
        String fontcolor    = marcado?"fontcolor=white":"fontcolor=white";
        String label        = e.isEstadoinicial()?",label=inicio":"";

        if(e.isEstadofinal() || e.isEstadoinicial() || marcado) {
            style += ","+fillcolor+","+coloresp+","+fontcolor+label;
        }

        return style+"];";
    }

    /**
     * Construye la cadena de atributos visuales para un enlace en particular
     * @param enalce El enlace que vamos a dibujar
     * @param lbl Label del enlace
     * @param marcado Si es true, significa que está marcado y tiene un color especial
     * @return
     */
    private String getEnlaceStyle(Enlace enlace, String lbl, boolean marcado) {

        String style = "[";
        // Características gráficas de cada estado
        String label = "label=\""+lbl+"\"";

        style += label;

        /**
         * Estilos Especiales.
         *
         * Definen los estilos para enlaces marcados en una simulacion
         */
        String coloresp     = marcado?",color=green4":"";

        style += coloresp+"];";

        return style;
```

```
    }

    private void habilitarSimulacion() {
        jButtonNext.setEnabled(true);
        jButtonPrev.setEnabled(true);
        jButtonSimular.setEnabled(true);
        jTextValidation.setEnabled(true);
    }

    private void bloquearSimulacion() {
        jButtonNext.setEnabled(false);
        jButtonPrev.setEnabled(false);
        jButtonSimular.setEnabled(false);
        jTextValidation.setEnabled(false);
    }

    /**
     * Funciones dedicadas al control de la simulación
     */

    /**
     * Método que inicia la simulación de la validación. Se encarga de inicializar
     * el entorno de simulación.
     *
     */
    private void iniciarSimulacion() {

        this.jTextFieldOutput.setText("");
        this.enSimulacion = true;                       // bandera que indica que estamos en medio
de una simulación
        this.simulacionTerminada = false;               // bandera que indica que la simulación ha
terminado


        // Se inicia la simulación:
        // 1. Crear un objeto simulación y validar la cadena de entrada para construir
        //    el camino de la simulación.
        this.setSimulacion(new Simulacion(this.jTextValidation.getText(), automata));
        this.simulacionResult = this.getSimulacion().validar();
        Color c = Color.blue;   // color por defecto para estados y enlaces de simulación actuales


        if (this.HayArchivos) {
            this.borrarArchivos(false); // borrar archivos de simulaciones previas sin borrar
original
        }

        this.fileList = new ArrayList<File>();  // creamos una nueva lista de archivos

        // Si se introdujo una cadena de prueba no vacía, simulamos, sino solo imprimimos
        // el mensaje de validación.
        if (this.jTextValidation.getText().compareTo("") != 0) {

            this.jButtonNext.setEnabled(true);  // habilitamos el botón siguiente

            // Solo en para la simulación con Graphviz se habilita el retroceder del simulador
            if (this.library.compareTo("graphviz") == 0) {

        this.jButtonPrev.setEnabled(true);
            }


            if (this.library.compareTo("jgraph") == 0) {
                jgraph.empezarSimulacion(simulacion.getEstadosPath());
                this.enSimulacion = true;
                cargarMensajeSimulacion("Estados = {", Color.blue);
                return;
            }

            // Mensaje inicial de simulación
            String msg = "La simulación ha comenzado! Utilice 'Next' y 'Prev' para avanzar y
retroceder...";

            this.cargarMensajeSimulacion(msg, c);

            // 2. Inicializamos variables de estado actual para la simulación
            this.camino = this.simulacion.getEstadosPath(); // camino completo de estados recorridos
para simular validación

            this.EstadoActual = this.camino.getEstado(0);   // estado actual de la simulación se
```

```
coloca al comienzo del camino

            if (IndexActual < (this.camino.size() - 1)) {
                this.EstadoSiguiente = this.camino.getEstado(1);// estado siguiente de la simulación
se coloca al comienzo del camino + 1
            }

            this.IndexActual = 0;                           // indice del estado actual de
simulación en el camino

            this.validationString = this.jTextValidation.   // actualizamos la variable de la cadena
e prueba
                    getText();
            this.CaracterActual = this.validationString.    // obtenemos el primer caracter de la
prueba
                    charAt(this.IndexActual) + "";

            this.cargarAutomata();                          // cargamos el nuevo dibujo generado,
con los estados y enlaces pintados

        } else {
            // Si el texto está vacío, damos por terminada la simulación y presentamos el resultado
de la validación
            this.simulacionTerminada = true;
            c = this.simulacionResult ? Color.green : Color.red;
            this.cargarMensajeSimulacion("", c);
        }

        this.jButtonSimular.setText("Reiniciar");
        this.jButtonSimular.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/img/16x16/view-refresh.png"))); // NOI18N
    }

    /**
     * Mueve el entorno de simulación al siguiente estado
     */
    private void nextStep() {

        if (this.library.compareTo("jgraph")==0){
            this.jgraph.nextSimulacion();
            Estado actual = this.jgraph.getEstadoActual();
            if(actual != null){
                String msg = this.jTextFieldOutput.getText()+
                                " - " + actual.getId();
                cargarMensajeSimulacion(msg, Color.blue);
            }else{
                if(this.enSimulacion){
                    this.enSimulacion = false;
                    this.simulacionTerminada = true;
                    String msg = this.jTextFieldOutput.getText() +"}";
                    cargarMensajeSimulacion(msg, Color.blue);
                }
            }
            return;
        }

        Color c = Color.blue;
        // Si se introdujo una cadena de prueba no vacía, simulamos, sino solo imprimimos
        // el mensaje de validación.
        if (this.enSimulacion) {

            String msg = ""; // Mensaje de simulación
            int cantidadEstados = this.camino.size();

            if (this.IndexActual < (cantidadEstados - 1)) {
                this.IndexActual++;
                this.EstadoActual = this.camino.getEstado(this.IndexActual);   // estado actual de
la simulación se coloca al comienzo del camino

                if (this.IndexActual != (cantidadEstados-1)) {
                    this.EstadoSiguiente = this.camino.getEstado(this.IndexActual + 1);
                    this.CaracterActual = this.validationString. // obtenemos el primer caracter de
la prueba
                            charAt(this.IndexActual) + "";
                } else {
                    this.EstadoSiguiente = null;
                    this.CaracterActual = "";
                }
            } else {
                this.EstadoActual = this.camino.getEstado(this.IndexActual);   // estado actual de
```

```
la simulación se coloca al comienzo del camino
                this.EstadoSiguiente = null;
                this.simulacionTerminada = true;               // Llegamos al último estado
                this.CaracterActual = "";
            }

            String actual = "Actual: -"+this.EstadoActual.getId() + "- | ";
            String simbolo = "Simbolo: -"+this.CaracterActual+"- ";
            String siguiente = (this.EstadoSiguiente == null)?"":"| Siguiente:
-"+this.EstadoSiguiente.getId()+"-";

            msg = actual + simbolo + siguiente;
            this.cargarMensajeSimulacion(msg, c);

            this.cargarAutomata();                            // cargamos el nuevo dibujo generado,
con los estados y enlaces pintados
            if(this.simulacionTerminada) {
                jButtonNext.setEnabled(false);
            } else {
                jButtonNext.setEnabled(true);
            }

        } else {
            // Si el texto está vacío, damos por terminada la simulación y presentamos el resultado
de la validación
            this.simulacionTerminada = true;
            c = this.simulacionResult ? Color.green : Color.red;
            this.cargarMensajeSimulacion("", c);
        }
    }

    /**
     * Mueve el entorno de simulación al paso anterior
     */
    private void prevStep() {


        Color c = Color.blue;
        // Si se introdujo una cadena de prueba no vacía, simulamos, sino solo imprimimos
        // el mensaje de validación.
        if (this.enSimulacion) {

            String msg = ""; // Mensaje de simulación
            int cantidadEstados = this.camino.size();
            this.simulacionTerminada = false;

            if (this.IndexActual > 0) {
                this.IndexActual--;
                this.EstadoActual = this.camino.getEstado(this.IndexActual);   // estado actual de
la simulación se coloca al comienzo del camino

            } else {
                this.EstadoActual = this.camino.getEstado(this.IndexActual);   // estado actual de
la simulación se coloca al comienzo del camino
                this.EstadoSiguiente = this.camino.getEstado(this.IndexActual + 1);
                this.simulacionTerminada = true;               // Llegamos al último estado
            }

            if (this.IndexActual < (cantidadEstados - 1)) {
                this.EstadoSiguiente = this.camino.getEstado(this.IndexActual + 1);
            }

            this.CaracterActual = this.validationString. // obtenemos el primer caracter de la
prueba
                    charAt(this.IndexActual) + "";

            String actual = "Actual: -"+this.EstadoActual.getId() + "- | ";
            String simbolo = "Simbolo: -"+this.CaracterActual+"- ";
            String siguiente = (this.EstadoSiguiente == null)?"":"| Siguiente:
-"+this.EstadoSiguiente.getId()+"-";

            msg = actual + simbolo + siguiente;
            this.cargarMensajeSimulacion(msg, c);
            this.cargarAutomata();                            // cargamos el nuevo dibujo generado,
con los estados y enlaces pintados
            if(this.IndexActual == 0) {
                jButtonPrev.setEnabled(false);
            } else {
                jButtonPrev.setEnabled(true);
            }
```

```
            this.jButtonNext.setEnabled(true);

        } else {
            // Si el texto está vacío, damos por terminada la simulación y presentamos el resultado
de la validación
            this.simulacionTerminada = true;
            c = this.simulacionResult ? Color.green : Color.red;
            this.cargarMensajeSimulacion("", c);
        }
    }

    /**
     * Verifica si se puede habilitar la simulación.
     * La versión actual no soporta la simulación en el AFN.
     *
     * Si el automata a simular no es AFN, habilita los controles de simulación,
     * sino deshabilita los controles.
     */
    private void verificarSimulacion() {
        if (this.automata.getTipo() != TipoAutomata.AFN) {
            this.habilitarSimulacion();
            this.jButtonNext.setEnabled(false);
            this.jButtonPrev.setEnabled(false);
        } else {
            this.bloquearSimulacion();
        }
    }

    /**
     * Construye la sintaxis adecuada para generar el gráfico por medio de la
     * aplicación "dot" del toolkit de GraphViz. <br> <br>
     *
     * De acuerdo a ciertos criterios del entorno de simulación, establece los
     * colores y otras características del grafo. <br><br>
     *
     * El estado inicial y los finales también tienen un formato especial <br><br>
     *
     * La sintaxis de GraphViz (El lenguaje DOT) se define aquí
     * <href="http://www.graphviz.org/doc/info/lang.html">DOT Language</href>
     *
     * @return String Cadena completa formateada del automata en versión grapviz
     */
    public String getDotSyntax(){

        String result_header =
                "Digraph AFN {\n" +
                "\trankdir=LR;\n\toverlap=scale;\n";

        String result_nodes = "\tnode [shape = circle];\n";
        String result_edges = "";


        ListaEstados estados = this.automata.getEstados();

        for (Estado e : estados) {
            boolean mark = false;

            if (this.enSimulacion) {
                mark = (e.getId() == this.EstadoActual.getId());

                if (!mark && this.EstadoSiguiente != null) {
                    mark = (e.getId() == this.EstadoSiguiente.getId());
                }
            }

            String EstadoStyle = this.getColorEstado(e,mark);


            result_nodes+="\t"+e.getId() + " "+EstadoStyle+"\n";

            for (Enlace enlace : e.getEnlaces()) {

                Estado orig = enlace.getOrigen();
                Estado dest = enlace.getDestino();
                String label = enlace.getEtiqueta();

                mark = ((label.compareTo(this.CaracterActual)==0) && (orig.getId() ==
this.EstadoActual.getId()));

                String EnlaceStyle = this.getEnlaceStyle(enlace,label,mark);
```

```
                result_edges += "\t"+orig.getId() + " -> " + dest.getId() +
                                " "+EnlaceStyle+"\n";

            }
        }
        String result = result_header + result_nodes + result_edges + "}";
        return result;
    }


    /**
     * GETTERS Y SETTERS DE ATRIBUTOS DE LA CLASE
     */

    public void setAutomata(Automata automata) {
        this.automata = automata;
    }

    public void setJLabelTituloText(String label) {
        this.jLabelTitulo.setText(label);
    }

    public void setJTextReGexString(String jTextReGex) {
        this.jTextReGex.setText(jTextReGex);
    }

    public void setJTextAlphaString(String jTextAlpha) {
        this.jTextAlpha.setText(jTextAlpha);
    }

    public void setJTextValidation(String jText) {
        this.jTextValidation.setText(jText);
    }

    public String getLibrary() {
        return library;
    }

    public void setLibrary(String library) {
        this.library = library;
    }

    public String getImgDir() {
        return imgDir;
    }

    public void setImgDir(String imgDir) {
        this.imgDir = imgDir;
    }

    public Simulacion getSimulacion() {
        return simulacion;
    }

    public void setSimulacion(Simulacion simulacion) {
        this.simulacion = simulacion;
    }

    public String getGraphvizPath() {
        return graphvizbin;
    }

    public void setGraphvizPath(String graphvizPath) {
        this.graphvizbin = graphvizPath;
    }


/**
 *
 * TODO EL CÓDIGO GENERADO DE LA CONFIGURACIÓN VISUAL
 *
 */
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
```

```
    private void initComponents() {

        jLabelTitulo = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jTextReGex = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        jTextAlpha = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        jTextValidation = new javax.swing.JTextField();
        jButtonClose = new javax.swing.JButton();
        jButtonSimular = new javax.swing.JButton();
        jButtonNext = new javax.swing.JButton();
        jScrollPane1 = new JScrollPane(this.ScrollPaneConstructor());
        ;
        jButtonPrev = new javax.swing.JButton();
        jTextFieldOutput = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("AfGen - Gráfico del Automata");

        jLabelTitulo.setFont(new java.awt.Font("Dialog", 1, 18));
        jLabelTitulo.setForeground(new java.awt.Color(100, 100, 255));
        jLabelTitulo.setText("Gráfico del Automata");

        jLabel2.setText("Expresión Regular:");

        jTextReGex.setBackground(new java.awt.Color(255, 255, 153));
        jTextReGex.setEditable(false);

        jLabel4.setText("Cadena de Prueba:");

        jTextAlpha.setBackground(new java.awt.Color(204, 255, 153));
        jTextAlpha.setEditable(false);

        jLabel3.setText("Alfabeto:");

        jTextValidation.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyReleased(java.awt.event.KeyEvent evt) {
                jTextValidationKeyReleased(evt);
            }
        });

        jButtonClose.setText("Close");
        jButtonClose.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonCloseActionPerformed(evt);
            }
        });

        jButtonSimular.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/16x16/media-
playback-start.png"))); // NOI18N
        jButtonSimular.setText("Simular");
        jButtonSimular.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonSimularActionPerformed(evt);
            }
        });

        jButtonNext.setText("Next");
        jButtonNext.setEnabled(false);
        jButtonNext.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonNextActionPerformed(evt);
            }
        });

        jScrollPane1.setBorder(javax.swing.BorderFactory.createTitledBorder("Graph"));

        jButtonPrev.setText("Prev");
        jButtonPrev.setEnabled(false);
        jButtonPrev.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonPrevActionPerformed(evt);
            }
        });

        jTextFieldOutput.setBackground(new java.awt.Color(204, 255, 204));

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
```

```
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabelTitulo)
                        .addGap(216, 216, 216))
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addComponent(jLabel2)
                            .addComponent(jLabel4)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jButtonClose)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATE
D)
                                .addComponent(jLabel1)))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                                .addComponent(jButtonSimular)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jButtonPrev)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jButtonNext))
                            .addComponent(jTextValidation, javax.swing.GroupLayout.DEFAULT_SIZE,
581, Short.MAX_VALUE)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jTextReGex, javax.swing.GroupLayout.PREFERRED_SIZE,
288, javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jLabel3)
                                .addGap(1, 1, 1)
                                .addComponent(jTextAlpha, javax.swing.GroupLayout.DEFAULT_SIZE, 243,
Short.MAX_VALUE)))
                        .addContainerGap())
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAIL
ING)
                            .addComponent(jTextFieldOutput,
javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE, 678,
Short.MAX_VALUE)
                            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 678,
Short.MAX_VALUE))
                        .addContainerGap())))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabelTitulo)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(jTextReGex, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel3)
                    .addComponent(jTextAlpha, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(4, 4, 4)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel4)
                    .addComponent(jTextValidation, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))

   .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jButtonClose)
                    .addComponent(jButtonNext)
                    .addComponent(jButtonPrev)
                    .addComponent(jButtonSimular)
                    .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 351,
```

```
Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 18,
Short.MAX_VALUE)
                .addComponent(jTextFieldOutput, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

/**
 * Handle de la acción que inicia el evento de simulación.
 * @param evt
 */
private void jButtonSimularActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButtonSimularActionPerformed
    this.iniciarSimulacion();
}//GEN-LAST:event_jButtonSimularActionPerformed

private void jButtonNextActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButtonNextActionPerformed
   // siguiente Paso de la simulación
    this.nextStep();
}//GEN-LAST:event_jButtonNextActionPerformed

private void jButtonCloseActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButtonCloseActionPerformed
    this.setVisible(false);

   /* File imageIcon = new File(this.imgUrl);
    if (this.library.compareTo("graphviz")==0 && imageIcon.exists()) {
        if (imageIcon.exists()) {
            imageIcon.delete();
        }
    }    */

    this.borrarArchivos(true);
    this.dispose();
}//GEN-LAST:event_jButtonCloseActionPerformed

private void jTextValidationKeyReleased(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_jTextValidationKeyReleased
    this.jButtonSimular.setText("Simular");
    this.jButtonSimular.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/16x16/media-
playback-start.png"))); // NOI18N
    this.fileList = new ArrayList<File>();
}//GEN-LAST:event_jTextValidationKeyReleased

private void jButtonPrevActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButtonPrevActionPerformed
    this.prevStep();
}//GEN-LAST:event_jButtonPrevActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonClose;
    private javax.swing.JButton jButtonNext;
    private javax.swing.JButton jButtonPrev;
    private javax.swing.JButton jButtonSimular;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabelTitulo;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField jTextAlpha;
    private javax.swing.JTextField jTextFieldOutput;
    private javax.swing.JTextField jTextReGex;
    private javax.swing.JTextField jTextValidation;
    // End of variables declaration//GEN-END:variables

}/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package app;

import afgenjava.Automata;
import afgenjava.Enlace;
```

```
import afgenjava.Estado;

import afgenjava.ListaEnlaces;
import afgenjava.ListaEstados;
import com.jgraph.layout.JGraphFacade;
import com.jgraph.layout.JGraphLayout;
import com.jgraph.layout.graph.JGraphSimpleLayout;
import com.jgraph.layout.hierarchical.JGraphHierarchicalLayout;
import com.jgraph.layout.organic.JGraphFastOrganicLayout;
import com.jgraph.layout.organic.JGraphOrganicLayout;
import exceptions.AutomataException;
import java.awt.Color;
import java.awt.Component;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.BorderFactory;
import java.util.ArrayList;

import java.util.Enumeration;
import java.util.Hashtable;
import java.util.List;
import java.util.Map;
import javax.swing.JFrame;
import org.jgraph.JGraph;
import org.jgraph.graph.*;

/**
 * Representacion del Grafo usando la definicion jgraph.
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class AutomataGraph extends JGraph{

    // Lista de cells que se agregan al jgraph
    private ArrayList<DefaultGraphCell> cells;

    int width = 40, height = 40;

    private Automata automata;

    private int indiceEstadoActual = 0;


    private ListaEstados estSimular;

    private Estado estadoActual;

    public AutomataGraph(Automata a, DefaultGraphModel model){
        super(model);
        this.automata = a;

        //Imprimimos en consola para verificar.
        System.out.println("El automata que se dibujara es\n_____________________\n");
        System.out.println(a.imprimir());


        //Convertimos y dibujamos con jgraph.
        convertirJgraph();

        aplicar_layout_organico();
    }



/**
 * METODO PRINCIPAL PARA CONVERTIR EL AUTOMATA(nodos, arcos) A LOS
 * ELEMENTOS DE UN JGRAPH
 */
    private void convertirJgraph(){

        //Borramos todo lo que habia en cells y creamos uno nuevo
        cells = new ArrayList<DefaultGraphCell>();

        for(Estado elEstado : automata.getEstados()){
            incluirEnlacesEstado(elEstado);
        }
```

```
        Object[] elementosObj = cells.toArray();
        //DefaultGraphCell[] elementos = (DefaultGraphCell[]) cells.toArray();

        // Control-drag should clone selection
        setCloneable(true);

        // Enable edit without final RETURN keystroke
        setInvokesStopCellEditing(true);

        // When over a cell, jump to its default port (we only have one, anyway)
        setJumpToDefaultPort(true);


        // Insert the cells via the cache, so they get selected
        getGraphLayoutCache().insert(elementosObj);

    }




/**
 * FUNCIONES PARA APLICAR UN LAYOUT EN PARTICULAR
 */
    private void aplicar_layout_circular(){
        // Pass the facade the JGraph instance
        JGraphFacade facade = new JGraphFacade(this);

        // Create aninstance of the circle layout
        JGraphLayout layout = new JGraphSimpleLayout(JGraphSimpleLayout.TYPE_CIRCLE);

        layout.run(facade); // Run the layout on the facade.
        Map nested = facade.createNestedMap(true, true); // Obtain a map of the resulting attribute
changes from the facade
        getGraphLayoutCache().edit(nested); // Apply the results to the actual graph
    }

    private void aplicar_layout_jerarquico(){
        JGraphFacade facade = new JGraphFacade(this);
        // Pass the facade the JGraph instance

        JGraphLayout layout = new JGraphHierarchicalLayout(true);//phOrganicLayout();
        // Create an instance of the appropriate layout

        layout.run(facade);
        // Run the layout on the facade. Note that layouts do not implement the Runnable interface,
to avoid confusion

        Map nested = facade.createNestedMap(true, true);
        // Obtain a map of the resulting
        this.getGraphLayoutCache().edit(nested);
    }


    private void aplicar_layout_organico(){
        JGraphFacade facade = new JGraphFacade(this);
        facade.setDirected(true);

        JGraphOrganicLayout layout = new JGraphOrganicLayout();
        layout.setOptimizeEdgeDistance(true);
        layout.setEdgeCrossingCostFactor(500000);
        layout.setOptimizeEdgeDistance(true);
        layout.setEdgeDistanceCostFactor(5000);

        layout.run(facade);
        Map nested = facade.createNestedMap(true, true);
        getGraphLayoutCache().edit(nested);

    }


/**
 * FUNCIONES AUXILIARES PARA CONVERTIR DE UN AUTOMATA A UN JGRAPH
 */
    private void incluirEnlacesEstado(Estado estado){

        // Crear un "cell" para el Estado
        DefaultGraphCell origen = createCell(estado, width * automata.getEstados().cantidad()/2,
250);
        double x = 0;
```

```
        double y;
        for (Enlace link : estado.getEnlaces()) {
                    if (estado.getEnlaces().indexOf(link)  % 2 == 0) {
                        y = 50;
                    } else {
                        y = 450;
                    }

                    DefaultGraphCell destino = createCell( link.getDestino(), x, y);
                    DefaultGraphCell currentLink = createEdge(link, origen, destino);
                    x = x + width;
                }
    }

    private DefaultGraphCell createCell(Estado estado, double x, double y) {
        DefaultGraphCell cell  = obtenerEstado(estado);
        if(cell == null){
            cell = new DefaultGraphCell(estado);
            GraphConstants.setBounds(cell.getAttributes(), new Rectangle2D.Double(x, y, width,
height));
            GraphConstants.setBorder(cell.getAttributes(), BorderFactory.createRaisedBevelBorder());
            GraphConstants.setOpaque(cell.getAttributes(), true);
            GraphConstants.setGradientColor(cell.getAttributes(), Color.LIGHT_GRAY);
            cell.addPort(new Point2D.Double(0, 0));

            //Agregamos al la lista
            cells.add(cell);
        }
        return cell;
    }

    private DefaultGraphCell createEdge(Enlace enlace, DefaultGraphCell source, DefaultGraphCell
target) {
        DefaultEdge edge = new DefaultEdge(enlace);
        source.addPort();
        edge.setSource(source.getChildAt(source.getChildCount()
 -1));
        target.addPort();
        edge.setTarget(target.getChildAt(target.getChildCount() -1));
        GraphConstants.setLabelAlongEdge(edge.getAttributes(), true);
        GraphConstants.setLineEnd(edge.getAttributes(), GraphConstants.ARROW_CLASSIC);
        cells.add(edge);
        return edge;
    }




/**
 * FUNCIONES PARA ACCEDER A LOS ELEMENTOS
 */

    private DefaultGraphCell obtenerEstado(Estado estado){

        for(DefaultGraphCell oneCell: cells){
              if(oneCell.getUserObject() instanceof Estado && oneCell !=null){
                    if(((Estado)oneCell.getUserObject()).getId() == estado.getId()){
                        return oneCell;
                    }
              }
        }
        return null;
    }



/**
 * FUNCIONES PARA MANIPULAR LOS ELEMENTOS
 */

    public void empezarSimulacion(ListaEstados estSimular){
        this.estSimular = estSimular;
        this.indiceEstadoActual = 0;
        this.estadoActual = null;
    }

    public void nextSimulacion(){
        if(indiceEstadoActual >= 0 && indiceEstadoActual < estSimular.cantidad()){
            estadoActual  = estSimular.get(indiceEstadoActual);
            pintarEstado(estadoActual);
```

```
            indiceEstadoActual++;
        }else{
            estadoActual = null;
        }
    }

    public void previewSimulacion(){
        indiceEstadoActual--;
        if(indiceEstadoActual >= 0 && indiceEstadoActual < estSimular.cantidad()){
            estadoActual = estSimular.get(indiceEstadoActual);
            pintarEstado(estadoActual);
        }else{
            estadoActual = null;
        }
    }

    public void pintarEstado(Estado e){

        clearSelection();
        final DefaultGraphCell nodo = obtenerEstado(e);
        GraphConstants.setGradientColor(nodo.getAttributes(), Color.BLACK);
        GraphConstants.setBorder(nodo.getAttributes(), BorderFactory.createLineBorder(Color.BLACK));
        GraphConstants.setBackground(nodo.getAttributes(), Color.BLACK);
        nodo.setUserObject(e);

        System.out.println("Se trato de pintar "+e.toString());
    }




/**
 * GETTER's y SETTER's
 *
 */
    public ListaEstados getEstSimular() {
        return estSimular;
    }

    public void setEstSimular(ListaEstados estSimular) {
        this.estSimular = estSimular;
    }

    public int getIndiceEstadoActual() {
        return indiceEstadoActual;
    }

    public void setIndiceEstadoActual(int indiceEstadoActual) {
        this.indiceEstadoActual = indiceEstadoActual;
    }


    public Estado getEstadoActual() {
        return estadoActual;
    }

    public void setEstadoActual(Estado estadoActual) {
        this.estadoActual = estadoActual;
    }


    public static void main(String args[]) {
        try {
            Automata a = afgenjava.Main.unAutomata();
            DefaultGraphModel model = new DefaultGraphModel();
            AutomataGraph ag = new AutomataGraph(a, model);

            JFrame f = new JFrame();
            f.setDefaultCloseOperation(f.EXIT_ON_CLOSE);
            f.getContentPane().add(ag);
            f.pack();
            f.setVisible(true);
        } catch (AutomataException ex) {
            Logger.getLogger(AutomataGraph.class.getName()).log(Level.SEVERE, null, ex);
        }
    }


}
package app;
```

```
import afgenjava.*;
import java.util.HashMap;
import java.util.Iterator;
import javax.swing.table.AbstractTableModel;

/**
 * Tabla de Transiciones de un automata ajustado para ser utilizado como el
 * modelo de un componente Jtable
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class AutomataTable extends AbstractTableModel {

    /**
     * Automata cuya tabla de tranciciones se quiere representar
     */
    private Automata automata;

    /**
     * Cantidad de columnas de la Tabla
     */
    private int columnCount = 0;

    /**
     * Cantidad de filas de la tabla
     */
    private int rowCount = 0;

    /**
     * Matriz que contiene los data a desplegar en el JTable.
     */
    private Object [][] data;

    /**
     * Etiqueta de cada columna de la tabla
     */
    private String [] columnsName;

    /**
     * Cuenta de simbolos del alfabeto que realmente se utilizan
     */
    private int columnRealCount;

    public AutomataTable(Automata automata) {
        this.automata       = automata;

        if (automata != null) {
            this.columnCount = this.automata.getAlpha().size()+1; // La primera columna es de las
etiquetas de estado

            // Si es AFN, se debe tener entre los elementos del alfabeto al vacio
            if (this.automata.getTipo() == TipoAutomata.AFN) {
                this.columnCount++;
                this.columnRealCount++;
            }

            this.rowCount = this.automata.getEstados().size();
            this.columnsName = new String[this.columnCount];
            this.data = new Object[this.rowCount][this.columnCount];
            this.columnRealCount = 0;

            // Si es AFN, se debe tener entre los elementos del alfabeto al vacio
            if (this.automata.getTipo() == TipoAutomata.AFN) {
                this.columnsName[1] = CONSTANS.getVacio();
                this.columnsName[0] = "Estado";
            }

            this.loadTable();
        }
    }

    /**
     * Constructor principal de la clase
     * @param col Cantidad de columnas de la tabla
     * @param fil Cantidad de filas de la tabla
     */
    public AutomataTable(int fil, int col) {
        this.rowCount       = fil;
```

```
    this.columnCount    = col;
    this.columnsName    = new String[col];
    this.data           = new Object[fil][col];
}

/**
 * Obtener la cantidad de columnas de la tabla
 * @return Cantidad de columnas de la tabla
 */
public int getColumnCount() {
    return this.columnCount;
}

/**
 * Obtener la cantidad de filas de la tabla.
 * @return Cantidad de filas de la tabla
 */
public int getRowCount() {
    return this.rowCount;
}

/**
 * Obtener el nombre de una de las columnas de la tabla
 * @param col Número de columna cuyo nombre desea obtenerse.
 * @return El nombre de la columna.
 */
@Override
public String getColumnName(int col) {
    return this.columnsName[col];
}

/**
 * Establecer el nombre de una columna de la Tabla
 * @param col Número de columna de la Tabla cuyo nombre desea establecerse.
 * @param nombre Nombre de la columna.
 */
public void setColumnName(int col, String nombre) {
    this.columnsName[col] = nombre;
}

/**
 * Obtener un valor almacenado en la Tabla.
 * @param row Número de fila de la Tabla.
 * @param col Número de columna de la Tabla.
 * @return Objeto almacenado en las posiciones [row,col]
 */
public Object getValueAt(int row, int col) {
    return this.data[row][col];
}

/**
 * Establecer un valor en la Tabla.
 * @param value Valor a almacenar en la Tabla en la posici□n [row,col].
 * @param row Número de columna en la Tabla.
 * @param col Número de columna en la Tabla.
 */
@Override
public void setValueAt(Object value, int row, int col) {
    this.data[row][col] = value;
    this.fireTableCellUpdated(row, col);
}

/**
 * Determinar el renderizador por defecto para cada celda.
 * @param c Número de columna cuyo tipo de Clase se quiere conocer.
 * @return Class de la columna en cuesti□n.
 */
@Override
public Class getColumnClass(int c) {
    return this.getValueAt(0, c).getClass();
}

/**
 * Arreglar las posiciones de la Tabla donde no se estableció ningún valor
 * (tiene objetos null). Los objetos null se reemplazan por Strings Vacíos
 */
public void arreglarObjetosNulos() {
    String vacio = " ";
    for (int i = 0; i < this.rowCount; i++) {
        for (int j = 0; j < this.columnCount; j++) {
```

```
                Object o = this.data[i][j];
                if (o == null) {
                    this.setValueAt(vacio, i, j);
                }
            }
        }
    }

    public

    /**
     * Automata cuya tabla de tranciciones se quiere representar
     */
    Automata getAutomata() {
        return automata;
    }

    public void setAutomata(Automata automata) {
        this.automata = automata;
    }

    private void loadTable() {

        // Recorremos el automata estado a estado y en cada paso, cargamos la
        // tabla en el índice que corresponde a la columna y fila del par
        // etiqueta, estado procesado
        for (Iterator<Estado> it = this.automata.getEstados().getIterator(); it.hasNext();) {

            Estado current = it.next();                     // Obtenemos el estadoa actual a
procesar
            ListaEnlaces enlaces = current.getEnlaces();    // Obtenemos sus enlaces
            int rowEstado = current.getId();                // La fila del estado es igual a su id

            String estadoLabel = rowEstado+"";
            if (current.isEstadoinicial()) {
                estadoLabel+="(ini)";
            }

            if (current.isEstadofinal()) {
                estadoLabel+="(fin)";
            }

            this.setValueAt(estadoLabel,rowEstado, 0);

            // Iteramos sobre los enlaces para agregar los destinos en las celdas
            // adecuadas de la matriz
            for (Iterator<Enlace> ite = enlaces.getIterator(); ite.hasNext();) {

                Enlace currentLink = ite.next();            // enlace actual a procesar

                String symbol = currentLink.getEtiqueta();  // simbolo del enlace
                int indexCol = this.findColumn(symbol);     // obtenemos la columna de la etiqueta

                // Si la columna obtenida es -1, todavía no se cargó
                // esta etiqueta al encabezado
                if (indexCol < 0) {
                   indexCol = this.columnRealCount+1;
                   this.columnsName[indexCol] = symbol;
                   this.columnRealCount++;

                }

                Estado destino = currentLink.getDestino();  // obtenemos el destino de esta enlace

                // vemos si para para este estado,simbolo ya habían destinos asociados
                Object estados = this.getValueAt(rowEstado, indexCol);

                if (estados == null) {
                    estados = new ListaEstados();
                }

                ((ListaEstados) estados).add(destino);                      // agregamos el nuevo
destino a la lista

                // Cargamos la lista de nuevo en la matriz de objetos
                this.setValueAt(estados,rowEstado, indexCol);
            }
        }
    }
```

```
    @Override
    public String toString() {
        String result="";

        for (int i = 0; i < data.length; i++) {
            Object[] objects = data[i];
            for (int j = 0; j < objects.length; j++) {
                Object object = objects[j];
                if (object != null) {
                    result = result + ( (ListaEstados) object).toString()+"\t";
                } else {
                    result = result + "null\t";
                }
            }
            result += "\n";
        }

        return result;
    }
}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package app;

import org.simpleframework.xml.*;

/**
 * Clase que implementa el modelo de la configuración del sistema
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
@Root
public class Configuracion {

    /**
     * Path al directorio que contiene el binario ejecutable de Dot (Graphviz)
     */
   @Element
   private String dotpath = "/usr/bin/dot";

   /**
    * Símbolo a ser desplegado como etiqueta de los enlaces vacíos
    */
   @Element
   private String emptySymbol="(vacio)";

   /**
    * Path al directorio que almacenará temporalmente las imágenes que se generan
    */
   @Element
   private String imgdir = "/tmp";

   /**
    * Índice de propiedades de configuración (pensado para implementar versiones
    * de configuración en el futuro)
    */
   @Attribute
   private int index;


   public Configuracion() {
      super();
   }

   public Configuracion(String text, int index) {
      this.index = index;
      this.dotpath = text;
   }

   public String getDotPath() {
      return dotpath;
   }

   public int getGraphViz() {
      return index;
   }
```

```
    public String getEmptySymbol() {
        return emptySymbol;
    }

    public void setEmptySymbol(String emptySymbol) {
        this.emptySymbol = emptySymbol;
    }

    public void setImgdir(String text) {
        this.imgdir = text;
    }

    public String getImgdir() {
        return this.imgdir;
    }

    void setDotPath(String absolutePath) {
        this.dotpath = absolutePath;
    }
}
/*
 * Main.java
 *
 * Created on 10 de noviembre de 2008, 10:50 PM
 */

package app;

import afgenjava.*;
import exceptions.*;
import graphviz.GraphViz;
import java.awt.Color;
import java.awt.Component;
import java.awt.Font;
import java.io.File;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFrame;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableModel;
import javax.xml.crypto.dsig.spec.C14NMethodParameterSpec;
import org.simpleframework.xml.Serializer;
import org.simpleframework.xml.load.Persister;
import traductor.*;

/**
 *
 * Ventana principal de la aplicación de generación de Automatas Finitos para
 * expresiones regulares. <br><br>
 *
 * @author  Cristhian Parra ({@link cdparra@gmail.com})
 * @author  Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class Main extends javax.swing.JFrame {


    /**
     * Variables utilizadas
     */
    /**
     * Configuración de la aplicación utilizada en todo el programa
     */
    private Configuracion conf;
    /**
     * Automata finito no determinista de la expresión procesada
     */
    private Automata AFN;
    /**
     * Automata finito determinista de la expresión procesada
     */
    private Automata AFD;

    /**
     * Automata finito determinista mínimo de la expresión procesada
     */
    private Automata AFDMin;
```

```
/**
 * Tabla de Transiciones de cada automata
 */
private Dtrans DT_AFN;
private Dtrans DT_AFD;
private Dtrans DT_AFDMin;

/**
 * Clases de simulación de automatas para cada tipo
 */
private Simulacion afnSim;
private Simulacion afdSim;
private Simulacion afdMinSim;

/**
 * Banderas de indicación del resultado de la validación
 */
private boolean afnSimResult;
private boolean afdSimResult;
private boolean afdMinSimResult;

/**
 * Ventana de about
 */
private JFrame about;

/**
 * Ventana de configuración
 */
private JFrame config;

/**
 * Ventana de dibujo y simulación
 */
private AutomataGrafico graphics;

/**
* Implementación de procedimientos que hacen la lógica del Programa completo
*/

/**
 * Método que verifica el campo de ingreso de la expresión regular,
 * y habilita los controles relacionados si este texto no està vacío.
 */
public void checkRegEx() {
        String regex = jTextReGex.getText();
        if (regex.compareTo("") != 0) {
            this.habilitarRegExProcess();
        } else {
            this.bloquearRegExProcess();
        }
}

/**
 * Llamada al proceso de validación de una cadena de entrada haciendo uso
 * del AFN
 * @return boolean: True si la cadena pertenece al lenguaje definido por la expresión <br>
 * False si no pertenece al lenguaje
 */
public boolean validarAFN() {
    this.setAfnSim(new Simulacion(this.jTextValidate.getText(), this.AFN));
    this.jTabbedPaneTables.setSelectedIndex(0);
    return this.getAfnSim().validar();
}

/**
 * Llamada al proceso de validación de una cadena de entrada haciendo uso
 * del AFD
 *
 * @return boolean: True si la cadena pertenece al lenguaje definido por la expresión <br>
 * False si no pertenece al lenguaje
 */
public boolean validarAFD() {
    this.setAfdSim(new Simulacion(this.jTextValidate.getText(), this.AFD));
    boolean exito = this.getAfdSim().validar();
    this.cambiarColorCeldaFinal(this.AFD,exito);
    return exito;
}

/**
```

```
 * Llamada al proceso de validación de una cadena de entrada haciendo uso
 * del AFDMin
 *
 * @return boolean: True si la cadena pertenece al lenguaje definido por la expresión <br>
 * False si no pertenece al lenguaje
 */
public boolean validarAFDMin() {
    this.setAfdMinSim(new Simulacion(this.jTextValidate.getText(), this.AFDMin));
    boolean exito = this.getAfdMinSim().validar();
    this.cambiarColorCeldaFinal(this.AFDMin,exito);
    return exito;
}

/**
 * Copia el Alfabeto seleccionado al jTextField correspondiente
 */
public void copyDefaultAlpha() {
    String defaultAlpha = (String) this.jListDefaultAlphas.getSelectedValue();

    String az = "abcdefghijklmnñopqrstuvwxyz";
    String numbers = "0123456789";
    String binario = "01";
    String vocales = "aeiou";
    String all = az + az.toUpperCase() + numbers;

    if (defaultAlpha.compareTo("[a-z]") == 0) {
        this.jTextAlpha.setText(az);
    } else if (defaultAlpha.compareTo("[A-Z]") == 0) {
        this.jTextAlpha.setText(az.toUpperCase());
    } else if (defaultAlpha.compareTo("[a-zA-Z]") == 0) {
        this.jTextAlpha.setText(az + az.toUpperCase());
    } else if (defaultAlpha.compareTo("[0-9]") == 0) {
        this.jTextAlpha.setText(numbers);
    } else if (defaultAlpha.compareTo("[0-1]") == 0) {
        this.jTextAlpha.setText(binario);
    } else if (defaultAlpha.compareTo("[vocals]") == 0) {
        this.jTextAlpha.setText(vocales);
    } else if (defaultAlpha.compareTo("[ALL]") == 0) {
        this.jTextAlpha.setText(all);
    }
}

/**
 *
 * Método principal que se encarga de procesar la entrada para generar los
 * Automatas deseados.
 *
 */
public void procesarRegEx() {


    String regex = jTextReGex.getText();
    String alpha = jTextAlpha.getText();
    boolean Errors = false;

    // Check entries
    if (regex.compareTo("") == 0) {
        jTextAreaOutput.append("# --> No se introdujo ninguna expresión regular\n");
        jTextAreaOutput.append("# <-------------------------------------------\n");
        Errors = true;
    } else if (alpha.compareTo("") == 0) {
        jTextAreaOutput.append("# --> No se introdujo ningún alfabeto\n");
        jTextAreaOutput.append("# <-------------------------------------------\n");
        Errors = true;

    } else {

        this.bloquearControles();

        // Procesar la expresión regular y Generar el AFN, el AFD y el AFDMínimos
        jTextAreaOutput.append("# --> Generando el AFN...\n");
        Analizador traductor = new Analizador(regex, alpha);

        Errors = traductor.isHayErrores();
        // 1. Generar el AFN
        if (!Errors) {
            this.setAFN(traductor.traducir());
        }

        Errors = traductor.isHayErrores();
```

```
        // 1.2. Verificar si hubieron errores.
        if (Errors) {
            jTextAreaOutput.append("# ERRORS: --> " + traductor.getErrMsg() + "\n");
            jTextAreaOutput.append("# <----------------------------------------\n");
        } else {

            jTextAreaOutput.append("# --> AFN Generado con éxito!\n");

            // 2. Generar el AFD
            jTextAreaOutput.append("# --> Generando el AFD...\n");
            AlgSubconjuntos algSub;
            Dtrans dtran;

            try {
                algSub = new AlgSubconjuntos(this.AFN);
                dtran = algSub.ejecutar();
                this.AFD = dtran.convertAutomata();

                this.AFD = AlgSubconjuntos.eliminar_estados_inalcanzables(this.AFD);

                this.AFD.setAlpha(this.AFN.getAlpha());
                this.AFD.setRegex(this.jTextReGex.getText());
                this.AFD.setTipo(TipoAutomata.AFD);


            } catch (AutomataException ex) {
                jTextAreaOutput.append("# ERRORS: --> " + ex.getMessage() + "\n");
                jTextAreaOutput.append("# <----------------------------------------\n");
                Errors = true;
            } catch (Exception ex) {
                jTextAreaOutput.append("# ERRORS: --> " + ex.getMessage() + "\n");
                jTextAreaOutput.append("# <----------------------------------------\n");
                Errors = true;
            }

            if (!Errors) {
                try {
                    jTextAreaOutput.append("# --> AFD Generado con éxito!\n");

                    // 3. Generar el AFDMínimo
                    jTextAreaOutput.append("# --> Generando el AFD Mínimo...\n");

                    AlgMinimizacion minimize = new AlgMinimizacion(this.AFD);
                    this.AFDMin = minimize.minimizar();
                    this.AFDMin.eliminar_estados_muertos();

                    this.AFDMin.setAlpha(this.AFN.getAlpha());
                    this.AFDMin.setRegex(this.jTextReGex.getText());
                    this.AFDMin.setTipo(TipoAutomata.AFDMin);

                    // 4. Poblar las tablas de la ventana principal
                    this.cargarTabla(jTableAFN, this.AFN);
                    this.cargarTabla(jTableAFD, this.AFD);
                    this.cargarTabla(jTableAFDMin, this.AFDMin);
                } catch (AutomataException ex) {
                    jTextAreaOutput.append("# ERRORS: --> " + ex.getMessage() + "\n");
                    jTextAreaOutput.append("# <----------------------------------------\n");
                    Errors = true;
                } catch (Exception ex) {
                    jTextAreaOutput.append("# ERRORS: --> " + ex.getMessage() + "\n");
                    jTextAreaOutput.append("# <----------------------------------------\n");
                    Errors = true;
                }
            }
        }

        this.habilitarControles();

        if (Errors) {
            this.bloquearValidacion();
            this.bloquearVistas();
        }
    }
}

/**
 * Método que carga la tabla correspondiente con las transiciones del
 * automata
 * @param Tabla compoente de tipo Jtable que se cargará
```

```
     * @param automata origen de los datos.
     */
    public void cargarTabla(JTable Tabla, Automata automata) {
        AutomataTable tmodel = new AutomataTable(automata);
        tmodel.arreglarObjetosNulos();
        Tabla.setModel(tmodel);
        this.resetTablaRenderer(Tabla);
    }

    /**
     * Método que se encarga de construir la ventana de dibujo y simulación
     * @param automata
     */
    private void viewGraphics(Automata automata) {

        int toolSelected = jComboBoxGraph.getSelectedIndex();

        jTextAreaOutput.append("# --> Construyendo Imagen del Automata...\n");

        if (toolSelected == 0) {
            jTextAreaOutput.append("# --> Debe seleccionar la herramienta para graficar\n");

        } else if (toolSelected == 1) {
            this.graphics = new AutomataGrafico("graphviz",automata,this.conf);
            this.graphics.setJTextAlphaString(jTextAlpha.getText());
            this.graphics.setJTextReGexString(jTextReGex.getText());
            this.graphics.setJTextValidation(jTextValidate.getText());
            this.cargarSimulacion(automata);
            this.graphics.setVisible(true);
            this.graphics.toFront();
        } else {
            this.graphics = new AutomataGrafico("jgraph",automata,this.conf);
            this.graphics.setAutomata(automata);
            this.graphics.setJTextAlphaString(jTextAlpha.getText());
            this.graphics.setJTextReGexString(jTextReGex.getText());
            this.graphics.setJTextValidation(jTextValidate.getText());
            this.cargarSimulacion(automata);
            this.graphics.setVisible(true);
            this.graphics.toFront();
        }
    }

    /**
     *
    * @param args the command line arguments
    */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Main().setVisible(true);
            }
        });
    }

    /**
     * Funciones auxiliares
     */

    /** Creates new form Main */
    public Main() {
        initComponents();
        this.readConf();
    }

    private void bloquearControles() {
        this.bloquearValidacion();
        this.bloquearRegEx();
        this.bloquearRegExProcess();
        this.bloquearVistas();
        this.bloqearAlpha();
        this.bloquearDefaultAlpha();
    }

    private void cambiarColorCeldaFinal(Automata a, boolean exito) {
        JTable tableACambiar = null;        // Tabla del AFN usado para la validación
        Color incorrectoBack = Color.red;   // Color de fondo que le ponemos a las celdas pintadas
si la validación fue incorrecta
        Color correctoBack   = Color.green; // Color de fondo que le ponemos a las celdas pintadas
si la validación fue correcta
        Color textColor      = Color.white; // Color del Texto en celdas seleccionadas
```

```
        Color preFinalBack   = Color.blue;
        Color finalBack      = exito ? correctoBack : incorrectoBack;
        AutomataTable model  = null;        // Modelo de la tabla a actualizar
        Estado estadoFinal   = null;        // Estado final al que llegó el proceso de validación
        Estado estadoPreFin  = null;        // Estado anterior al estado final en el proceso de
validación
        int estadoFinalCol   = 0;           // Columna del estado final
        int estadoFinalFil   = 0;           // Fila del estado final
        int estadoPreFinCol  = 0;           // Columna del estado anterior final
        int estadoPreFinFil  = 0;           // Fila del estado anterior final
        Component Celda      = null;         // Componente que representa a la Celda de la tabla a
modificar
        Object textValue     = null;
        int indexTable       = 0;

        String valText       = this.jTextValidate.getText();
        int valTextLength    = valText.length();

        String lastChar      = "";
        if (valTextLength > 0) {
            lastChar      = valText.charAt(valTextLength-1) + "";
        }

        if (a.getTipo() == TipoAutomata.AFN) {
            estadoFinal   = this.afnSim.getEstadoFinal();
            estadoPreFin  = this.afnSim.getEstadoPreFinal();
            tableACambiar = this.jTableAFN;
            indexTable    = 0;
        } else if (a.getTipo() == TipoAutomata.AFD) {
            estadoFinal   = this.afdSim.getEstadoFinal();
            estadoPreFin  = this.afdSim.getEstadoPreFinal();
            tableACambiar = this.jTableAFD;
            indexTable    = 1;
        } else if (a.getTipo() == TipoAutomata.AFDMin) {
            estadoFinal   = this.afdMinSim.getEstadoFinal();
            estadoPreFin  = this.afdMinSim.getEstadoPreFinal();
            tableACambiar = this.jTableAFDMin;
            indexTable    = 2;
        }

        model = (AutomataTable) tableACambiar.getModel();

        if (estadoFinal != null) {
            estadoFinalFil = estadoFinal.getId();
        }
        estadoFinalCol = 0;

        if (estadoPreFin != null) {
            estadoPreFinFil = estadoPreFin.getId();
        }

        if (this.jTextAlpha.getText().indexOf(lastChar) < 0) {
            estadoPreFinCol = 0;
        } else {
            estadoPreFinCol = model.findColumn(lastChar);
        }

        if (estadoPreFinCol < 0) {
            estadoPreFinCol = 0;
        }
        // Seleccionar la fila del estado final
        tableACambiar.setRowSelectionInterval(estadoFinalFil, estadoFinalFil);

        // Pintar los estados notables
        OneCellRenderer cr2 = new OneCellRenderer(estadoPreFinFil, estadoPreFinCol,
 preFinalBack, textColor);
        tableACambiar.getColumnModel().getColumn(estadoPreFinCol).setCellRenderer(cr2);

        OneCellRenderer cr = new OneCellRenderer(estadoFinalFil, estadoFinalCol, finalBack,
textColor);
        tableACambiar.getColumnModel().getColumn(estadoFinalCol).setCellRenderer(cr);

        tableACambiar.setCellSelectionEnabled(true);
        this.jTabbedPaneTables.setSelectedIndex(indexTable);
    }

    private void cargarSimulacion(Automata automata) {
        Simulacion sim = null;
        boolean simResult = false;
```

```
        if (automata.getTipo() == TipoAutomata.AFN) {
            if (afnSim != null) {
                sim = afnSim;
                simResult = afnSimResult;
            }
        } else if (automata.getTipo() == TipoAutomata.AFD) {
            if (afdSim != null) {
                sim = afdSim;
                simResult = afdSimResult;
            }
        } else {
            if (afdMinSim != null) {
                sim = afdMinSim;
                simResult = afdMinSimResult;
            }
        }

        if (this.graphics != null) {
            this.graphics.setSimulacion(sim);
            this.graphics.setSimulacionResult(simResult);
        }
    }

    private void readConf() {
        try {
            Serializer serializer = new Persister();
            File source = new File("conf.xml");

            if (source.exists()) {
                this.setConf(serializer.read(Configuracion.class, source));
            } else {
                // Si la configuración no existe, creamos una nueva

                String dotP = "/usr/bin";
                String imgD = "/tmp";

                // si estamos en windows, cambiamos los directorios por defecto
                if (File.separator.compareTo("\\")==0) {

                    dotP = "C:\\Archivos de programa\\Graphviz 2.21\\bin\\d";
                    imgD = "C:\\";
                }

                this.setConf(new Configuracion(dotP, 1));
                this.getConf().setEmptySymbol("(vacio)");
                this.getConf().setImgdir(imgD);

                serializer.write(this.conf, source);
            }

            CONSTANS.setVacio(this.getConf().getEmptySymbol());
            GraphViz.setDot(this.getConf().getDotPath() + File.separator + "dot");
            GraphViz.setDir(this.getConf().getImgdir());


        } catch (Exception ex) {
            this.jTextAreaOutput.append("ERROR: Hubo algún problema con la configuración" +
ex.getMessage() + "\n");
            this.jTextAreaOutput.append("# <-------------------------------------------\n");
        }

    }

    private void resetTablaRenderer(JTable Tabla) {
        Tabla.setBackground(Color.white);
        Tabla.setForeground(Color.black);

        DefaultTableCellRenderer dt = (DefaultTableCellRenderer)
Tabla.getDefaultRenderer(String.class);
        dt.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);
        dt.setBackground(Color.white);
        dt.setForeground(Color.black);

        OneColumnRenderer cr = new OneColumnRenderer(0, Color.gray, Color.white);
        cr.setFont(new Font("Verdana",Font.BOLD, 12));
        Tabla.getColumnModel().getColumn(0).setCellRenderer(cr);
    }

    private void changeValidationTextResult(boolean SimResult) {
```

```
    Color incorrecto = Color.red;
    Color correcto   = Color.green;

    if (SimResult) {
        this.jTextValidateResult.setText("La cadena pertenece al Lenguaje");
        this.jTextValidateResult.setForeground(correcto);
    } else {
        this.jTextValidateResult.setText("La cadena NO pertenece al Lenguaje");
        this.jTextValidateResult.setForeground(incorrecto);
    }
}

private void habilitarControles() {
    this.habilitarValidacion();
    this.habilitarRegEx();
    this.habilitarRegExProcess();
    this.habilitarVistas();
    this.habilitarAlpha();
    this.habilitarDefaultAlpha();
}

private void habilitarRegEx() {
    this.jTextReGex.setEnabled(true);
}

private void bloquearRegEx() {
    this.jTextReGex.setEnabled(false);
}

private void habilitarRegExProcess() {
    this.processBtn.setEnabled(true);
    this.jMenuItemProcesarReGex.setEnabled(true);
}

private void bloquearRegExProcess() {
    this.processBtn.setEnabled(false);
    this.jMenuItemProcesarReGex.setEnabled(false);
}

private void habilitarAlpha() {
    this.jTextAlpha.setEnabled(true);
}

private void bloqearAlpha() {
    this.jTextAlpha.setEnabled(false);
}

private void habilitarVistas() {
    this.viewAFNbtn.setEnabled(true);
    this.viewAFDMinbtn.setEnabled(true);
    this.viewAFDbtn.setEnabled(true);
    this.jComboBoxGraph.setEnabled(true);
}

private void bloquearVistas() {
    this.viewAFNbtn.setEnabled(false);
    this.viewAFDMinbtn.setEnabled(false);
    this.viewAFDbtn.setEnabled(false);
    this.jComboBoxGraph.setEnabled(false);
}

private void habilitarValidacion() {
    this.jTextValidate.setEnabled(true);
    this.validateBtn.setEnabled(true);
    this.jComboBoxValidation.setEnabled(true);
}

private void bloquearValidacion() {
    this.jTextValidate.setEnabled(false);
    this.validateBtn.setEnabled(false);
    this.jComboBoxValidation.setEnabled(false);
    this.afnSim = null;
    this.afdSim = null;
    this.afdMinSim = null;

}

private void habilitarDefaultAlpha() {
    useSelectedAlphaBtn.setEnabled(true);
}
```

```
private void bloquearDefaultAlpha() {
    useSelectedAlphaBtn.setEnabled(false);
}

/**
 * SETTERS Y GETTERS de la aplicación principal
 */


public Automata getAFN() {
    return AFN;
}

public void setAFN(Automata AFN) {
    this.AFN = AFN;
}

public Automata getAFD() {
    return AFD;
}

public void setAFD(Automata AFD) {
    this.AFD = AFD;
}

public Automata getAFDMin() {
    return AFDMin;
}

public void setAFDMin(Automata AFDMin) {
    this.AFDMin = AFDMin;
}

public Dtrans getDT_AFN() {
    return DT_AFN;
}

public void setDT_AFN(Dtrans DT_AFN) {
    this.DT_AFN = DT_AFN;
}

public Dtrans getDT_AFD() {
    return DT_AFD;
}

public void setDT_AFD(Dtrans DT_AFD) {
    this.DT_AFD = DT_AFD;
}

public Dtrans getDT_AFDMin() {
    return DT_AFDMin;
}

public void setDT_AFDMin(Dtrans DT_AFDMin) {
    this.DT_AFDMin = DT_AFDMin;
}

public boolean isAfnSimResult() {
    return afnSimResult;
}

public boolean isAfdSimResult() {
    return afdSimResult;
}

public boolean isAfdMinSimResult() {
    return afdMinSimResult;
}

public Simulacion getAfnSim() {
    return afnSim;
}

public void setAfnSim(Simulacion afnSim) {
    this.afnSim = afnSim;
}

public Simulacion getAfdSim() {
    return afdSim;
```

```
}

public void setAfdSim(Simulacion afdSim) {
    this.afdSim = afdSim;
}

public Simulacion getAfdMinSim() {
    return afdMinSim;
}

public void setAfdMinSim(Simulacion afdMinSim) {
    this.afdMinSim = afdMinSim;
}

public Configuracion getConf() {
    return conf;
}

public void setConf(Configuracion conf) {
    this.conf = conf;
    CONSTANS.setVacio(this.getConf().getEmptySymbol());
    GraphViz.setDot(this.getConf().getDotPath() + File.separator + "dot");
    GraphViz.setDir(this.getConf().getImgdir());
}


/**
 * Programación de los componentes gráficos y eventos
 */

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
private void initComponents() {

    jPanel1 = new javax.swing.JPanel();
    jTextReGex = new javax.swing.JTextField();
    jLabelReGex = new javax.swing.JLabel();
    jLabel1 = new javax.swing.JLabel();
    jTextAlpha = new javax.swing.JTextField();
    jScrollPane1 = new javax.swing.JScrollPane();
    jListDefaultAlphas = new javax.swing.JList();
    processBtn = new javax.swing.JButton();
    jPanel7 = new javax.swing.JPanel();
    viewAFNbtn = new javax.swing.JButton();
    viewAFDbtn = new javax.swing.JButton();
    viewAFDMinbtn = new javax.swing.JButton();
    jComboBoxGraph = new javax.swing.JComboBox();
    useSelectedAlphaBtn = new javax.swing.JButton();
    jButton1 = new javax.swing.JButton();
    jTabbedPaneTables = new javax.swing.JTabbedPane();
    jPanel2 = new javax.swing.JPanel();
    jScrollPane4 = new javax.swing.JScrollPane();
    jTableAFN = new javax.swing.JTable();
    jPanel3 = new javax.swing.JPanel();
    jScrollPane3 = new javax.swing.JScrollPane();
    jTableAFD = new javax.swing.JTable();
    jPanel4 = new javax.swing.JPanel();
    jScrollPane2 = new javax.swing.JScrollPane();
    jTableAFDMin = new javax.swing.JTable();
    jPanel5 = new javax.swing.JPanel();
    jPanel6 = new javax.swing.JPanel();
    jScrollPane5 = new javax.swing.JScrollPane();
    jTextAreaOutput = new javax.swing.JTextArea();

   cleanBtn = new javax.swing.JButton();
    jLabel2 = new javax.swing.JLabel();
    jTextValidate = new javax.swing.JTextField();
    validateBtn = new javax.swing.JButton();
    jMenuBar1 = new javax.swing.JMenuBar();
    jMenu1 = new javax.swing.JMenu();
    jMenuItemNewRegex = new javax.swing.JMenuItem();
    jMenuItem1 = new javax.swing.JMenuItem();
    jSeparator1 = new javax.swing.JSeparator();
    jMenuItem2 = new javax.swing.JMenuItem();
    jMenu2 = new javax.swing.JMenu();
```

```
        jMenuItem3 = new javax.swing.JMenuItem();
        jComboBoxValidation = new javax.swing.JComboBox();
        jLabelValidationResult = new javax.swing.JLabel();
        jTextValidateResult = new javax.swing.JTextField();
        jMenuBar2 = new javax.swing.JMenuBar();
        jMenu3 = new javax.swing.JMenu();
        jMenuItemNewRegex1 = new javax.swing.JMenuItem();
        jMenuItemConf = new javax.swing.JMenuItem();
        jMenuItemProcesarReGex = new javax.swing.JMenuItem();
        jSeparator2 = new javax.swing.JSeparator();
        jMenuItem5 = new javax.swing.JMenuItem();
        jMenu4 = new javax.swing.JMenu();
        jMenuItem6 = new javax.swing.JMenuItem();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("ReGex Automaton Generator");
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        setForeground(new java.awt.Color(153, 153, 153));

        jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Definiciones",
javax.swing.border.TitledBorder.CENTER, javax.swing.border.TitledBorder.TOP, new
java.awt.Font("Dialog", 0, 12), new java.awt.Color(51, 0, 102))); // NOI18N

        jTextReGex.setToolTipText("Introduzca su expresión regular Aquí");
        jTextReGex.setAutoscrolls(false);
        jTextReGex.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextReGexActionPerformed(evt);
            }
        });
        jTextReGex.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyReleased(java.awt.event.KeyEvent evt) {
                jTextReGexKeyReleased(evt);
            }
        });

        jLabelReGex.setLabelFor(jTextReGex);
        jLabelReGex.setText("Expresión Regular");

        jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        jLabel1.setLabelFor(jTextAlpha);
        jLabel1.setText("Alfabeto");

        jListDefaultAlphas.setModel(new javax.swing.AbstractListModel() {
            String[] strings = { "[a-z]", "[A-Z]", "[0-9]", "[a-zA-Z]", "[0-1]", "[vocals]", "[ALL]"
};
            public int getSize() { return strings.length; }
            public Object getElementAt(int i) { return strings[i]; }
        });
        jListDefaultAlphas.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        jListDefaultAlphas.setValueIsAdjusting(true);
        jListDefaultAlphas.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jListDefaultAlphasMouseClicked(evt);
            }
        });
        jListDefaultAlphas.addListSelectionListener(new javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                jListDefaultAlphasValueChanged(evt);
            }
        });
        jScrollPane1.setViewportView(jListDefaultAlphas);

        processBtn.setText("Procesar");
        processBtn.setEnabled(false);
        processBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                processBtnActionPerformed(evt);
            }
        });

        jPanel7.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Gráficos",
javax.swing.border.TitledBorder.CENTER, javax.swing.border.TitledBorder.DEFAULT_POSITION));

        viewAFNbtn.setText("AFN");
        viewAFNbtn.setEnabled(false);
        viewAFNbtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                viewAFNbtnActionPerformed(evt);
            }
```

```
        });

        viewAFDbtn.setText("AFD");
        viewAFDbtn.setEnabled(false);
        viewAFDbtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                viewAFDbtnActionPerformed(evt);
            }
        });

        viewAFDMinbtn.setText("AFDMin");
        viewAFDMinbtn.setEnabled(false);
        viewAFDMinbtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                viewAFDMinbtnActionPerformed(evt);
            }
        });

        jComboBoxGraph.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Graficar
con...", "GraphViz", "jGraph" }));
        jComboBoxGraph.setEnabled(false);
        jComboBoxGraph.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jComboBoxGraphActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel7Layout = new javax.swing.GroupLayout(jPanel7);
        jPanel7.setLayout(jPanel7Layout);
        jPanel7Layout.setHorizontalGroup(
            jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel7Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                    .addGroup(jPanel7Layout.createSequentialGroup()
                        .addComponent(viewAFNbtn)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(viewAFDbtn)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(viewAFDMinbtn))
                    .addComponent(jComboBoxGraph, javax.swing.GroupLayout.Alignment.TRAILING, 0,
214, Short.MAX_VALUE))
                .addContainerGap())
        );
        jPanel7Layout.setVerticalGroup(
            jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel7Layout.createSequentialGroup()
                .addGroup(jPanel7Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELI
NE)
                    .addComponent(viewAFNbtn)
                    .addComponent(viewAFDbtn)
                    .addComponent(viewAFDMinbtn))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 9,
Short.MAX_VALUE)
                .addComponent(jComboBoxGraph, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        );

        useSelectedAlphaBtn.setText("Usar Alfabeto");
        useSelectedAlphaBtn.setEnabled(false);
        useSelectedAlphaBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                useSelectedAlphaBtnActionPerformed(evt);
            }
        });

        jButton1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/16x16/system-log-
out.png"))); // NOI18N
        jButton1.setText("Salir");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addGap(71, 71, 71)
                                .addComponent(jLabel1))
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addContainerGap()
                                .addComponent(jButton1)))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING, false)
                            .addComponent(useSelectedAlphaBtn, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 129,
Short.MAX_VALUE))
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.TRAILING)
                            .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jTextAlpha, javax.swing.GroupLayout.DEFAULT_SIZE, 591,
Short.MAX_VALUE))
                            .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                                .addGap(93, 93, 93)

   .addComponent(processBtn)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
170, Short.MAX_VALUE)
                                .addComponent(jPanel7, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jLabelReGex)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jTextReGex, javax.swing.GroupLayout.DEFAULT_SIZE, 726,
Short.MAX_VALUE)))
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELI
NE)
                    .addComponent(jLabelReGex)
                    .addComponent(jTextReGex, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                    .addComponent(jLabel1)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jTextAlpha, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addGap(29, 29, 29)
                                .addComponent(processBtn))
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jPanel7, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
)))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 63,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.BASELINE)
                            .addComponent(useSelectedAlphaBtn)
                            .addComponent(jButton1))))
                .addContainerGap())
        );
```

```
        jTabbedPaneTables.setBorder(javax.swing.BorderFactory.createTitledBorder("Tablas de
Transición"));
        jTabbedPaneTables.setTabLayoutPolicy(javax.swing.JTabbedPane.SCROLL_TAB_LAYOUT);
        jTabbedPaneTables.setAutoscrolls(true);

        jPanel2.setAutoscrolls(true);

        jTableAFN.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String [] {
                "Title 1", "Title 2", "Title 3", "Title 4"
            }
        ));
        jTableAFN.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        jScrollPane4.setViewportView(jTableAFN);

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane4, javax.swing.GroupLayout.DEFAULT_SIZE, 592, Short.MAX_VALUE)
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane4, javax.swing.GroupLayout.DEFAULT_SIZE, 207, Short.MAX_VALUE)
        );

        jTabbedPaneTables.addTab("AFN", jPanel2);

        jTableAFD.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String [] {
                "Title 1", "Title 2", "Title 3", "Title 4"
            }
        ));
        jTableAFD.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        jScrollPane3.setViewportView(jTableAFD);

        javax.swing.GroupLayout jPanel3Layout = new javax.swing.GroupLayout(jPanel3);
        jPanel3.setLayout(jPanel3Layout);
        jPanel3Layout.setHorizontalGroup(
            jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane3, javax.swing.GroupLayout.DEFAULT_SIZE, 592, Short.MAX_VALUE)
        );
        jPanel3Layout.setVerticalGroup(
            jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane3, javax.swing.GroupLayout.DEFAULT_SIZE, 207, Short.MAX_VALUE)
        );

        jTabbedPaneTables.addTab("AFD", jPanel3);

        jTableAFDMin.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String [] {
                "Title 1", "Title 2", "Title 3", "Title 4"
            }
        ));
        jTableAFDMin.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        jScrollPane2.setViewportView(jTableAFDMin);

        javax.swing.GroupLayout jPanel4Layout = new javax.swing.GroupLayout(jPanel4);
        jPanel4.setLayout(jPanel4Layout);
        jPanel4Layout.setHorizontalGroup(
            jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE, 592, Short.MAX_VALUE)
```

```
        );
        jPanel4Layout.setVerticalGroup(
            jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE, 207, Short.MAX_VALUE)
        );

        jTabbedPaneTables.addTab("AFDMin", jPanel4);

        jPanel5.setBorder(javax.swing.BorderFactory.createEtchedBorder());

        javax.swing.GroupLayout jPanel5Layout = new javax.swing.GroupLayout(jPanel5);
        jPanel5.setLayout(jPanel5Layout);
        jPanel5Layout.setHorizontalGroup(
            jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 904, Short.MAX_VALUE)
        );
        jPanel5Layout.setVerticalGroup(
            jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 15, Short.MAX_VALUE)
        );

        jPanel6.setBorder(javax.swing.BorderFactory.createTitledBorder("Salida Textual"));

        jScrollPane5.setBackground(new java.awt.Color(255, 255, 204));

        jTextAreaOutput.setBackground(new java.awt.Color(255, 255, 204));
        jTextAreaOutput.setColumns(20);
        jTextAreaOutput.setEditable(false);
        jTextAreaOutput.setRows(5);
        jScrollPane5.setViewportView(jTextAreaOutput);

        cleanBtn.setText("Clean");
        cleanBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                cleanBtnActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel6Layout = new javax.swing.GroupLayout(jPanel6);
        jPanel6.setLayout(jPanel6Layout);
        jPanel6Layout.setHorizontalGroup(
            jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel6Layout.createSequentialGroup()
                .addGap(96, 96, 96)
                .addComponent(cleanBtn)
                .addContainerGap(102, Short.MAX_VALUE))
            .addComponent(jScrollPane5, javax.swing.GroupLayout.DEFAULT_SIZE, 267, Short.MAX_VALUE)
        );
        jPanel6Layout.setVerticalGroup(
            jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel6Layout.createSequentialGroup()
                .addComponent(jScrollPane5, javax.swing.GroupLayout.DEFAULT_SIZE, 252,
Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(cleanBtn)
                .addContainerGap())
        );

        jLabel2.setLabelFor(jTextValidate);
        jLabel2.setText("Texto de Validación");

        jTextValidate.setEnabled(false);

        validateBtn.setText("Validar");
        validateBtn.setEnabled(false);
        validateBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                validateBtnActionPerformed(evt);
            }
        });

        jMenu1.setText("Acciones");

        jMenuItemNewRegex.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.
VK_N, java.awt.event.InputEvent.CTRL_MASK));
        jMenuItemNewRegex.setIcon(new
javax.swing.ImageIcon("/home/cparra/Projects/afgen/afgenjava/src/img/16x16/window-new.png"));
 // NOI18N
        jMenuItemNewRegex.setText("Nuevo...");
```

```
        jMenuItemNewRegex.setToolTipText("Ingresar una nueva Expresión regular");
        jMenuItemNewRegex.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItemNewRegexActionPerformed(evt);
            }
        });
        jMenu1.add(jMenuItemNewRegex);

        jMenuItem1.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_G,
java.awt.event.InputEvent.CTRL_MASK));
        jMenuItem1.setIcon(new javax.swing.ImageIcon("/home/cparra/Projects/afgen/afgenjava/src/img/
16x16/button_ok.png")); // NOI18N
        jMenuItem1.setText("Generar Autómata");
        jMenu1.add(jMenuItem1);
        jMenu1.add(jSeparator1);

        jMenuItem2.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_Q,
java.awt.event.InputEvent.CTRL_MASK));
        jMenuItem2.setIcon(new javax.swing.ImageIcon("/home/cparra/Projects/afgen/afgenjava/src/img/
16x16/system-log-out.png")); // NOI18N
        jMenuItem2.setText("Salir");
        jMenuItem2.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jMenuItem2MouseClicked(evt);
            }
        });
        jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem2ActionPerformed(evt);
            }
        });
        jMenu1.add(jMenuItem2);

        jMenuBar1.add(jMenu1);

        jMenu2.setText("Ayuda");

        jMenuItem3.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_F1,
0));
        jMenuItem3.setIcon(new javax.swing.ImageIcon("/home/cparra/Projects/afgen/afgenjava/src/img/
16x16/help.png")); // NOI18N
        jMenuItem3.setText("About...");
        jMenu2.add(jMenuItem3);

        jMenuBar1.add(jMenu2);

        jComboBoxValidation.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Con...",
"AFN", "AFD", "AFD Mínimo" }));
        jComboBoxValidation.setToolTipText("Seleccione con que Automata Validar");
        jComboBoxValidation.setEnabled(false);

        jLabelValidationResult.setText("Resultado de Validación");

        jTextValidateResult.setBackground(new java.awt.Color(255, 255, 153));
        jTextValidateResult.setEditable(false);
        jTextValidateResult.setFont(new java.awt.Font("Dialog", 1, 12));

        jMenu3.setText("Acciones");

        jMenuItemNewRegex1.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent
.VK_N, java.awt.event.InputEvent.CTRL_MASK));
        jMenuItemNewRegex1.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/img/16x16/window-new.png"))); // NOI18N
        jMenuItemNewRegex1.setText("Nuevo...");
        jMenuItemNewRegex1.setToolTipText("Ingresar una nueva Expresión regular");
        jMenuItemNewRegex1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItemNewRegexActionPerformed(evt);
            }
        });
        jMenu3.add(jMenuItemNewRegex1);

        jMenuItemConf.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_O
, java.awt.event.InputEvent.CTRL_MASK));
        jMenuItemConf.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/16x16/edit-
clear.png"))); // NOI18N
        jMenuItemConf.setText("Configuraciones");
        jMenuItemConf.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItemConfActionPerformed(evt);
```

```
            }
        });
        jMenu3.add(jMenuItemConf);

        jMenuItemProcesarReGex.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyE
vent.VK_G, java.awt.event.InputEvent.CTRL_MASK));
        jMenuItemProcesarReGex.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/16x16/
button_ok.png"))); // NOI18N
        jMenuItemProcesarReGex.setText("Generar Autómata");
        jMenuItemProcesarReGex.setEnabled(false);
        jMenuItemProcesarReGex.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItemProcesarReGexActionPerformed(evt);
            }
        });
        jMenu3.add(jMenuItemProcesarReGex);
        jMenu3.add(jSeparator2);

        jMenuItem5.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_Q,
java.awt.event.InputEvent.CTRL_MASK));
        jMenuItem5.setIcon(new javax.swing.ImageIcon("/home/cparra/Projects/afgen/afgenjava/src/img/
16x16/system-log-out.png")); // NOI18N
        jMenuItem5.setText("Salir");
        jMenuItem5.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jMenuItem2MouseClicked(evt);
            }
        });
        jMenuItem5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem2ActionPerformed(evt);
            }
        });
        jMenu3.add(jMenuItem5);

        jMenuBar2.add(jMenu3);

        jMenu4.setText("Ayuda");

        jMenuItem6.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_F1,
0));
        jMenuItem6.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/img/16x16/help.png"))); // NOI18N
        jMenuItem6.setText("About...");
        jMenuItem6.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem6ActionPerformed(evt);
            }
        });
        jMenu4.add(jMenuItem6);

        jMenuBar2.add(jMenu4);

        setJMenuBar(jMenuBar2);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel5, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addContainerGap())
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jTabbedPaneTables,
javax.swing.GroupLayout.DEFAULT_SIZE, 607, Short.MAX_VALUE)
                                .addGap(5, 5, 5))
                            .addGroup(layout.createSequentialGroup()
                                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                    .addComponent(jLabel2)
                                    .addComponent(jLabelValidationResult))
```

```
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                                    .addGroup(layout.createSequentialGroup()
                                        .addComponent(jTextValidateResult,
javax.swing.GroupLayout.DEFAULT_SIZE, 334, Short.MAX_VALUE)
                                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
RELATED)
                                        .addComponent(jComboBoxValidation,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                                    .addGroup(layout.createSequentialGroup()
                                        .addComponent(jTextValidate,
javax.swing.GroupLayout.DEFAULT_SIZE, 362, Short.MAX_VALUE)
                                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
RELATED)
                                        .addComponent(validateBtn)))
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jPanel6, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addGap(7, 7, 7))))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jTabbedPaneTables, javax.swing.GroupLayout.DEFAULT_SIZE, 258,
Short.MAX_VALUE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)

    .addComponent(validateBtn)
                            .addComponent(jLabel2)
                            .addComponent(jTextValidate, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                            .addComponent(jComboBoxValidation,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabelValidationResult)
                            .addComponent(jTextValidateResult,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addComponent(jPanel6, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 15,
Short.MAX_VALUE)
                .addComponent(jPanel5, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

private void jMenuItem2MouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_jMenuItem2MouseClicked
    System.exit(0);
}//GEN-LAST:event_jMenuItem2MouseClicked

private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jMenuItem2ActionPerformed
    System.exit(0);
}//GEN-LAST:event_jMenuItem2ActionPerformed

private void jMenuItemNewRegexActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jMenuItemNewRegexActionPerformed
    this.jTextReGex.setText("");
    this.jTextAlpha.setText("");
    this.bloquearRegExProcess();
    this.bloquearValidacion();
    this.bloquearVistas();
```

```
}//GEN-LAST:event_jMenuItemNewRegexActionPerformed

private void processBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_processBtnActionPerformed
    this.procesarRegEx();
}//GEN-LAST:event_processBtnActionPerformed

private void jListDefaultAlphasValueChanged(javax.swing.event.ListSelectionEvent evt) {//GEN-
FIRST:event_jListDefaultAlphasValueChanged
    useSelectedAlphaBtn.setEnabled(true);
}//GEN-LAST:event_jListDefaultAlphasValueChanged

private void useSelectedAlphaBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_useSelectedAlphaBtnActionPerformed
    this.copyDefaultAlpha();
}//GEN-LAST:event_useSelectedAlphaBtnActionPerformed

private void cleanBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_cleanBtnActionPerformed
    jTextAreaOutput.setText("");
}//GEN-LAST:event_cleanBtnActionPerformed

private void jMenuItemProcesarReGexActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jMenuItemProcesarReGexActionPerformed
    this.processBtnActionPerformed(evt);
}//GEN-LAST:event_jMenuItemProcesarReGexActionPerformed

private void jListDefaultAlphasMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_jListDefaultAlphasMouseClicked

    // Revisamos si se hizo doble-click
    if (evt.getClickCount() == 2) {
        this.copyDefaultAlpha();
    }
}//GEN-LAST:event_jListDefaultAlphasMouseClicked

private void jTextReGexActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jTextReGexActionPerformed
// TODO add your handling code here:
}//GEN-LAST:event_jTextReGexActionPerformed

private void jTextReGexKeyReleased(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_jTextReGexKeyReleased
    this.checkRegEx();
}//GEN-LAST:event_jTextReGexKeyReleased

private void jMenuItem6ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jMenuItem6ActionPerformed

    if (about == null) {
        about = new About();
    }

    about.setEnabled(true);
    about.setVisible(true);
}//GEN-LAST:event_jMenuItem6ActionPerformed

private void jMenuItemConfActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jMenuItemConfActionPerformed

    if (this.config == null) {
            try {
                this.config = new jFrameConf(this.conf);
            } catch (Exception ex) {
                //Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
                this.jTextAreaOutput.append("ERROR: Hubo algún problema con la
configuración"+ex.getMessage()+"\n");
                this.jTextAreaOutput.append("# <----------------------------------------\n");
            }
    }

    this.config.setEnabled(true);
    this.config.setVisible(true);
}//GEN-LAST:event_jMenuItemConfActionPerformed

private void validateBtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_validateBtnActionPerformed

    this.resetTablaRenderer(jTableAFD);
    this.resetTablaRenderer(jTableAFN);
```

```
    this.resetTablaRenderer(jTableAFDMin);


    switch(this.jComboBoxValidation.getSelectedIndex()) {
        case 1: this.afnSimResult = this.validarAFN();
            this.changeValidationTextResult(this.afnSimResult);
            break;
        case 2: this.afdSimResult = this.validarAFD();
            this.changeValidationTextResult(this.afdSimResult);
            break;
        case 3: this.afdMinSimResult = this.validarAFDMin();
            this.changeValidationTextResult(this.afdMinSimResult);
            break;
        default:
            jTextAreaOutput.append("# --> Seleccione con que Automata desea Validar!\n");

    }
}//GEN-LAST:event_validateBtnActionPerformed

private void viewAFDbtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_viewAFDbtnActionPerformed
    this.viewGraphics(this.AFD);
}//GEN-LAST:event_viewAFDbtnActionPerformed

private void viewAFNbtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_viewAFNbtnActionPerformed
    this.viewGraphics(this.AFN);
}//GEN-LAST:event_viewAFNbtnActionPerformed

private void viewAFDMinbtnActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_viewAFDMinbtnActionPerformed
    this.viewGraphics(this.AFDMin);
}//GEN-LAST:event_viewAFDMinbtnActionPerformed

private void jComboBoxGraphActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jComboBoxGraphActionPerformed
// TODO add your handling code here:
}//GEN-LAST:event_jComboBoxGraphActionPerformed

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
    System.exit(0);
}//GEN-LAST:event_jButton1ActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton cleanBtn;
    private javax.swing.JButton jButton1;
    private javax.swing.JComboBox jComboBoxGraph;
    private javax.swing.JComboBox jComboBoxValidation;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabelReGex;
    private javax.swing.JLabel jLabelValidationResult;
    private javax.swing.JList jListDefaultAlphas;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenu jMenu3;
    private javax.swing.JMenu jMenu4;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuBar jMenuBar2;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JMenuItem jMenuItem3;
    private javax.swing.JMenuItem jMenuItem5;
    private javax.swing.JMenuItem jMenuItem6;
    private javax.swing.JMenuItem jMenuItemConf;
    private javax.swing.JMenuItem jMenuItemNewRegex;
    private javax.swing.JMenuItem jMenuItemNewRegex1;
    private javax.swing.JMenuItem jMenuItemProcesarReGex;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel6;
    private javax.swing.JPanel jPanel7;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
```

```
    private javax.swing.JScrollPane jScrollPane5;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JTabbedPane jTabbedPaneTables;
    private javax.swing.JTable jTableAFD;
    private javax.swing.JTable jTableAFDMin;
    private javax.swing.JTable jTableAFN;
    private javax.swing.JTextField jTextAlpha;
    private javax.swing.JTextArea jTextAreaOutput;
    private javax.swing.JTextField jTextReGex;
    private javax.swing.JTextField jTextValidate;
    private javax.swing.JTextField jTextValidateResult;
    private javax.swing.JButton processBtn;
    private javax.swing.JButton useSelectedAlphaBtn;
    private javax.swing.JButton validateBtn;
    private javax.swing.JButton viewAFDMinbtn;
    private javax.swing.JButton viewAFDbtn;
    private javax.swing.JButton viewAFNbtn;
    // End of variables declaration//GEN-END:variables
}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package app;

import java.awt.Color;
import java.awt.Component;
import java.awt.Font;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;

/**
 * Clase que implementa el coloreo de una celda en particular
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 * @author Fernando Mancía ({@link fernandomancia@gmail.com})
 */
public class OneCellRenderer extends DefaultTableCellRenderer {

    private int fila;
    private int columna;
    private Color background;
    private Color foreground;

    public OneCellRenderer() {

        this.fila = 0;
        this.columna = 0;
        this.background = Color.white;
        this.foreground = Color.black;

    }

    public OneCellRenderer(int filaFin, int columnaFin, Color b, Color f) {
        this.fila = filaFin;
        this.columna = columnaFin;
        this.background = b;
        this.foreground = f;
    }

    @Override
    public Component getTableCellRendererComponent(JTable table, Object value, boolean selected,
boolean focused, int row, int column) {
        setEnabled(table == null || table.isEnabled()); // see question above

        if ((row == this.fila) && (column == this.columna)) {
            setBackground(this.background);
            setForeground(this.foreground);
            setFont(new Font("Verdana",Font.BOLD,12));
        } else if (column == 0) {
            setBackground(Color.gray);
            setForeground(Color.white);
            setFont(new Font("Verdana",Font.BOLD,12));
        } else {
            setBackground(Color.white);
            setForeground(Color.black);
        }
```

```
        setHorizontalAlignment(DefaultTableCellRenderer.CENTER);
        super.getTableCellRendererComponent(table, value, selected, focused, row, column);

        return this;
    }
}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package app;

import java.awt.Color;
import java.awt.Component;
import java.awt.Font;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;

/**
 *
 * @author Cristhian Parra ({@link cdparra@gmail.com})
 */
public class OneColumnRenderer extends DefaultTableCellRenderer {

    private int columna;
    private Color background;
    private Color foreground;

    public OneColumnRenderer() {
        this.columna = 0;
        this.background = Color.white;
        this.foreground = Color.black;

    }

    public OneColumnRenderer(int columna, Color b, Color f) {
        this.columna = columna;
        this.background = b;
        this.foreground = f;
    }

    @Override
    public Component getTableCellRendererComponent(JTable table, Object value, boolean selected,
boolean focused, int row, int column) {
        setEnabled(table == null || table.isEnabled()); // see question above

        if ((column == this.columna)) {
            setBackground(this.background);
            setForeground(this.foreground);
            setFont(new Font("Verdana", Font.BOLD, 12));
        } else {
            setBackground(Color.white);
            setForeground(Color.black);
        }


        setHorizontalAlignment(DefaultTableCellRenderer.CENTER);
        super.getTableCellRendererComponent(table, value, selected, focused, row, column);

        return this;
    }
}
/*
 * jFrameConf.java
 *
 * Created on 16 de noviembre de 2008, 07:19 PM
 */

package app;

import afgenjava.CONSTANS;
import graphviz.GraphViz;
import java.io.File;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFileChooser;
import org.simpleframework.xml.Serializer;
import org.simpleframework.xml.load.Persister;
```

```
/**
 *
 * @author  Cristhian Parra ({@link cdparra@gmail.com})
 */
public class jFrameConf extends javax.swing.JFrame {

    private boolean cancelled = true;
    private String gvPaht = "";
    private Configuracion conf;
    private Configuracion mainConf;

    /** Creates new form jFrameConf */
    public jFrameConf(Configuracion mainConf) throws Exception {
        this.mainConf = mainConf;
        Serializer serializer = new Persister();
        File source = new File("conf.xml");

        this.conf = serializer.read(Configuracion.class, source);

        initComponents();
    }

    private File getInicio() {
        String initDir = this.conf.getDotPath();
        File finitDir = new File(initDir);

        if (!finitDir.isDirectory()) {
            initDir = ".";
            finitDir = new File(initDir);
        }
        return finitDir;
    }


    public boolean isCancelled() {
        return cancelled;
    }

    public void setCancelled(boolean cancelled) {
        this.cancelled = cancelled;
    }

    public String getGvPaht() {
        return gvPaht;
    }

    public void setGvPaht(String gvPaht) {
        this.gvPaht = gvPaht;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents() {

        chooser = new javax.swing.JFileChooser(this.getInicio());
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        jTextGraphvizPath = new javax.swing.JTextField(this.conf.getDotPath());
        jPanel1 = new javax.swing.JPanel();
        jLabel2 = new javax.swing.JLabel();
        jTextGraphvizVacio = new javax.swing.JTextField(this.conf.getEmptySymbol());
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jTextGraphvizTempDir = new javax.swing.JTextField(this.conf.getImgdir());
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();

        chooser.setFileSelectionMode(javax.swing.JFileChooser.DIRECTORIES_ONLY);

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("AfGen - Configuraciones");
        setForeground(java.awt.Color.white);
```

```
        jButton1.setText("Guardar");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jButton2.setText("Cancelar");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jLabel1.setText("GraphViz Path");

        jPanel1.setBorder(javax.swing.BorderFactory.createEtchedBorder());

        jLabel2.setFont(new java.awt.Font("Alien Encounters", 1, 36));
        jLabel2.setForeground(new java.awt.Color(0, 0, 102));
        jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel2.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/img/logo_afgen.png"))); // NOI18N

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jLabel2)
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );

        jLabel3.setText("Etiqueta Vacía");

        jLabel4.setFont(new java.awt.Font("As seen on TV", 1, 14));
        jLabel4.setForeground(new java.awt.Color(0, 0, 102));
        jLabel4.setText("Edite las Variables de configuración");

        jLabel5.setText("Directorio de Imágenes");

        jLabel6.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/img/16x16/fileopen.png"))); // NOI18N
        jLabel6.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jLabel6MouseClicked(evt);
            }
        });

        jLabel7.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/img/16x16/fileopen.png"))); // NOI18N
        jLabel7.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jLabel7MouseClicked(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(106, 106, 106)
                        .addComponent(jButton1)
```

```
                        .addGap(54, 54, 54)
                        .addComponent(jButton2))
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAIL
ING)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel5)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

                                .addComponent(jTextGraphvizTempDir,
javax.swing.GroupLayout.PREFERRED_SIZE, 185, javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel1)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jTextGraphvizPath,
javax.swing.GroupLayout.PREFERRED_SIZE, 185, javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel3)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jTextGraphvizVacio,
javax.swing.GroupLayout.PREFERRED_SIZE, 185, javax.swing.GroupLayout.PREFERRED_SIZE)))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 19,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 19,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(74, 74, 74)
                        .addComponent(jLabel4)))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                            .addComponent(jLabel1)
                            .addComponent(jTextGraphvizPath, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                            .addComponent(jLabel5)
                            .addComponent(jTextGraphvizTempDir,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                            .addComponent(jLabel3)
                            .addComponent(jTextGraphvizVacio,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 31,
Short.MAX_VALUE)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                            .addComponent(jButton1)
                            .addComponent(jButton2)))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(7, 7, 7)
                        .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 15,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 15,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap())
        );
```

```
        pack();
    }// </editor-fold>//GEN-END:initComponents

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton2ActionPerformed
    this.setCancelled(true);
    this.setEnabled(false);
    this.setVisible(false);
}//GEN-LAST:event_jButton2ActionPerformed

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed

    Serializer serializer = new Persister();
    this.setGvPaht(jTextGraphvizPath.getText());

    this.conf = new Configuracion(jTextGraphvizPath.getText(), 1);
    this.conf.setEmptySymbol(this.jTextGraphvizVacio.getText());
    this.conf.setImgdir(this.jTextGraphvizTempDir.getText());

    File result = new File("conf.xml");
    try {
        serializer.write(this.conf, result);
    } catch (Exception ex) {
        Logger.getLogger(jFrameConf.class.getName()).log(Level.SEVERE, null, ex);
    }

    GraphViz.setDir(this.conf.getImgdir());
    GraphViz.setDot(this.conf.getDotPath()+File.separator+"dot");
    CONSTANS.setVacio(this.conf.getEmptySymbol());

    this.mainConf.setDotPath(this.conf.getDotPath());
    this.mainConf.setEmptySymbol(this.conf.getEmptySymbol());
    this.mainConf.setImgdir(this.conf.getImgdir());

    this.setCancelled(false);
    this.setEnabled(false);
    this.setVisible(false);


}//GEN-LAST:event_jButton1ActionPerformed

private void jLabel7MouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_jLabel7MouseClicked
    int status = this.chooser.showOpenDialog(null);
    if (status == JFileChooser.APPROVE_OPTION) {
        File selectedFile = chooser.getSelectedFile();
        this.conf.setDotPath(selectedFile.getAbsolutePath());
        this.jTextGraphvizPath.setText(this.conf.getDotPath());
    } else if (status == JFileChooser.CANCEL_OPTION) {
        // nothing
    }
}//GEN-LAST:event_jLabel7MouseClicked

private void jLabel6MouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_jLabel6MouseClicked
    int status = this.chooser.showOpenDialog(null);
    if (status == JFileChooser.APPROVE_OPTION) {
        File selectedFile = chooser.getSelectedFile();
        this.conf.setImgdir(selectedFile.getAbsolutePath());
        this.jTextGraphvizTempDir.setText(this.conf.getImgdir());
    } else if (status == JFileChooser.CANCEL_OPTION) {
        // nothing
    }
}//GEN-LAST:event_jLabel6MouseClicked

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JFileChooser chooser;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextGraphvizPath;
```

```
    private javax.swing.JTextField jTextGraphvizTempDir;
    private javax.swing.JTextField jTextGraphvizVacio;
    // End of variables declaration//GEN-END:variables
}
```

5. Paquete graphviz

```
package graphviz;

// GraphViz.java - a simple API to call dot from Java programs

/*$Id$*/
/**
 ******************************************************************************
 *                                                                            *
 *              (c) Copyright 2003 Laszlo Szathmary                           *
 *                                                                            *
 * This program is free software; you can redistribute it and/or modify it    *
 * under the terms of the GNU Lesser General Public License as published by   *
 * the Free Software Foundation; either version 2.1 of the License, or        *
 * (at your option) any later version.                                        *
 *                                                                            *
 * This program is distributed in the hope that it will be useful, but        *
 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY *
 * or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public    *
 * License for more details.                                                  *
 *                                                                            *
 * You should have received a copy of the GNU Lesser General Public License   *
 * along with this program; if not, write to the Free Software Foundation,    *
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.                              *
 *                                                                            *
 ******************************************************************************
 *
 * Modified by Cristhian Parra ({@link cdparra@gmail.com})<br>
 */

import java.io.*;
import java.util.*;

/**
 * <dl>
 * <dt>Purpose: GraphViz Java API
 * <dd>
 *
 * <dt>Description:
 * <dd> With this Java class you can simply call dot
 *      from your Java programs
 * <dt>Example usage:
 * <dd>
 * <pre>
 *    GraphViz gv = new GraphViz();
 *    gv.addln(gv.start_graph());
 *    gv.addln("A -> B;");
 *    gv.addln("A -> C;");
 *    gv.addln(gv.end_graph());
 *    System.out.println(gv.getDotSource());
 *
 *    File out = new File("out.gif");
 *    gv.writeGraphToFile(gv.getGraph(gv.getDotSource()), out);
 * </pre>
 * </dd>
 *
 * </dl>
 *
 * @version v0.1, 2003/12/04 (Decembre)
 * @author  Laszlo Szathmary (<a href="szathml@delfin.unideb.hu">szathml@delfin.unideb.hu</a>)
 */
public class GraphViz
{
   /**
    * The dir where temporary files will be created.
    */
   private static String TEMP_DIR   = "/tmp";

   /**
    * Where is your dot program located? It will be called externally.
    */
   private static String DOT        = "/usr/bin/dot";

   /**
    * The source of the graph written in dot language.
    */
            private StringBuffer graph = new StringBuffer();
```

```
/**
 * Constructor: creates a new GraphViz object that will contain
 * a graph.
 */
public GraphViz() {
}

/**
 * Returns the graph's source description in dot language.
 * @return Source of the graph in dot language.
 */
public String getDotSource() {
   return graph.toString();
}

/**
 * Adds a string to the graph's source (without newline).
 */
public void add(String line) {
   graph.append(line);
}

/**
 * Adds a string to the graph's source (with newline).
 */
public void addln(String line) {
   graph.append(line+"\n");
}

/**
 * Adds a newline to the graph's source.
 */
public void addln() {
   graph.append('\n');
}

/**
 * Returns the graph as an image in binary format.
 * @param dot_source Source of the graph to be drawn.
 * @return A byte array containing the image of the graph.
 */
public byte[] getGraph(String dot_source)
{
   File dot;
   byte[] img_stream = null;

   try {
      dot = writeDotSourceToFile(dot_source);
      if (dot != null)
      {
         img_stream = get_img_stream(dot);
         if (dot.delete() == false)
            System.err.println("Warning: "+dot.getAbsolutePath()+" could not be deleted!");
         return img_stream;
      }
      return null;
   } catch (java.io.IOException ioe) { return null; }
}

/**
 * Writes the graph's image in a file.
 * @param img   A byte array containing the image of the graph.
 * @param file  Name of the file to where we want to write.
 * @return Success: 1, Failure: -1
 */
public int writeGraphToFile(byte[] img, String file)
{
   File to = new File(file);
   return writeGraphToFile(img, to);
}

/**
 * Writes the graph's image in a file.
 * @param img   A byte array containing the image of the graph.
 * @param to    A File object to where we want to write.
 * @return Success: 1, Failure: -1
 */
public int writeGraphToFile(byte[] img, File to)
{
```

```
   try {
      FileOutputStream fos = new FileOutputStream(to);
      fos.write(img);
      fos.close();
   } catch (java.io.IOException ioe) { return -1; }
   return 1;
}

/**
 * It will call the external dot program, and return the image in
 * binary format.
 * @param dot Source of the graph (in dot language).
 * @return The image of the graph in .gif format.
 */
private byte[] get_img_stream(File dot)
{
   File img;
   byte[] img_stream = null;

   try {
      img = File.createTempFile("graph_", ".gif", new File(this.TEMP_DIR));
      String temp = img.getAbsolutePath();

      Runtime rt = Runtime.getRuntime();
      String cmd = DOT + " -Tgif "+dot.getAbsolutePath()+" -o"+img.getAbsolutePath();
      Process p = rt.exec(cmd);
      p.waitFor();

      FileInputStream in = new FileInputStream(img.getAbsolutePath());
      img_stream = new byte[in.available()];
      in.read(img_stream);
      // Close it if we need to
      if( in != null ) in.close();

      if (img.delete() == false)
         System.err.println("Warning: "+img.getAbsolutePath()+" could not be deleted!");
   }
   catch (java.io.IOException ioe) {
      System.err.println("Error:    in I/O processing of tempfile in dir "+this.TEMP_DIR+"\n");
      System.err.println("       or in calling external command");
      ioe.printStackTrace();
   }
   catch (java.lang.InterruptedException ie) {
      System.err.println("Error: the execution of the external program was interrupted");
      ie.printStackTrace();
   }

   return img_stream;
}

/**
 * Writes the source of the graph in a file, and returns the written file
 * as a File object.
 * @param str Source of the graph (in dot language).
 * @return The file (as a File object) that contains the source of the graph.
 */
private File writeDotSourceToFile(String str) throws java.io.IOException
{
   File temp;
   try {
      temp = File.createTempFile("graph_", ".dot.tmp", new File(this.TEMP_DIR));
      FileWriter fout = new FileWriter(temp);
      fout.write(str);
      fout.close();
   }
   catch (Exception e) {
      System.err.println("Error: I/O error while writing the dot source to temp file!");
      return null;
   }
   return temp;
}

/**
 * Returns a string that is used to start a graph.
 * @return A string to open a graph.
 */
public String start_graph() {
   return "digraph G {";
}
```

```
    /**
     * Returns a string that is used to end a graph.
     * @return A string to close a graph.
     */
    public String end_graph() {
       return "}";
    }

    /**
     * Clase especifica para dibujar directamente en un grafico de salida a partir
     * de un String que contiene toda la definición del Grafo.
     * @param dotString
     * @param where
     */
    public void dibujar(String dotString, String where) {
         this.addln(dotString);
         System.out.println(this.getDotSource());
         File out = new File(where);
         this.writeGraphToFile(this.getGraph(this.getDotSource()), out);
    }

    /**
     * Método para modificar en tiempo de ejecución el path al ejecutable de
     * Graphviz
     * @param dot
     */
    public static void setDot(String dot) {
        GraphViz.DOT = dot;
    }

    /**
     * Método para modificar en tiempo de ejecución el path al directorio donde
     * guardaremos las imágenes generadas por el programa
     * @param dir
     */
    public static void setDir(String dir) {
        GraphViz.TEMP_DIR = dir;
    }

    /**
     * Verifica que Graphviz esté instalado en el directorio especificado
     * @return true si están instalado falso en caso contrario
     */
    public boolean testGraphViz() {
       String dotfile = GraphViz.DOT;
       if (File.separator.compareTo("\\")==0) {
           dotfile +=".exe";
       }

       File dot = new File(dotfile);
       return dot.exists();
    }
}
```