# MANUAL TÉCNICO DE LA APLICACIÓN AUTÓMATAS FINITOS

ALEJANDRO CASTAÑOS ROJAS

ESTEFANY MURIEL CANO

UNIVERSIDAD DE ANTIOQUIA

2017

**Página**

## 2. Requisitos del sistema

Para lograr la ejecución adecuada de la aplicación de autómatas finitos, su equipo debe de tener, preferiblemente,  sistema operativo Windows, además del IDE Netbeans, recomendablemente una versión reciente, un lector PDF y editor de texto.

## 3. Descripción de métodos implementados

### Paquete controlador:

### En la clase ControladorAutomata tenemos los siguientes métodos:

| Modifier and Type | Method and Description |
|---|---|
| void | **actualizarAutomata**()<br>Este método luego de haber sacado los estados extraños, permite reestablecer todos los valores del autómata, actualiza los estados sin los estados que fueron extraidos, y los estados de aceptación y las particiones también los actualiza. |
| void | **actualizarParticiones**(java.util.ArrayList<java.lang.String> estadosAceptacion, java.util.ArrayList<java.lang.String> estados)<br>Este método actualiza las particiones de aceptación o de rechazo cuando el autómata es convertido a determinístico, dichas particiones son utilizadas en simplificar y en verificar la hilera. |
| java.util.ArrayList<java.lang.String> | **completarTransicion**(java.util.ArrayList<java.lang.String> transicion, java.lang.String[] estados, java.util.ArrayList<java.lang.String> estados1)<br>Este método mira el estado en cada transición, y si almenos contiene un caracter que pereteneza a a cualquiera de los nuevos estados, se reemplaza con el estado correspondiente |
| void | **construirAutomata**(java.util.ArrayList<java.util.ArrayList> particiones)<br>De acuerdo a las particiones que se tomaron del método simplificar, este método une las transiciones basandose en los estados que estan en cada partición, además actualiza las transiciones y finalemte actualiza el autómata, es decir, sus nuevos estados, estados de aceptación y transiciones. |
| void | **convertirArray**(java.util.ArrayList<java.lang.String> a, java.lang.String[] b)<br>Este método permite convertir un ArrayList de Strings en un arreglo de String |
| java.util.ArrayList<java.util.ArrayList> | **convertirEnDeterministico**()<br>Este método permite convertir un autómata no deterministico a deterministico, tomando el primer estado con sus transiciones del autómata inicial, y a partir de este mediante la concatenación de estados y transiciones, contruir el autómata final. |
| int | **convertirEstados**(java.lang.String a)<br>Este método permite retornar la posición del estado que se le entra por parámetro para saber su ubicación dentro del arreglo. |
| java.util.ArrayList<java.util.ArrayList> | **unionAutomata**(boolean tipo)<br>Este método permite realizar la unión e intersección de uno o dos autómatas, para ello se recibe un booleano que indica el tipo de proceso a seguir, si es true se realiza la unión y si es false, se realiza la intersección, para esto se toman los estados iniciales y se juntan con sus respectivas transiciones, y a partir de esos nuevos estados se empieza a contruir el nuevo autómata. |
| java.util.ArrayList<java.lang.String> | **unirTransiciones**(java.lang.String[] estados)<br>Este método permite hacer la unión de las transiciones de varios estados, tomando las transiciones de cada estado que se va unir y luego compara las transiciones que se van a unir para evitar la repetición de estados en las transiciones finales. |
| boolean | **verificarHilera**(java.lang.String hilera)<br>Este método permite que al tener un autómata ya definido, podamos relaizar un reconocimiento de secuencias mediante el autómata, para saber si la secuencia es válida o no, nos ubicamos en el estado inicial, luego miramos que simbolo entra y avanzamos al estado que indique la transición, hasta llegar al final de la hilera que es denotada por el símbolo '*', cuando se llega al final se mira en que tipo de estado se finalizó y si es de aceptación se acepta la secuencia, de lo contrario se rechaza. |

| | |
|---|---|
| int | **convertirSimbolos**(java.lang.String a)<br>Este método permite retornar la posición del simbolo que se le entra por parámetro para saber su ubicación dentro del arreglo. |
| java.lang.String[] | **convertirString**(java.lang.String a)<br>Este método permite tomar un String y llevarlo a un arreglo con un caracter del String en cada posición. |
| java.util.ArrayList<java.lang.String> | **copiaArray**(java.util.ArrayList<java.lang.String> array)<br>Este método crea una copia con todos los elementos de un Arraylist. |
| java.util.ArrayList<java.lang.String> | **creaNuevaParticion**(java.util.ArrayList<java.lang.String> particion, java.util.ArrayList<java.lang.String> noContenidos, int posSym)<br>Este método permite crear una nueva partición para ello se mira la partición que se va a modificar, y la transición que hace que esta partición se tenga que modificar, luego miramos los estados de la transición correspondiente y si estos estados no estan contenidos en la partición se crea un nuevo array con la nueva partición |
| boolean | **definirEstadoDeAceptacion**(java.lang.String[] estados, boolean tipo)<br>Permite decidir que estado de aceptación va a tener la unión o intersección de varios estados * |
| boolean | **esDeterministico**()<br>Permite verificar si el autómata ingresado es determinístico o no determinístico, mediante la revisión de sus transiciones. |
| boolean | **esEstadoDeAceptacion**(java.lang.String estado)<br>Permite determinar si el estado ingresado es de aceptación. |
| void | **estadosAceptacion**()<br>Aunque su nombre indique que selecciona los estados de aceptación, en realidad separa estados de aceptación de los estados de rechazo generando así 2 particiones; una partición P0 con los estados de rechazo y una particion P1 con los estados de aceptación. |
| void | **estadosExtraños**(int p)<br>Este método permite tomar las transiciones actuales del autómata, y mira cuales estados estan contenidos en las transiciones para así ir llenando el vector de visitados e ir identificando que estados son extraños, finalmente actulizamos las transiciones sin estados extraños. |
| void | **estadosIniciales**()<br>Este método guarda los estado iniciales que fueron ingrresados en la tabla del autómata, toma solo los estados que en el campo "E.I" tienen el siguiente simbolo "#". |
| java.util.ArrayList<java.lang.String> | **estadosNoContenidos**(java.util.ArrayList<java.lang.String> particion, java.util.ArrayList<java.lang.String> transicionASym)<br>Entra el array con la partición a tratar y otro Array con los símbolos a buscar dentro del primer Array, este arrojará el Array con los símbolos que no estan en esa partición. |
| boolean | **estadoValido**(java.lang.String[] estados, java.lang.String estado)<br>Permite verificar si el estado que se va a guardar es válido. |
| boolean | **estaEnAceptacion**(java.lang.String estado)<br>Este método determina si un estado es de aceptación o no |
| boolean | **existeEstado**(java.util.ArrayList<java.lang.String> estados, java.lang.String estado)<br>Este método permite verificar si ya existe un estado, para evitar las repeticiones en el momento de agregar un nuevo estado. |
| boolean | **existeEstadoAceptacion**(java.util.ArrayList<java.lang.String> estadosAceptacion, java.lang.String estado)<br>Este método analiza los estados de aceptación que se tienen en el momento y verifica que el estado que va a entrar no se encuentre repetido entre los estados de aceptación que ya están. |

| | |
|---|---|
| boolean | **existeSimbolo**(java.lang.String simbolo)<br>Este método permite verificar si el simbolo es correcto, para evitar evaluar simbolos que no pertenecen al autómata. |
| **AutomataF** | **getAf**() |
| javax.swing.table.DefaultTableModel | **getDtm**() |
| java.lang.String[] | **getEstados**() |
| java.util.ArrayList<java.util.ArrayList> | **getParticiones**() |
| java.lang.String[] | **getSimbolos**() |
| java.util.ArrayList<java.util.ArrayList> | **getTransDePart**() |
| java.util.ArrayList<java.util.ArrayList> | **guardarAutomata**()<br>Este método trae de la tabla todas las transiciones y las guarda en un ArrayList de ArrayList con datos de tipo String |
| java.lang.String | **intercambiarEstados**(java.util.ArrayList<java.lang.String> estados, java.lang.String estado)<br>Este método permite tomar un estado que se encuentra desordenado, y ordenarlo de la forma en que se define previamente, es decir, si existe un estado compuesto de más estados con un orden específico, los otros estados que contenga sus mismos estados, tambien van a estar con el mismo orden del estado previamente definido. |
| void | **llenarVisitados**()<br>Este método llena con ceros, el vector de visitados, el cual se utiliza para sacar los estados extraños |
| java.util.ArrayList<java.util.ArrayList> | **organizarAutomata**(java.util.ArrayList<java.lang.String> estados, java.util.ArrayList<java.util.ArrayList> automata)<br>Toma el autómata que se construyó y reordena los estados de las transiciones,para que coincidan con los estados ya definidos, para evitar confusiones en el usuario,y sea más ordenado el proceso de reconocimiento. |
| java.lang.String | **quitarEstadosRepetidos**(java.lang.String a, java.lang.String b)<br>Este método compara dos estados y evita que hayan repeticiones de estados cuando se realice la unión de los mismos. |
| java.util.ArrayList<java.lang.String> | **revisarTransiciones**(java.util.ArrayList<java.lang.String> transiciones)<br>Este método permite revisar las transiciones de un estado, y en caso de que un transición vaya a dos estados, los concatena. |
| void | **setAf**(**AutomataF** af) |
| void | **setDtm**(javax.swing.table.DefaultTableModel dtm) |
| void | **setEstados**(java.lang.String[] estados) |
| void | **setParticiones**(java.util.ArrayList<java.util.ArrayList> particiones) |
| void | **setSimbolos**(java.lang.String[] simbolos) |
| void | **setTransDePart**(java.util.ArrayList<java.util.ArrayList> transDePart) |
| void | **simplificar**()<br>Este método permite mirar que estados son equivalente y eliminar los estados extraños, para esto tomamos las particiones con los estados de aceptación y de rechazo, y vamos mirando sus transiciones cuando entra determinado simbolo, si sus transiciones van a una partición diferente se crea otra nueva partición con el estado que difiere y se agrega a un array de particiones, se sigue analizando cada partición hasta que terminemos todas las particiones con los diferentes símbolos, finalmente actualiza los valores del autómata: transiciones, estados,estados de Aceptación. |
| java.util.ArrayList<java.util.ArrayList> | **transicionesParticion**(java.util.ArrayList<java.lang.String> particion)<br>Este método evalua la partición que se le ingresa por parámetro, y trae del autómata inicial las transiciones pertenecientes a cada elemento del Array. |

## Constructor de la Clase ControladorAutomata:

**ControladorAutomata**

```
public ControladorAutomata(AutomataF af,
                           javax.swing.table.DefaultTableModel dtm)
```

Constructor de la clase ControladorAutomata

**Parameters:**

af - El autómata a trabajar

dtm - La tabla perteneciente.

**Explicación más detallada de los métodos:**

- **convertirSimbolos**

```
public int convertirSimbolos(java.lang.String a)
```

Este método permite retornar la posición del simbolo que se le entra por parámetro para saber su ubicación dentro del arreglo.

**Parameters:**

a - String con el simbolo que se quiere buscar

**Returns:**

entero con la posición del símbolo del estado en el arreglo

- **llenarVisitados**

```
public void llenarVisitados()
```

Este método llena con ceros, el vector de visitados, el cual se utiliza para sacar los estados extraños

- **convertirEstados**

```
public int convertirEstados(java.lang.String a)
```

Este método permite retornar la posición del estado que se le entra por parámetro para saber su ubicación dentro del arreglo.

**Parameters:**

a - String con el simbolo que se quiere buscar

**Returns:**

entero con la posición del estado en el arreglo

- **guardarAutomata**

```
public java.util.ArrayList<java.util.ArrayList> guardarAutomata()
```

Este método trae de la tabla todas las transiciones y las guarda en un ArrayList de ArrayList con datos de tipo String

**Returns:**

un ArrayLsit con todas las transiciones del autómata

- **estadoValido**

- `public boolean estadoValido(java.lang.String[] estados,`
  `java.lang.String estado)`

  Permite verificar si el estado que se va a guardar es válido.

  **Parameters:**

  `estados - Arreglo con los estados del autómata`

  `estado - String con el estado a verificar`

  **Returns:**

  `boolen en true si existe estado, false de lo contrario.`

- **estadosAceptacion**

  `public void estadosAceptacion()`

  Aunque su nombre indique que selecciona los estados de aceptación, en realidad separa estados de aceptación de los estados de rechazo generando así 2 particiones; una partición P0 con los estados de rechazo y una particion P1 con los estados de aceptación.

- **estadosIniciales**

  `public void estadosIniciales()`

  Este método guarda los estado iniciales que fueron ingrresados en la tabla del autómata, toma solo los estados que en el campo "E.I" tienen el siguiente simbolo "#".

- **esEstadoDeAceptacion**

  `public boolean esEstadoDeAceptacion(java.lang.String estado)`

  Permite determinar si el estado ingresado es de aceptación.

  **Parameters:**

  `estado - String con el estado que se desea conocer si es de aceptación`

  **Returns:**

  `un booleano con la confirmación de si es un estado de aceptación o no.`

- **esDeterministico**

  `public boolean esDeterministico()`

  Permite verificar si el autómata ingresado es deterministico o no deterministico, mediante la revisión de sus transiciones.

  **Returns:**

  `un booleano con la confirmación de si es deterministico o no.`

- **quitarEstadosRepetidos**

- `public java.lang.String quitarEstadosRepetidos(java.lang.String a,`

```
                                        java.lang.String b)
```

Este método compara dos estados y evita que hayan repeticiones de estados cuando se realice la unión de los mismos.

**Parameters:**

```
a - String con el primer estado a comparar

b - String con el segundo estado a comparar
```

**Returns:**

```
Un String con el nuevo estado sin repeticiones de estados en el mismo.
```

- **unirTransiciones**

```
public java.util.ArrayList<java.lang.String> unirTransiciones(java.lang.String
[] estados)
```

Este método permite hacer la unión de las transiciones de varios estados, tomando las transiciones de cada estado que se va unir y luego compara las transiciones que se van a unir para evitar la repetición de estados en las transiciones finales.

**Parameters:**

```
estados - Arreglo de Strings con los estados a los cuales se les va a
hacer la unión de transiciones.
```

**Returns:**

```
Un Array de Strings con las transiciones previamente concatenadas y sin
repeticiones en ellas.
```

- **existeEstado**

- ```
  public boolean existeEstado(java.util.ArrayList<java.lang.String> estados,
                              java.lang.String estado)
  ```

Este método permite verificar si ya existe un estado, para evitar las repeticiones en el momento de agregar un nuevo estado.

**Parameters:**

```
estados - Lista con los estados existentes hasta el momento.

estado - String con el estado a comparar.
```

**Returns:**

```
un booleano en true si el estado ya existe, o false de lo contrario.
```

- **existeSimbolo**

```
public boolean existeSimbolo(java.lang.String simbolo)
```

Este método permite verificar si el simbolo es correcto, para evitar evaluar simbolos que no pertenecen al autómata.

**Parameters:**

```
simbolo - String con el simbolo a verificar.
```

**Returns:**
```

un booleano en true si el simbolo es correcto, o false de lo contrario.

- **definirEstadoDeAceptacion**

- `public boolean definirEstadoDeAceptacion(java.lang.String[] estados, boolean tipo)`

  Permite decidir que estado de aceptación va a tener la unión o intersección de varios estados *

  **Parameters:**

  `estados` - Arreglo con los estados a verificar su estado de aceptación

  `tipo` - Ingresa un booleano que define el tipo de operación

  **Returns:**

  un booleano en true si al menos uno de los estados es de aceptación, o false de lo contrario.

- **revisarTransiciones**

`public java.util.ArrayList<java.lang.String> revisarTransiciones(java.util.ArrayList<java.lang.String> transiciones)`

Este método permite revisar las transiciones de un estado, y en caso de que un transición vaya a dos estados, los concatena.

**Parameters:**

`transiciones` - Array de String con las transciones a revisar.

**Returns:**

Un ArrayList de Strings con las transiciones concatenadas

- **existeEstadoAceptacion**

- `public boolean existeEstadoAceptacion(java.util.ArrayList<java.lang.String> estadosAceptacion, java.lang.String estado)`

  Este método analiza los estados de aceptación que se tienen en el momento y verifica que el estado que va a entrar no se encuentre repetido entre los estados de aceptación que ya están.

  **Parameters:**

  `estadosAceptacion` - Lista que contiene los estados de aceptación que se tienen hasta el momento

  `estado` - String con el estado a verificar.

  **Returns:**

  un booleano en true si el estado ya existe en la lista o false si no existe

- **actualizarParticiones**

- `public void actualizarParticiones(java.util.ArrayList<java.lang.String> estadosAceptacion,`

```
java.util.ArrayList<java.lang.String> estados)
```

Este método actualiza las particiones de aceptación o de rechazo cuando el autómata es convertido a determinístico, dichas particiones son utilizadas en simplificar y en verificar la hilera. Para actualizar se toman los estados que no pertenecen a la lista de estado de aceptación y se asigan en la partición de estados de rechazo.

**Parameters:**

```
estadosAceptacion - Lista con los estados de aceptación del autómata.

estados - lista con los estados del autómata.
```

- **convertirEnDeterministico**

```
public java.util.ArrayList<java.util.ArrayList> convertirEnDeterministico()
```

Este método permite convertir un autómata no deterministico a deterministico, tomando el primer estado con sus transiciones del autómata inicial, y a partir de este mediante la concatenación de estados y transiciones, contruir el autómata final.

**Returns:**

```
un ArrayList de ArrayList de tipo String con el nuevo autómata.
```

- **unionAutomata**

```
public java.util.ArrayList<java.util.ArrayList> unionAutomata(boolean tipo)
```

Este método permite realizar la unión e intersección de uno o dos autómatas, para ello se recibe un booleano que indica el tipo de proceso a seguir, si es true se realiza la unión y si es false, se realiza la intersección, para esto se toman los estados iniciales y se juntan con sus respectivas transiciones, y a partir de esos nuevos estados se empieza a contruir el nuevo autómata.

**Parameters:**

```
tipo - booleano que indica si se va a hacer la unión o la intersección.
```

**Returns:**

```
Array con el nuevo autómata.
```

- **convertirString**

```
public java.lang.String[] convertirString(java.lang.String a)
```

Este método permite tomar un String y llevarlo a un arreglo con un caracter del String en cada posición.

**Parameters:**

```
a - String que se quiere almacenar en un arreglo.
```

**Returns:**

```
Un arreglo con el String inicial.
```

- **convertirArray**

- ```
  public void convertirArray(java.util.ArrayList<java.lang.String> a,
                             java.lang.String[] b)
  ```
  ```

Este método permite convertir un ArrayList de Strings en un arreglo de String

**Parameters:**

```
a - entra el Array que se va a convertir

b - entra el arreglo donde se van a pasar los datos del Array
```

- **intercambiarEstados**

- ```
  public java.lang.String intercambiarEstados(java.util.ArrayList<java.lang.String> estados,
                                              java.lang.String estado)
  ```

  Este método permite tomar un estado que se encuentra desordenado, y ordenarlo de la forma en que se define previamente, es decir, si existe un estado compuesto de más estados con un orden específico, los otros estados que contenga sus mismos estados, tambien van a estar con el mismo orden del estado previamente definido.

  **Parameters:**

  ```
  estados - Lista con los estados del autómata.

  estado - String con el estado que se quiere reordenar.
  ```

  **Returns:**

  ```
  String con el estado ordenado.
  ```

- **organizarAutomata**

- ```
  public java.util.ArrayList<java.util.ArrayList> organizarAutomata(java.util.ArrayList<java.lang.String> estados,

  java.util.ArrayList<java.util.ArrayList> automata)
  ```

  Toma el autómata que se construyó y reordena los estados de las transiciones,para que coincidan con los estados ya definidos, para evitar confusiones en el usuario,y sea más ordenado el proceso de reconocimiento.

  **Parameters:**

  ```
  estados - Lista con los estdos del autómata.

  automata - ArrayList con todas las transiciones del autómata.
  ```

  **Returns:**

  ```
  Un ArrayList con todas las transiciones ordenadas del autómata.
  ```

- **verificarHilera**

```
public boolean verificarHilera(java.lang.String hilera)
```

Este método permite que al tener un autómata ya definido, podamos relaizar un reconocimiento de secuencias mediante el autómata, para saber si la secuencia es válida o no, nos ubicamos en el estado inicial, luego miramos que simbolo entra y avanzamos al estado que indique la transición, hasta llegar al final de la hilera que es denotada por el símbolo '*', cuando se llega al final se mira en que tipo de estado se finalizó y si es de aceptación se acepta la secuencia, de lo contrario se rechaza.

**Parameters:**

hilera - String con la secuencia que se quiere verificar

**Returns:**

un booleano en true si la secuencia se acepta, o false de lo contrario.

- **estaEnAceptacion**

```
public boolean estaEnAceptacion(java.lang.String estado)
```

Este método determina si un estado es de aceptación o no

**Parameters:**

estado - el estado que se quiere verificar.

**Returns:**

boolean

- **transicionesParticion**

```
public java.util.ArrayList<java.util.ArrayList> transicionesParticion(java.uti
l.ArrayList<java.lang.String> particion)
```

Este método evalua la partición que se le ingresa por parámetro, y trae del autómata inicial las transiciones pertenecientes a cada elemento del Array.

**Parameters:**

particion - La partición que se quiere analizar

**Returns:**

Retorna todas las transciones de la particion que se entren por parámetro

- **estadosNoContenidos**

- ```
  public java.util.ArrayList<java.lang.String> estadosNoContenidos(java.util.Arr
  ayList<java.lang.String> particion,

  java.util.ArrayList<java.lang.String> transicionASym)
  ```

  Entra el array con la partición a tratar y otro Array con los símbolos a buscar dentro del primer Array, este arrojará el Array con los símbolos que no estan en esa partición.

  **Parameters:**

  particion - partición que se quiere analizar.

  transicionASym - los simbolos que se van a buscar

  **Returns:**

  ArrayList con los estados no contenidos en la partición.

- **creaNuevaParticion**

- ```
  public java.util.ArrayList<java.lang.String> creaNuevaParticion(java.util.Arra
  yList<java.lang.String> particion,
  ```

- 
```
java.util.ArrayList<java.lang.String> noContenidos,
                                                    int posSym)
```

  Este método permite crear una nueva partición para ello se mira la partición que se va a modificar, y la transición que hace que esta partición se tenga que modificar, luego miramos los estados de la transición correspondiente y si estos estados no estan contenidos en la partición se crea un nuevo array con la nueva partición

  **Parameters:**

  `particion - partición a analizar`

  `noContenidos - los estados no contenidos en la determinada partición`

  `posSym - la posición en el array que pertenece a el símbolo que se está manejando`

  **Returns:**

  `Un Array con la nueva partición creada.`

- **estadosExtraños**

```
public void estadosExtraños(int p)
```

  Este método permite tomar las transiciones actuales del autómata, y mira cuales estados estan contenidos en las transiciones para así ir llenando el vector de visitados e ir identificando que estados son extraños, finalmente actulizamos las transiciones sin estados extraños.

  **Parameters:**

  `p - ingresa la posición del estado desde el cual se va a empezar a analizar.`

- **actualizarAutomata**

```
public void actualizarAutomata()
```

  Este método luego de haber sacado los estados extraños, permite reestablecer todos los valores del autómata, actualiza los estados sin los estados que fueron extraidos, y los estados de aceptación y las particiones también los actualiza.

- **simplificar**

```
public void simplificar()
```

  Este método permite mirar que estados son equivalente y eliminar los estados extraños, para esto tomamos las particiones con los estados de aceptación y de rechazo, y vamos mirando sus transiciones cuando entra determinado simbolo, si sus transiciones van a una partición diferente se crea otra nueva partición con el estado que difiere y se agrega a un array de particiones, se sigue analizando cada partición hasta que terminemos todas las particiones con los diferentes símbolos, finalmente actualiza los valores del autómata: transiciones, estados,estados de Aceptación.

- **construirAutomata**

```
public void construirAutomata(java.util.ArrayList<java.util.ArrayList> particiones)
```

De acuerdo a las particiones que se tomaron del método simplificar, este método une las transiciones basandose en los estados que estan en cada partición, además actualiza las transiciones y finalemte actualiza el autómata, es decir, sus nuevos estados, estados de aceptación y transiciones.

**Parameters:**

`particiones - ingre el Array con todas las particiones obtenidas de los estados equivalentes.`

- **completarTransicion**

- `public java.util.ArrayList<java.lang.String> completarTransicion(java.util.ArrayList<java.lang.String> transicion,`
-

`java.lang.String[] estados,`

`java.util.ArrayList<java.lang.String> estados1)`

Este método mira el estado en cada transición, y si almenos contiene un caracter que pereteneza a a cualquiera de los nuevos estados, se reemplaza con el estado correspondiente

**Parameters:**

`transicion - Ingresa la transición que se va a analizar`

`estados - Ingresa un arreglo los estados actuales del autómata`

`estados1 - ingresa un array con los estados del autómata`

**Returns:**

`Un array con la nueva transición.`

- **copiaArray**

`public java.util.ArrayList<java.lang.String> copiaArray(java.util.ArrayList<java.lang.String> array)`

Este método crea una copia con todos los elementos de un Arraylist.

**Parameters:**

`array - elemento que se quiere clonar.`

**Returns:**

`un array con la copia del elemento.`

## En la clase DobleAutomata tenemos los siguientes métodos:

### El constructor de la clase:

**DobleAutomata**

```
public DobleAutomata(AutomataF a1,
                     AutomataF a2)
```

### Los métodos:

| Modifier and Type | Method and Description |
|---|---|
| void | contruirAutomata()<br>Este método permite juntar todas las transiciones, estados, estados de aceptación, estados iniciales de los dos autómatas que se les va a hacer la unión o la intersección, y crea un nuevo autómata nuevo con todos estos elementos. |
| AutomataF | getAf() |
| void | setAf(AutomataF af) |

### Una breve descripción:

- **contruirAutomata**

```
public void contruirAutomata()
```

Este método permite juntar todas las transiciones, estados, estados de aceptación, estados iniciales de los dos autómatas que se les va a hacer la unión o la intersección, y crea un nuevo autómata nuevo con todos estos elementos.

**En la clase Archivos tenemos los siguientes métodos:**

| Modifier and Type | Method and Description |
|---|---|
| void | **GuardarEnPDF**(**AutomataF** automata, javax.swing.table.DefaultTableModel dtm)<br>Este método recolecta toda la información pertinente para el usuario y para el sistema, la guarda en formato de impresión PDF en la ubicación seleccionada por el usuario. |
| void | **GuardarEnTXT**(**AutomataF** automata, javax.swing.table.DefaultTableModel dtm)<br>Este método recolecta toda la información pertinente para el usuario y para el sistema, la guarda en la ubicación seleccionada por el usuario de tal modo que pueda ser ingresado desde allí posteriormente. |

**Una breve descripción:**

- **GuardarEnTXT**

- `public void GuardarEnTXT(AutomataF automata,`
  `javax.swing.table.DefaultTableModel dtm)`

  Este método recolecta toda la información pertinente para el usuario y para el sistema, la guarda en la ubicación seleccionada por el usuario de tal modo que pueda ser ingresado desde allí posteriormente.

  **Parameters:**
  `automata - el elemento que se va a guardar.`

  `dtm - la tabla que se está tratando.`

- **GuardarEnPDF**

- `public void GuardarEnPDF(AutomataF automata,`
  `javax.swing.table.DefaultTableModel dtm)`

  Este método recolecta toda la información pertinente para el usuario y para el sistema, la guarda en formato de impresión PDF en la ubicación seleccionada por el usuario.

  **Parameters:**
  `automata - automta que se va a imprimir`

  `dtm - la tabla del autómata`

## Paquete modelo:

En la clase AutomataF tenemos la siguiente implementación:

All Methods | Instance Methods | Concrete Methods

| Modifier and Type | Method and Description |
|---|---|
| java.util.ArrayList<java.util.ArrayList> | getAutomataSinExtraños() |
| java.lang.String[] | getEstados() |
| java.lang.String[] | getEstadosAceptacion() |
| java.lang.String[] | getSimbolos() |
| java.util.ArrayList<java.util.ArrayList> | getTransiciones() |
| void | setAutomataSinExtraños(java.util.ArrayList<java.util.ArrayList> automataSinExtraños) |
| void | setEstados(java.lang.String[] estados) |
| void | setEstadosAceptacion(java.lang.String[] estadosAceptacion) |
| void | setSimbolos(java.lang.String[] simbolos) |
| void | setTransiciones(java.util.ArrayList<java.util.ArrayList> Transiciones) |

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Una breve descripción:**

### *Constructor Detail*

#### AutomataF

```
public AutomataF()
```

### *Method Detail*

#### getSimbolos

```
public java.lang.String[] getSimbolos()
```

#### setSimbolos

```
public void setSimbolos(java.lang.String[] simbolos)
```

#### getEstados

```
public java.lang.String[] getEstados()
```

#### setEstados

```
public void setEstados(java.lang.String[] estados)
```

#### getEstadosAceptacion

```
public java.lang.String[] getEstadosAceptacion()
```

#### setEstadosAceptacion

```
public void setEstadosAceptacion(java.lang.String[] estadosAceptacion)
```

#### getTransiciones

```
public java.util.ArrayList<java.util.ArrayList> getTransiciones()
```

- **setTransiciones**

```
public void setTransiciones(java.util.ArrayList<java.util.ArrayList> T
ransiciones)
```

- **getAutomataSinExtraños**

```
public java.util.ArrayList<java.util.ArrayList> getAutomataSinExtraños
()
```

- **setAutomataSinExtraños**

```
public void setAutomataSinExtraños(java.util.ArrayList<java.util.Array
List> automataSinExtraños)
```

## Paquete Vista:

### En la clase PantallaIngreso tenemos la siguiente implementación de métodos:

| Modifier and Type | Method and Description |
|---|---|
| AutomataF | automataSeleccionado()<br>Este método permite saber sobre cuál autómata vamos a trabajar |
| ControladorAutomata | controlSeleccionado() |
| boolean | definirEstadoAceptacion(java.lang.String estado)<br>Este método permite saber si un estado es de aceptación para llenar la tabla cuando se convierte de no determinístico a determinístico |
| void | llenarTabla(javax.swing.JTable tabla)<br>Este método permite mostrar el autómata en un tabla para que el usuario pueda interactuar dinámicamente con ella. |
| static void | main(java.lang.String[] args) |
| javax.swing.JTable | tablaSeleccionada()<br>Este método permite saber sobre cual tabla vamos a trabajar |

### Una breve descripción:

- **definirEstadoAceptacion**

```
public boolean definirEstadoAceptacion(java.lang.String estado)
```

Este método permite saber si un estado es de aceptación para llenar la tabla cuando se convierte de no determinístico a determinístico

**Parameters:**

estado - String con el estado que se quiere verificar.

**Returns:**

un booleano en true si el estado es de aceptación, o false de lo contrario.

- **llenarTabla**

```
public void llenarTabla(javax.swing.JTable tabla)
```

Este método permite mostrar el autómata en un tabla para que el usuario pueda interactuar dinámicamente con ella.

**Parameters:**

tabla - la tabla que se va a llenar.

- **tablaSeleccionada**

```
public javax.swing.JTable tablaSeleccionada()
```

Este método permite saber sobre cual tabla vamos a trabajar

**Returns:**

la tabla que se va a modificar

- **automataSeleccionado**

```
public AutomataF automataSeleccionado()
```

Este método permite saber sobre cuál autómata vamos a trabajar

**Returns:**

el autómata que se va a modificar o evaluar.