Tutorial AutomataTester

El siguiente documento pretende mostrar los pasos básicos que permiten agregar un nuevo autómata a la aplicación AutomataTester

Agregar un nuevo autómata no parametrizado

1. En primera instancia, cree una nueva clase en el paquete "uniandes.teolen.automatas.mundoAutomata", asociada al nuevo autómata no parametrizado

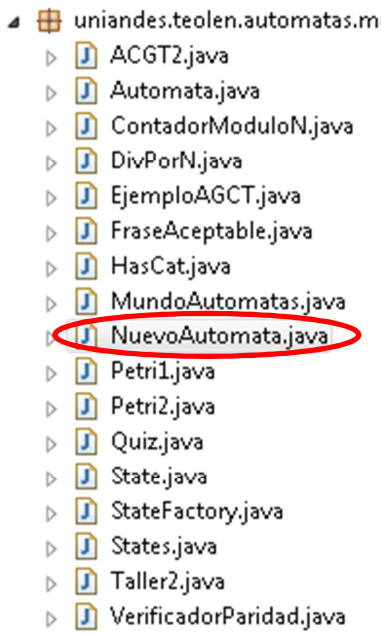

Figura 1. Creación de un nuevo autómata en el paquete uniandes.teolen.automatas.mundoAutomata

2. El siguiente paso consiste en definir el nuevo autómata. El paquete "uniandes.teolen.automatas.mundoAutomata" cuenta con una clase, "Automata.java", en la que se definen las características básicas de un autómata, esta clase servirá como base para la creación del nuevo autómata. De este modo, la clase asociada al nuevo autómata debe extender la clase "Automata"
3. Ahora se definirán los estados asociados al nuevo autómata. Para este tutorial, se construirá un autómata sencillo que reconozca aquellas cadenas formadas por a's y b's, tales que el número de b´s sea múltiplo de 3. Para ello se definen 3 estados: inicial_final (corresponde al estado inicial y final del autómata), intermedio1, e intermedio2 (ver Figura 2)

```
public class NuevoAutomata extends Automata {
    final State inicial_final = s(0);
    final State intermedio1 = s(1);
    final State intermedio2 = s(3);
```

Figura 2. Definición de los estados del nuevo autómata

4. En este paso se definirá el constructor asociado al nuevo autómata (ver Figura 3). Los elementos básicos de un autómata de estados finitos son los siguientes: un estado inicial (representado en este caso por la variable "myStartSate"), un conjunto finito de estados (representados por la variable "myStates"), un conjunto de estados finales (representados por el arreglo myFinalStates) y una función de transición (definida más adelante). Para el caso específico de este ejemplo, tanto el estado inicial, como el final corresponden al estado "inicial_final" definido previamente. De esta forma, la inicialización del autómata se realiza con los valores definidos anteriormente para el estado final, estado inicial y conjunto de estados. Observe que el conjunto finito de estados asociados al autómata no se define como un arreglo (a diferencia del conjunto de estados finales), en lugar de esto se emplea una clase especial, "States", utilizada por la clase Autómata para definir el conjunto finito de estados asociados a un autómata. El método "storeState", de la clase States, permite agregar nuevos estados al conjunto de estados del nuevo autómata

```
public NuevoAutomata()
{
    super();


    States myStates = new States() ;

    ArrayList <State> myFinalStates = new ArrayList <State> ();

    State myStartState;

    myStates.storeState(inicial_final);
    myStates.storeState(intermedio1);
    myStates.storeState(intermedio2);

    myFinalStates.add(inicial_final);
    myStartState = inicial_final;

    initializeAutomata(myStates,myStartState,myFinalStates);
}
```

Figura 3. Constructor asociado al nuevo autómata

5. En este paso se definirá la función de transición del autómata (ver Figura 4). En la Tabla 1 es mostrada la función delta asociada al autómata del ejemplo, de acuerdo a esta tabla fue construida la función mostrada en la Figura 4. Observe que, de acuerdo a la función definida en la Figura 4, si la cinta de entrada contiene símbolos diferentes a "a" o "b", se produce una excepción (puesto que el alfabeto del autómata corresponde al conjunto {a, b})

w (Simbolo de entrada)	q (Estado actual)	$\delta(q, w)$
a	Inicial_Final	Inicial_Final
b	Inicial_Final	Intermedio1
a	Intermedio1	Intermedio1
b	Intermedio1	Intermedio2
a	Intermedio2	Intermedio2
b	Intermedio2	Inicial_Final

Tabla 1. Función delta asociada al nuevo autómata

```
protected   State delta(State s, char c) throws Exception{

        State result = null;

        if (s == inicial_final){
          if (c == 'a')
              result =  inicial_final;
          else if (c == 'b')
              result =  intermedio1;
        }
        else if (s == intermedio1)
        {
          if (c == 'a')
                result = intermedio1;
          else if (c == 'b')
                result = intermedio2;
        }
        else if (s == intermedio2)
        {
          if (c == 'a')
                result = intermedio2;
          else if (c == 'b')
                result = inicial_final;
        }

        else {
            throw new Exception ( "Caracter indefinido para automata: "+c );
        }

        return result;
    }
```

Figura 4. Función de transición del nuevo autómata

6. Se definirá también una función de salida en las transiciones (ver Figura 5), la cual escribe en la cinta de salida del autómata la misma cadena de entrada, reemplazando las b's por x's (las a's permanecen iguales). Asimismo se define una método adicional, "toString", el cual determinará el nombre con el que aparecerá el autómata en la interfaz de la aplicación

```
protected String transitionOutput(State s, char c){

    if (c == 'a')
        return new String("a");
    else
        return new String("x");
   }

    public String toString() {
       return "Nuevo Automata";
   }
```

Figura 5. Función de salida del nuevo autómata

7. A continuación se define el nuevo autómata no parametrizado, en la clase MundoAutomatas, localizada en el paquete "uniandes.teolen.automatas.mundoAutomata" (ver Figura 6). Para hacer lo anterior, nos dirigimos al constructor de la clase y agregamos el nuevo autómata al arreglo "auto", definido en la clase MundoAutomatas (ver Figura 7)

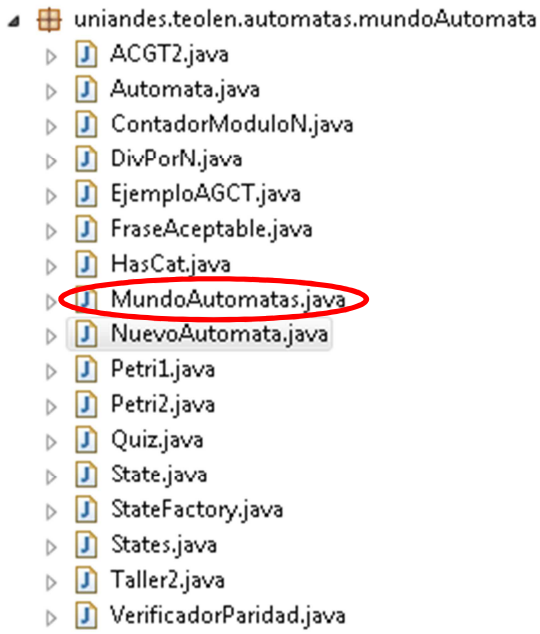

Figura 6. Clase MundoAutomatas

```
public  MundoAutomatas () {

    // Agreguen al final de esta lista los nuevos autómatas no parametrizados

    autos.add(new VerificadorParidad());
    autos.add(new ContadorModuloN(3));
    autos.add( new HasCat());
    autos.add( new Taller2());
    autos.add( new Petri1());
    autos.add( new Petri2());
    autos.add(  new Quiz());
    autos.add( new EjemploAGCT());
    autos.add(new NuevoAutomata());

    // Agreguen al final de esta lista los nombres de los nuevos autómatas parametrizados

    autosP.add("Contador Modulo N");
    autosP.add("Dividir por  N");
    autosP.add("Cifrar por N");

    currentAuto =  autos.get(0);

}
```

Figura 7. Modificación del constructor de la clase MundoAutomatas

8. De esta manera, el nuevo autómata queda ubicado en el menú "Automata" (ver Figura 8)

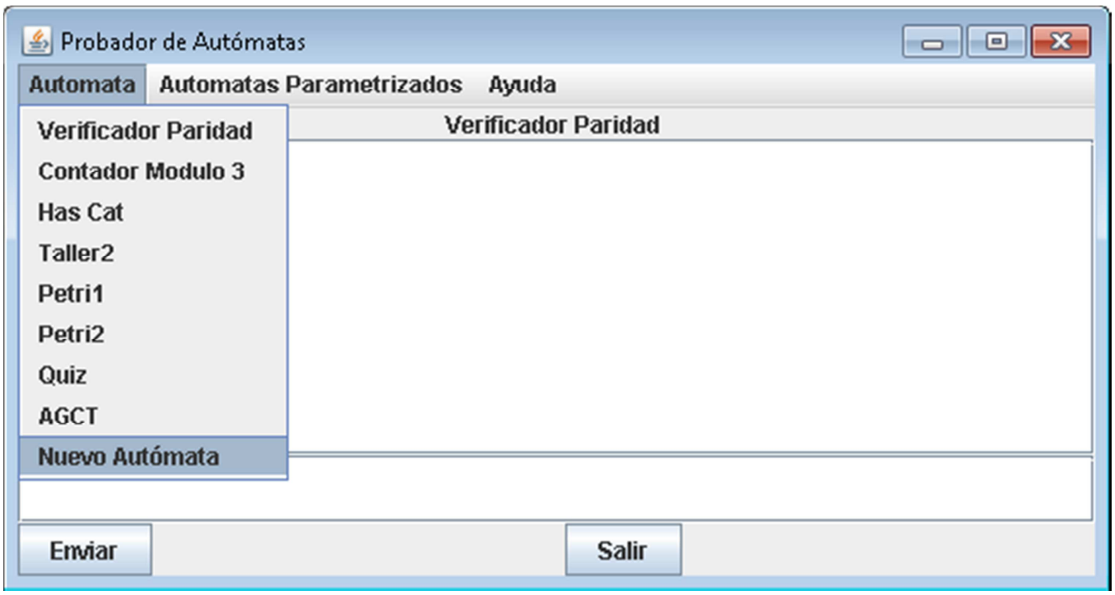

Figura 8. Ubicación del nuevo autómata en la interfaz

Agregar un nuevo autómata parametrizado

Un autómata parametrizado, en el contexto de esta aplicación, es un autómata cuya función de transición depende de un parámetro definido por el usuario; por lo que, este tipo de autómatas reciben en su constructor este parámetro, lo cual hace indispensable contar con este dato antes de crear una nueva instancia de este

tipo de autómatas. Para agregar un nuevo autómata parametrizado a la aplicación se siguen los pasos 1 a 7 de la anterior sección (que serán resumidos a continuación), con algunas adiciones:

1. En primera instancia, cree una nueva clase en el paquete "uniandes.teolen.automatas.mundoAutomata", asociada al nuevo autómata parametrizado (ver Figura 9)

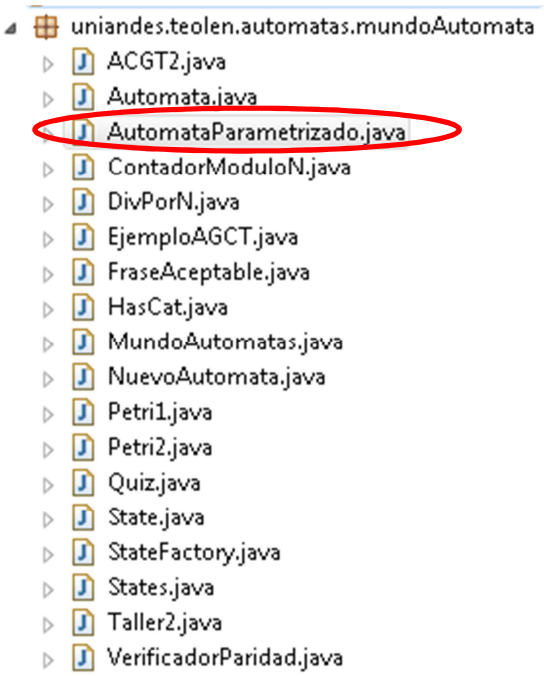

Figura 9. Creación de un nuevo autómata en el paquete uniandes.teolen.automatas.mundoAutomata

2. Se definen los estados, el constructor (para este tipo de autómatas se agrega un atributo adicional: n, correspondiente al dato de entrada del autómata) y la función de transición del nuevo autómata parametrizado (ver Figura 10, Figura 11, Figura 12, Figura 13)

```
public class AutomataParametrizado extends Automata {
private int n;
```

Figura 10. Definición del atributo asociado al nuevo autómata parametrizado, en donde se almacenará el dato introducido por el usuario (que en este caso corresponde a un entero)

```
public AutomataParametrizado(int  n)
{
    super();
    this.n = n;
    States myStates = new States() ;

    ArrayList <State> myFinalStates = new ArrayList <State> ();

    State myStartState;

    int i;
    for (i=0; i<n; i++){
        myStates.storeState(s(i));
    }
    myFinalStates.add(s(0));
    myStartState = s(0);
    initializeAutomata(myStates,myStartState,myFinalStates);
}
```

Figura 11. Constructor asociado al nuevo autómata parametrizado

```
protected State delta(State st, char c) throws Exception{

  State resp;
  int stVal = (Integer) st.id();

  if (c == 'a') {
      resp =  s((stVal+1)%n);
  }
  else if (c == 'b'){
      resp = st;
  }
  else
      throw new Exception("Error: caracter no permitido - "+c);

  return resp;
}
```

Figura 12. Función de transición asociada al nuevo autómata parametrizado

```
public String toString() {
        return "Automata Parametrizado, entrada: "+n;
    }
```

Figura 13. Función encargada de definir el nombre que se asignará al nuevo autómata parametrizado en la interfaz de la aplicación

3. A continuación se define el nuevo autómata no parametrizado, en la clase MundoAutomatas. Para hacer lo cual, nos dirigimos al constructor de la clase y agregamos una nueva cadena de caracteres, con el nombre del nuevo autómata parametrizado, al arreglo “autoP”, definido en la clase MundoAutomatas (ver Figura 14)

```
public  MundoAutomatas () {

    // Agreguen al final de esta lista los nuevos autómatas no parametrizados

    autos.add(new VerificadorParidad());
    autos.add(new ContadorModuloN(3));
    autos.add( new HasCat());
    autos.add( new Taller2());
    autos.add( new Petri1());
    autos.add( new Petri2());
    autos.add(  new Quiz());
    autos.add( new EjemploAGCT());
    autos.add(new NuevoAutomata());

    // Agreguen al final de esta lista los nombres de los nuevos autómatas parametrizados

    autosP.add("Contador Modulo N");
    autosP.add("Dividir por  N");
    autosP.add("Cifrar por N");
    autosP.add("Automata Parametrizado");

    currentAuto =  autos.get(0);

}
```

Figura 14. Modificación del constructor de la clase MundoAutomatas

4. Puesto que el constructor de este tipo de autómatas tiene como entrada el parámetro introducido por el usuario, su instanciación es un poco diferente a la de los autómatas no parametrizados. Es necesario contar con una función en MundoAutomatas, encargada de hacer la instanciación de este tipo autómatas **una vez se cuente con el parámetro introducido por el usuario**; la función encargada de realizar lo anterior en la clase MundoAutomatas, se denomina "getAutoP(int a, int p)". En esta función se debe agregar un nuevo case, correspondiente al nuevo autómata parametrizado (ver Figura 15)

```
public Automata getAutoP(int a, int p) {

    // Agregue en el orden de los nombres  el llamado al constructor del autómata correspondiente

    switch (a)  {
      case 0: return new ContadorModuloN(p);
      case 1: return new DivPorN(p);
      case 2: return new FraseAceptable(p);
      case 3: return new AutomataParametrizado(p);
    }

    return currentAuto;

}
```

Figura 15. Modificación de la función encargada de hacer la instanciación de los autómatas parametrizados, una vez se cuenta con el parámetro introducido por el usuario

5. De esta forma, se crea un nuevo autómata parametrizado en la aplicación, el cual estará ubicado en el menú "Autómatas Parametrizados"

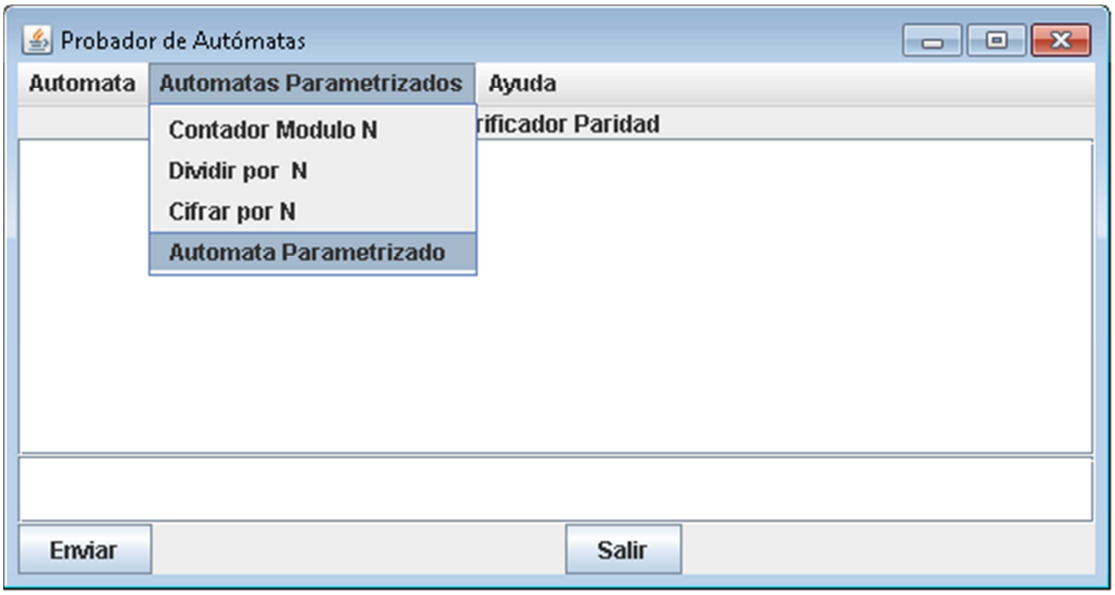

Figura 16. Ubicación del nuevo autómata parametrizado en la interfaz

Agregar un nuevo autómata de pila

1. En primera instancia, cree una nueva clase en el paquete "uniandes.teolen.automatas.mundoAutomata", asociada al nuevo autómata de pila (ver Figura 17)

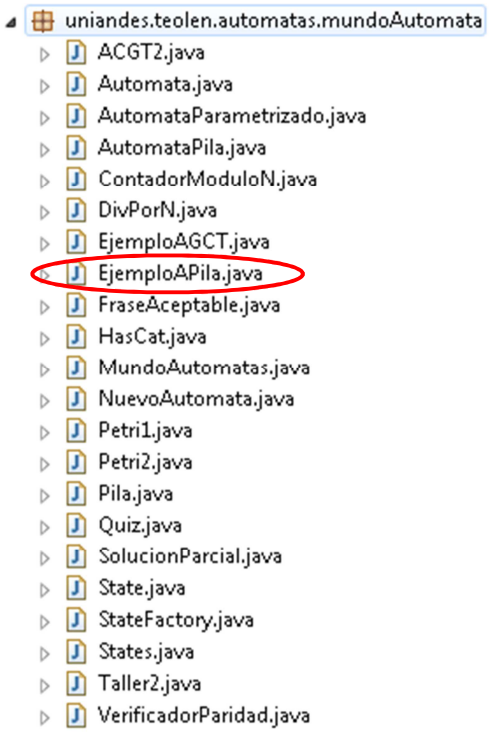

Figura 17. Creación de un nuevo autómata en el paquete uniandes.teolen.automatas.mundoAutomata

2. El siguiente paso consiste en definir el nuevo autómata de pila. El paquete "uniandes.teolen.automatas.mundoAutomata" cuenta con una clase, "AutomataPila.java", en la que se definen las características básicas de un autómata de pila, esta clase servirá como base para la creación del nuevo autómata. De este modo, la clase asociada al nuevo autómata debe extender la clase "AutomataPila"
3. Ahora se definirán los estados asociados al nuevo autómata. Para este tutorial, se construirá el siguiente autómata determinístico: M=({1,2},{a, b}, {a},1, {1,2}, {((1,λ ,a),(1,a)), ((1,a,b),(2, λ)), ((2,a,b),(2, λ))}) que reconoce $\{a^n b^n : n \geq 0\}$. Para ello se definen 3 estados: estado1 y estado 2 (ver Figura 18)

```
public class EjemploAPila extends AutomataPila {

    final State estado1 = s(0);
    final State estado2 = s(1);
```

Figura 18. Definición de los estados del nuevo autómata de pila

4. En este paso se definirá el constructor asociado al nuevo autómata, en donde son definidos los estados del autómata, su estado inicial y sus estados finales (ver Figura 19)

```
public EjemploAPila(){
    super();

    States myStates = new States();

    ArrayList<State> myFinalStates = new ArrayList<State> ();

    State myStartState;

    myStates.storeState(estado1);
    myStates.storeState(estado2);

    myFinalStates.add(estado1);
    myFinalStates.add(estado2);
    myStartState = estado1;

    initializeAutomataPila(myStates, myStartState, myFinalStates);
}
```

Figura 19. Constructor asociado al nuevo autómata de pila

5. En este paso se definirá la función de transición del autómata utilizado como ejemplo, definido en el paso 3 (ver Figura 20). Observe que para realizar modificaciones sobre la pila, se emplean las funciones: changeTop(), pop(), pushOn() y push(). No son definidas las funciones skip() o ignore() puesto que no realizan ningún cambio sobre la pila, en estos casos, simplemente no se invoca ninguna de las funciones que se encuentran definidas en la plantilla

```
protected State stateTransitionRelation(State s, char stackTop, char tapeSymbol) throws Exception{

    State result = null;

    if(s == estado1){
        if((stackTop == 'a')&&(tapeSymbol == 'b')){
            this.pop();//Desempile el símbolo que se encuentra en el tope de la pila y no empile nada
            result = estado2;
        }
        else if(tapeSymbol == 'a'){
            this.push('a');//Sin importar lo que haya en la pila empile a
            result = estado1;
        }
        else{
            throw new Exception("Simbolo indefinido para el autómata de pila" + tapeSymbol);
        }
    }
    else if(s == estado2){
        if((stackTop == 'a')&&(tapeSymbol == 'b')){
            this.pop();//Desempile el símbolo que se encuentra en el tope de la pila y no empile nada
            result = estado2;
        }
        else{
            throw new Exception("Símbolo indefinido para el autómata de pila" + tapeSymbol);
        }
    }
    else
        throw new Exception("El estado se encuentra indefinido para el autómta de pila" + tapeSymbol);

    return result;
}
```

Figura 20. Función de transición del nuevo autómata

6. A continuación se define el nuevo autómata de pila, en la clase MundoAutomatas, localizada en el paquete “uniandes.teolen.automatas.mundoAutomata” (ver Figura 21). Para hacer lo anterior, nos dirigimos al constructor de la clase y agregamos el nuevo autómata al arreglo “autosPila”, definido en la clase MundoAutomatas (ver Figura 22)

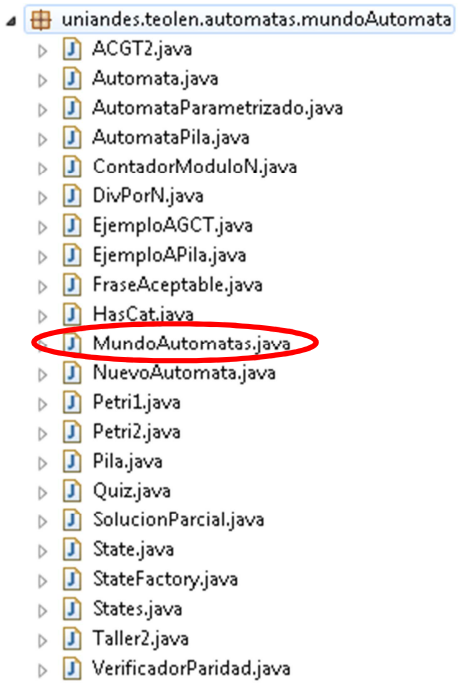

Figura 21. Clase MundoAutomatas

```
public  MundoAutomatas () {

    // Agreguen al final de esta lista los nuevos autómatas no parametrizados

    autos.add(new VerificadorParidad());
    autos.add(new ContadorModuloN(3));
    autos.add( new HasCat());
    autos.add( new Taller2());
    autos.add( new Petri1());
    autos.add( new Petri2());
    autos.add(  new Quiz());
    autos.add( new EjemploAGCT());
    autos.add(new NuevoAutomata());

    // Agreguen al final de esta lista los nombres de los nuevos autómatas parametrizados

    autosP.add("Contador Modulo N");
    autosP.add("Dividir por  N");
    autosP.add("Cifrar por N");
    autosP.add("Automata Parametrizado");

    //Agregue al final de esta lista los nuevos autómtas de pila
    autosPila.add(new EjemploAPila());
    autosPila.add(new SolucionParcial());

    currentAuto =  autos.get(0);

    currentAutoPila = autosPila.get(0);

    tipoAutomata = "Auto";
}
```

Figura 22. Modificación del constructor de la clase MundoAutomatas

7. De esta manera, el nuevo autómata queda ubicado en el menú “Autómatas de Pila” (ver Figura 8)

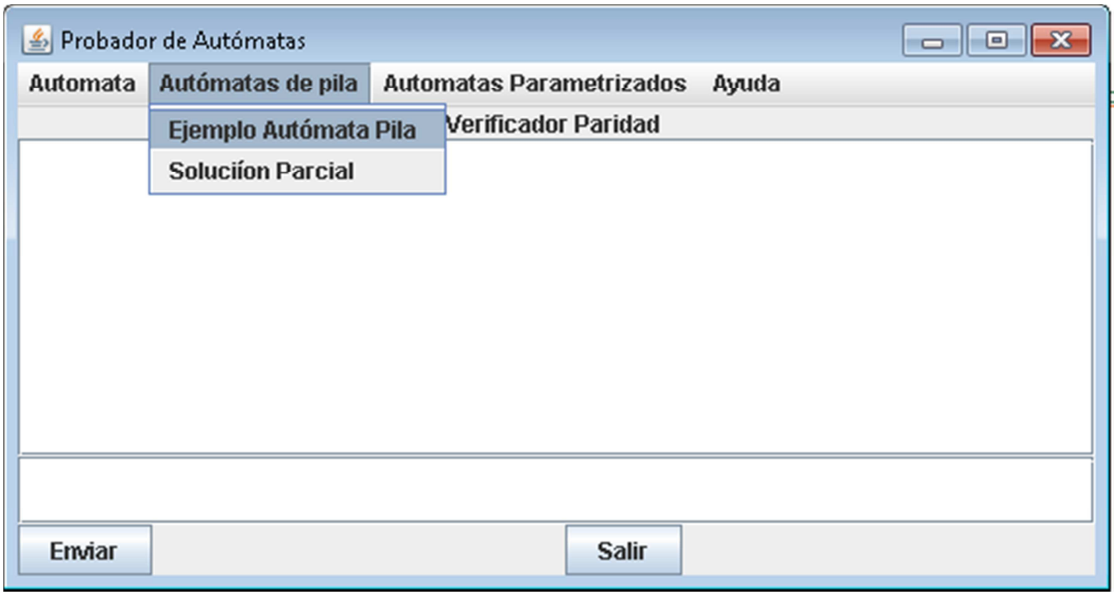

Figura 23. Ubicación del nuevo autómata de pila en la interfaz