Plan de Pruebas de software

Ingeniería de Software – 3ra Evaluación

Instituto Tecnológico de Celaya

Santana De Los Ríos Carolina

Tabla de contenido

[Instrucciones de examen 3](#_Toc451965496)

[Historial de Versiones 3](#_Toc451965497)

[Información del Proyecto 3](#_Toc451965498)

[Resumen 4](#_Toc451965499)

[Alcance de las pruebas 4](#_Toc451965500)

[Elementos de Pruebas 4](#_Toc451965501)

[Clase: Identifier 4](#_Toc451965502)

[Clase: Automaton 4](#_Toc451965503)

[Pruebas de regresión 5](#_Toc451965504)

[Funcionalidades a No Probar 5](#_Toc451965505)

[Enfoque de pruebas 5](#_Toc451965506)

[Criterios de Aceptación o Rechazo 6](#_Toc451965507)

[Criterios de Aceptación o Rechazo 6](#_Toc451965508)

[Criterios de suspención 6](#_Toc451965509)

[Criterios de Reanudación 6](#_Toc451965510)

[Entregables 6](#_Toc451965511)

[Recursos 6](#_Toc451965512)

[Requerimientos de Entornos - Hardware 6](#_Toc451965513)

[Requerimientos de Entornos - Software 6](#_Toc451965514)

[Herramientas de Pruebas Requeridas 7](#_Toc451965515)

[Planificación y Organización 7](#_Toc451965516)

[Procedimientos para las pruebas 7](#_Toc451965517)

[Premisas 7](#_Toc451965518)

[Dependencias y riesgos 7](#_Toc451965519)

[Código del programa 7](#_Toc451965520)

[Clase: Identifier 7](#_Toc451965521)

[Clase: Automaton 11](#_Toc451965522)

[Resultados JUnit 20](#_Toc451965523)

[Clase: Automaton 20](#_Toc451965524)

[Clase: Identifier 20](#_Toc451965525)

[Correcciones y cambios realizados 21](#_Toc451965526)

[Clase: Automaton 21](#_Toc451965527)

[Clase: Identifier 21](#_Toc451965528)

[Código corregido 21](#_Toc451965529)

[Clase: Automaton 21](#_Toc451965530)

[Clase: Identifier 23](#_Toc451965531)

[Conclusión 26](#_Toc451965532)

[Referencias 26](#_Toc451965533)

[Glosario 26](#_Toc451965534)

# Instrucciones de examen

**Tecnológico Nacional de México**

**Instituto Tecnológico de Celaya**

**Examen Ingeniería de Software 3ra evaluación Fecha 13/05/2016**

Paso 1) Busque una aplicación que haya desarrollado en otra materia en Java que tenga por lo menos 500 líneas de código, sin contar líneas en blanco y comentarios.

Paso 2) Baje e instale JUnit para Netbeans o Eclipse según el ambiente de desarrollo que tú prefieras.

Paso 3) Realice las pruebas unitarias de su software y entregue un reporte que incluya los siguientes puntos:

1. Plan de pruebas incluyendo las categorías de pruebas unitarias necesarias para la aplicación.
2. Pruebas y resultados que arroje JUnit para definir correcciones y vistos buenos de los métodos probados.
3. Resumen de correcciones y cambios aplicados al código fuente.
4. Códigos corregidos con documentación explícita dentro del código fuente.

**Este reporte se entrega en equipos de máximo 2 integrantes (de 3 no se acepta)**

**Fecha de entrega: viernes 20 de mayo 2016 en conacad**

# Historial de Versiones

|  |  |  |
| --- | --- | --- |
| Fecha | Versión | Descripción |
| 2016-05-16 | 1.0 | Primer uso de la librería JUnit |
| 2016-05-17 | 1.1 | Creación de más y mejores pruebas |
| 2016-05-18 | 1.2 | Cambio de clases y ejecución de pruebas sobre la misma |

# Información del Proyecto

|  |  |
| --- | --- |
| Proyecto | Analizador Léxico |
| Fecha de preparación | 2016-05-16 |
| Encargado del proyecto | Santana De Los Ríos Carolina |

# Resumen

El proyecto “Analizador Léxico”, como su nombre lo indica, se trata de un programa que simula la etapa inicial de un compilador cuando éste recibe un código fuente.

El programa también incluye un “Analizador Sintáctico”, sin embargo, con solo el Léxico se logra alcanzar el objetivo del examen de la evaluación 3. Además, la parte correspondiente al análisis de sintaxis contiene un código bastante extenso que fácilmente supera el rango de número de líneas impuestas en el examen.

El proceso de análisis léxico consta de las siguientes clases:

* PrincipalWindow
* LexicalAnalysis
* Automaton
* TDA\_Token
* Identifier

# Alcance de las pruebas

## Elementos de Pruebas

### Clase: Identifier

Se analizarán las funciones:

* getIdComments
* getIdConstants
* getIdDataTypes
* getIdDelimiters
* getIdOperators
* getIdReservedWords
* getIdVariables

### Clase: Automaton

* automatonComments
* automatonConstants
* automatonDataTypes
* automatonDelimiters
* automatonOperators
* automatonReservedWords
* automatonVariables
* isLetter
* isNumber

Todas estas funciones regresan como resultado un valor booleano dependiendo de si el análisis fue completado con éxito o si surgió algún error durante el proceso.

## Pruebas de regresión

Las clases dentro del alcance de prueba afectan a las clases:

* Identifier
* TDA\_TokenData
* PrincipalWindow
* GrapSymbolTable

Sin embargo, aunque estas clases no están dentro del alcance de prueba, los métodos analizados dentro de ésta afectan a los métodos de estas clases.

## Funcionalidades a No Probar

Las clases que no se toman en cuenta en absoluto para estas pruebas son:

* syntacticAnalysis
* syntacticGrammar

Esto es, debido a que con el análisis léxico se cumplen los requerimientos del examen. Debido a esto, no es necesario abordar estas clases.

## Enfoque de pruebas

Para este trabajo se realizarán pruebas unitarias.

Las pruebas unitarias son pruebas por cada unidad, en este caso una unidad es equivalente a un requerimiento. El requerimiento es aprobado y aprobado si este cumple con lo que está escrito en la especificación de requerimientos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NOMBRE: | Pruebas Unitarias | | IDENTIFICADOR: | UT01 |
| ACTIVIDADES | | Análisis de requerimientos del sistema | | |
| TIEMPO ESTIMADO | | 45 – 60 minutos por unidad | | |
| MÉTODOS O HERRAMIENTAS | | Netbeans junto con la librería JUnit | | |
| ENTREGABLES | | Plan de Pruebas | | |

# Criterios de Aceptación o Rechazo

## Criterios de Aceptación o Rechazo

Los criterios a utilizar para este trabajo sobre el proyecto de analizador léxico son:

* Terminar de implementar y documentar todas las pruebas unitarias
* Obtener un porcentaje de éxito, en cada prueba, mayor o igual a 85%
* Asegurarse de que cada prueba cubre los componentes más importantes del programa
* Las áreas analizadas deben juntar igual o más de 500 líneas de código

## Criterios de suspención

Se detendrán las pruebas en caso de que se presente alguna de las siguientes condiciones:

* Imposible ejecutar más de tres casos de pruebas
* Porcentaje de pruebas fallidas mayor a 50%

## Criterios de Reanudación

Si las pruebas fueron suspendidas, debe verificar si puede reanudarlas mediante el cumplimiento de alguna de las siguientes funciones:

* Modificación del código que, consecuentemente, modifica los resultados de la prueba

# Entregables

Básicamente solo será un entregable: Documento de Plan de Pruebas. Sin embargo, éste contendrá elementos muy importantes como lo son:

* Contenido original del Plan de Pruebas
* Pruebas y resultados arrojados por JUnit para definir correcciones y vistos buenos de los métodos aprobados
* Resumen de correcciones y cambios aplicados al código fuente
* Códigos corregidos con documentación explícita dentro del código fuente.

# Recursos

## Requerimientos de Entornos - Hardware

* Equipo de cómputo (Laptop | Computadora de Escritorio)

## Requerimientos de Entornos - Software

* SO Windows | Linux | IOS
* Netbeans (versión mayor a la 6.0)

## Herramientas de Pruebas Requeridas

Para poder realizar las pruebas Unitarias se hace uso de las librerías proporcionadas por JUnit.

# Planificación y Organización

## Procedimientos para las pruebas

Se creará la estructura del contenido de las pruebas Unitarias para cada módulo o función dentro del programa.

Se ejecutará cada prueba y se documentará su resultado, así como su correspondiente modificación de código en caso de algún fallo.

## Premisas

La premisa más grande y la que más promete causar inconvenientes es el hecho de que éste no es el único proyecto o trabajo a desarrollar, es decir, existen otros proyectos por realizar y se tiene que cronometrar el trabajo para cada uno.

## Dependencias y riesgos

* Dependencias con otros proyectos
* Restricciones de tiempo

Estos puntos a tener en cuenta están clasificados según su probabilidad de impacto.

# Código del programa

## Clase: Identifier

package TDA**;**

/\*\*

\*

\* @author Santana De Los Ríos Carolina

\* @author Yáñez Martínez Ramón Emmanuel

\*

\*/

public class Identifier **{**

/\*\*

\* Analiza un token y determina si es una Constante valida

\*

\* @param token Es un token del lenguaje

\* @return Regresa un arreglo bidimensional de enteros con el tipo y la

\* clave del token en caso de pertenecer al lenguaje- (Ejemplo new

\* int[][]{{5}, {1}}).En caso contrario regresa null.

\*

\*/

public int**[][]** getIdConstants**(**String token**)** **{**

**if** **(**token**.**startsWith**(**"\""**))** **{**

**if** **(**token**.**endsWith**(**"\""**))** **{**

**return** **new** int**[][]{{**7**},** **{**1**}};**

**}**

**}** **else** **if** **(!**token**.**startsWith**(**"+"**)** **&&** **!**token**.**startsWith**(**"-"**)** **&&** **!**isNumber**(**token**.**charAt**(**0**)))** **{**

**return** **null;**

**}**

**return** **new** int**[][]{{**7**},** **{**2**}};**

**}**

/\*\*

\* Analiza un token y determina si es una variable

\*

\* @param token Es un token del lenguaje

\* @return Regresa un arreglo bidimensional de enteros con el tipo y la

\* clave del token en caso de pertenecer al lenguaje- (Ejemplo new

\* int[][]{{6}, {1}}).En caso contrario regresa null.

\*

\*/

public int**[][]** getIdVariables**(**String token**)** **{**

**return** **new** int**[][]{{**6**},** **{**1**}};**

**}**

/\*\*

\* Analiza un token de tipo delimitador y obtiene su Tipo e Identificador

\*

\* @param token Es un token del lenguaje

\* @return Regresa un arreglo bidimensional de enteros con el tipo y la

\* clave del token en caso de pertenecer al lenguaje- (Ejemplo new

\* int[][]{{5}, {1}}).En caso contrario regresa null.

\*

\*/

public int**[][]** getIdDelimiters**(**String token**)** **{**

**switch** **(**token**.**charAt**(**0**))** **{**

**case** '{'**:**

**return** **new** int**[][]{{**12**},** **{**1**}};**

**case** '}'**:**

**return** **new** int**[][]{{**12**},** **{**2**}};**

**case** '('**:**

**return** **new** int**[][]{{**12**},** **{**3**}};**

**case** ')'**:**

**return** **new** int**[][]{{**12**},** **{**4**}};**

**case** ';'**:**

**return** **new** int**[][]{{**12**},** **{**5**}};**

**default:**

**return** **null;**

**}**

**}**

/\*\*

\* Analiza un token de tipo Operador y obtiene su Tipo e Identificador

\*

\* @param token Es un token del lenguaje

\* @return Regresa un arreglo bidimensional de enteros con el tipo y la

\* clave del token en caso de pertenecer al lenguaje- (Ejemplo new

\* int[][]{{5}, {1}}).En caso contrario regresa null.

\*

\*/

public int**[][]** getIdOperators**(**String token**)** **{**

**if** **(**token**.**length**()** **==** 1 **&&** token**.**startsWith**(**"+"**))** **{**

**return** **new** int**[][]{{**8**},** **{**1**}};**

**}** **else** **if** **(**token**.**length**()** **==** 1 **&&** token**.**startsWith**(**"-"**))** **{**

**return** **new** int**[][]{{**8**},** **{**2**}};**

**}** **else** **if** **(**token**.**length**()** **==** 1 **&&** token**.**startsWith**(**"\*"**))** **{**

**return** **new** int**[][]{{**8**},** **{**3**}};**

**}** **else** **if** **(**token**.**length**()** **==** 1 **&&** token**.**startsWith**(**"/"**))** **{**

**return** **new** int**[][]{{**8**},** **{**4**}};**

**}** **else** **if** **(**token**.**charAt**(**0**)** **==** '>'**)** **{**

**return** **new** int**[][]{{**9**},** **{**1**}};**

**}** **else** **if** **(**token**.**charAt**(**0**)** **==** '<'**)** **{**

**return** **new** int**[][]{{**9**},** **{**2**}};**

**}** **else** **if** **(**token**.**charAt**(**0**)** **==** '='**)** **{**

**if** **(**token**.**length**()** **==** 1**)** **{**

**return** **new** int**[][]{{**11**},** **{**1**}};**

**}** **else** **if** **(**token**.**length**()** **==** 2 **&&** token**.**charAt**(**1**)** **==** '='**)** **{**

**return** **new** int**[][]{{**9**},** **{**3**}};**

**}**

**}** **else** **if** **(**token**.**charAt**(**0**)** **==** '!'**)** **{**

**if** **(**token**.**length**()** **==** 1**)** **{**

**return** **new** int**[][]{{**10**},** **{**3**}};**

**}** **else** **if** **(**token**.**length**()** **==** 2 **&&** token**.**charAt**(**1**)** **==** '='**)** **{**

**return** **new** int**[][]{{**9**},** **{**4**}};**

**}**

**}** **else** **if** **(**token**.**charAt**(**0**)** **==** '&'**)** **{**

**return** **new** int**[][]{{**10**},** **{**1**}};**

**}** **else** **if** **(**token**.**charAt**(**0**)** **==** '|'**)** **{**

**return** **new** int**[][]{{**10**},** **{**2**}};**

**}**

**return** **null;**

**}**

/\*\*

\* Analiza un token de tipo palabra reservada y obtiene su Tipo e

\* Identificador

\*

\* @param token Es un token del lenguaje

\* @return Regresa un arreglo bidimensional de enteros con el tipo y la

\* clave del token en caso de pertenecer al lenguaje- (Ejemplo new

\* int[][]{{5}, {1}}).En caso contrario regresa null.

\*

\*/

public int**[][]** getIdReservedWords**(**String token**)** **{**

**switch** **(**token**)** **{**

**case** "while"**:**

**return** **new** int**[][]{{**1**},** **{**1**}};**

**case** "if"**:**

**return** **new** int**[][]{{**1**},** **{**2**}};**

**case** "break"**:**

**return** **new** int**[][]{{**2**},** **{**1**}};**

**case** "null"**:**

**return** **new** int**[][]{{**3**},** **{**1**}};**

**case** "print"**:**

**return** **new** int**[][]{{**4**},** **{**1**}};**

**default:**

**return** **null;**

**}**

**}**

/\*\*

\* Analiza un token de tipo de dato y obtiene su Tipo e Identificador

\*

\* @param token Es un token del lenguaje

\* @return Regresa un arreglo bidimensional de enteros con el tipo y la

\* clave del token en caso de pertenecer al lenguaje- (Ejemplo new

\* int[][]{{5}, {1}}).En caso contrario regresa null.

\*

\*/

public int**[][]** getIdDataTypes**(**String token**)** **{**

**switch** **(**token**)** **{**

**case** "int"**:**

**return** **new** int**[][]{{**5**},** **{**1**}};**

**case** "str"**:**

**return** **new** int**[][]{{**5**},** **{**2**}};**

**default:**

**return** **null;**

**}**

**}**

/\*\*

\* Analiza un token de tipo comentario y obtiene su Tipo e Identificador

\*

\* @param token Es un token del lenguaje

\* @return Regresa un arreglo bidimensional de enteros con el tipo y la

\* clave del token en caso de pertenecer al lenguaje- (Ejemplo new

\* int[][]{{5}, {1}}).En caso contrario regresa null.

\*

\*/

public int**[][]** getIdComments**(**String token**)** **{**

**return** **new** int**[][]{{**13**},** **{**1**}};**

**}**

/\*\*

\* Analiza un símbolo y determina si es un número. Es un número si el

\* símbolo esta entre el 0-9

\*

\* @param character

\* @return Regresa true si es un número y false en caso contrario

\*/

private boolean isNumber**(**char character**)** **{**

**return** character **>=** '0' **&&** character **<=** '9'**;**

**}**

/\*\*

\* Analiza un símbolo y determina si es una letra. Es una letra si el

\* símbolo se encuentra entre a-z o A-Z

\*

\* @param character

\* @return Regresa true si es una letra y false en caso contrario

\*/

private boolean isLetter**(**char character**)** **{**

**return** character **>=** 'a' **&&** character **<=** 'z' **||** character **>=** 'A' **&&** character **<=** 'Z'**;**

**}**

**}**

## Clase: Automaton

package Lexic**;**

/\*\*

\*

\* @author Santana De Los Ríos Carolina

\* @author Yáñez Martínez Ramón Emmanuel

\*

\*/

public class Automaton **{**

int idError**;**

public Automaton**()** **{**

idError **=** 0**;**

**}**

/\*\*

\* Método que emula el funcionamiento de un autómata finito, analiza un

\* token para determinar si es una constante

\*

\* @param token Es un token del lenguaje

\* @return Regresa un booleano indicando si el token es una constante siendo

\* el booleano verdadero si es una constante y falso en caso contrario.

\*/

public boolean automatonConstants**(**String token**)** **{**

int state **=** 0**;**

**for** **(**int i **=** 0**;** i **<** token**.**length**();** i**++)** **{**

**switch** **(**state**)** **{**

**case** 0**:**

**switch** **(**token**.**charAt**(**i**))** **{**

**case** '"'**:**

state **=** 3**;**

**break;**

**case** '+'**:**

**case** '-'**:**

state **=** 1**;**

**break;**

**default:**

state **=** 1**;**

i**--;**

**break;**

**}**

**break;**

**case** 1**:**

**if** **(**isNumber**(**token**.**charAt**(**i**)))** **{**

state **=** 2**;**

**}** **else** **{**

idError **=** 12**;**

**return** **false;**

**}**

**break;**

**case** 2**:**

**if** **(**isNumber**(**token**.**charAt**(**i**)))** **{**

**}** **else** **{**

idError **=** 13**;**

**return** **false;**

**}**

**break;**

**case** 3**:**

**if** **(**token**.**charAt**(**i**)** **==** '"'**)** **{**

state **=** 4**;**

**}** **else** **if** **(**Character**.**isDefined**(**token**.**charAt**(**i**)))** **{**

**}** **else** **{**

idError **=** 14**;**

**return** **false;**

**}**

**break;**

**default:**

idError **=** 15**;**

**return** **false;**

**}**

**}**

**if** **(**state **==** 4 **||** state **==** 2**)** **{**

**return** **true;**

**}** **else** **{**

idError **=** 16**;**

**return** **false;**

**}**

**}**

/\*\*

\* Método que emula el funcionamiento de un autómata finito, analiza un

\* token para determinar si es una variable

\*

\* @param token Es un token del lenguaje

\* @return Regresa un booleano indicando si el token es una variable siendo

\* el booleano verdadero si es un variable y falso en caso contrario.

\*/

public boolean automatonVariables**(**String token**)** **{**

int state **=** 0**;**

**for** **(**int i **=** 0**;** i **<** token**.**length**();** i**++)** **{**

**switch** **(**state**)** **{**

**case** 0**:**

**if** **(**'$' **==** token**.**charAt**(**i**))** **{**

state **=** 1**;**

**}** **else** **{**

idError **=** 1**;**

**return** **false;**

**}**

**break;**

**case** 1**:**

**if** **(**isNumber**((**char**)** token**.**charAt**(**i**)))** **{**

**}** **else** **if** **(**isLetter**((**char**)** token**.**charAt**(**i**)))** **{**

**}** **else** **if** **(**token**.**charAt**(**i**)** **==** '\_'**)** **{**

**}** **else** **if** **(**token**.**charAt**(**i**)** **==** '?'**)** **{**

**}** **else** **{**

idError **=** 2**;**

**return** **false;**

**}**

**break;**

**default:**

idError **=** 3**;** /\*No codificado\*/

**return** **false;**

**}**

**}**

**return** state **==** 1**;**

**}**

/\*\*

\* Método que emula el funcionamiento de un autómata finito, analiza un

\* token para determinar si es un Delimitador

\*

\* @param token Es un token del lenguaje

\* @return Regresa un booleano indicando si el token es un Delimitador

\* siendo el booleano verdadero si es un Delimitador y falso en caso

\* contrario.

\*/

public boolean automatonDelimiters**(**String token**)** **{**

int state **=** 0**;**

**for** **(**int i **=** 0**;** i **<** token**.**length**();** i**++)** **{**

**switch** **(**state**)** **{**

**case** 0**:**

**switch** **(**token**.**charAt**(**i**))** **{**

**case** '{'**:**

state **=** 1**;**

**break;**

**case** '}'**:**

state **=** 2**;**

**break;**

**case** '('**:**

state **=** 3**;**

**break;**

**case** ')'**:**

state **=** 4**;**

**break;**

**case** ';'**:**

state **=** 5**;**

**break;**

**default:**

**return** **false;**

**}**

**break;**

**default:**

**return** **false;**

**}**

**}**

**return** state **==** 1 **||** state **==** 2 **||** state **==** 3 **||** state **==** 4

**||** state **==** 5**;**

**}**

/\*\*

\* Método que emula el funcionamiento de un autómata finito, analiza un

\* token para determinar si es un Operador

\*

\* @param token Es un token del lenguaje

\* @return Regresa un booleano indicando si el token es un Operador siendo

\* el booleano verdadero si es un Operador y falso en caso contrario.

\*/

public boolean automatonOperators**(**String token**)** **{**

int state **=** 0**;**

**for** **(**int i **=** 0**;** i **<** token**.**length**();** i**++)** **{**

**switch** **(**state**)** **{**

**case** 0**:**

**switch** **(**token**.**charAt**(**i**))** **{**

**case** '+'**:**

state **=** 1**;**

**break;**

**case** '-'**:**

state **=** 2**;**

**break;**

**case** '\*'**:**

state **=** 3**;**

**break;**

**case** '/'**:**

state **=** 4**;**

**break;**

**case** '>'**:**

state **=** 5**;**

**break;**

**case** '<'**:**

state **=** 6**;**

**break;**

**case** '='**:**

state **=** 7**;**

**break;**

**case** '!'**:**

state **=** 9**;**

**break;**

**case** '&'**:**

state **=** 11**;**

**break;**

**case** '|'**:**

state **=** 12**;**

**break;**

**default:**

idError **=** 17**;**

**return** **false;**

**}**

**break;**

**case** 7**:**

**if** **(**token**.**charAt**(**i**)** **==** '='**)** **{**

state **=** 8**;**

**}** **else** **{**

idError **=** 18**;**

**return** **false;**

**}**

**break;**

**case** 9**:**

**if** **(**token**.**charAt**(**i**)** **==** '='**)** **{**

state **=** 10**;**

**}** **else** **{**

idError **=** 19**;**

**return** **false;**

**}**

**break;**

**default:**

idError **=** 20**;**

**return** **false;**

**}**

**}**

**return** state **==** 1 **||** state **==** 2 **||** state **==** 3 **||** state **==** 4

**||** state **==** 5 **||** state **==** 6 **||** state **==** 7 **||** state **==** 8

**||** state **==** 9 **||** state **==** 10 **||** state **==** 11 **||** state **==** 12**;**

**}**

/\*\*

\* Método que emula el funcionamiento de un autómata finito, analiza un

\* token para determinar si es una palabra reservada

\*

\* @param token Es un token del lenguaje

\* @return Regresa un booleano indicando si el token es un palabra reservada

\* siendo el booleano verdadero si es una palabra reservada y falso en caso

\* contrario.

\*/

public boolean automatonReservedWords**(**String token**)** **{**

int state **=** 0**;**

**for** **(**int i **=** 0**;** i **<** token**.**length**();** i**++)** **{**

**switch** **(**state**)** **{**

**case** 0**:**

int j **=** i**;**

**if** **((**token**.**length**()** **-** j**)** **>=** 2**)** **{**

i **=** i **+** 1**;**

**if** **(**token**.**subSequence**(**j**,** j **+** 2**).**equals**(**"if"**))** **{**

state **=** 2**;**

**break;**

**}**

**}** **else** **{**

idError **=** 7**;**

**return** **false;**

**}**

**if** **((**token**.**length**()** **-** j**)** **>=** 4**)** **{**

i **=** i **+** 2**;**

**if** **(**token**.**subSequence**(**j**,** j **+** 4**).**equals**(**"null"**))** **{**

state **=** 4**;**

**break;**

**}**

**}** **else** **{**

idError **=** 8**;**

**return** **false;**

**}**

**if** **((**token**.**length**()** **-** j**)** **>=** 5**)** **{**

i **=** i **+** 1**;**

**if** **(**token**.**subSequence**(**j**,** j **+** 5**).**equals**(**"while"**))** **{**

state **=** 1**;**

**break;**

**}** **else** **if** **(**token**.**subSequence**(**j**,** j **+** 5**).**equals**(**"break"**))** **{**

state **=** 3**;**

**break;**

**}** **else** **if** **(**token**.**subSequence**(**j**,** j **+** 5**).**equals**(**"print"**))** **{**

state **=** 5**;**

**break;**

**}** **else** **{**

idError **=** 9**;**

**return** **false;**

**}**

**}** **else** **{**

idError **=** 10**;**

**return** **false;**

**}**

**default:**

idError **=** 11**;**

**return** **false;**

**}**

**}**

**return** state **==** 1 **||** state **==** 2 **||** state **==** 3 **||** state **==** 4 **||** state **==** 5**;**

**}**

/\*\*

\* Método que emula el funcionamiento de un autómata finito, analiza un

\* token para determinar si es un tipo de dato valido

\*

\* @param token Es un token del lenguaje

\* @return Regresa un booleano indicando si el token es un tipo de dato

\* siendo siendo el booleano verdadero si es un tipo de dato y falso en caso

\* contrario.

\*/

public boolean automatonDataTypes**(**String token**)** **{**

int state **=** 0**;**

**for** **(**int i **=** 0**;** i **<** token**.**length**();** i**++)** **{**

**switch** **(**state**)** **{**

**case** 0**:**

**if** **((**token**.**length**()** **-** i**)** **>=** 3**)** **{**

**if** **(**token**.**subSequence**(**i**,** i **+** 3**).**equals**(**"int"**))** **{**

state **=** 1**;**

**}** **else** **if** **(**token**.**subSequence**(**i**,** i **+** 3**).**equals**(**"str"**))** **{**

state **=** 2**;**

**}** **else** **{**

idError **=** 4**;**

**return** **false;**

**}**

i **=** i **+** 2**;**

**}** **else** **{**

idError **=** 5**;**

**return** **false;**

**}**

**break;**

**default:**

idError **=** 6**;**

**return** **false;**

**}**

**}**

**return** state **==** 1 **||** state **==** 2**;**

**}**

/\*\*

\* Método que emula el funcionamiento de un autómata finito, analiza un

\* token para determinar si es un comentario

\*

\* @param token Es un token del lenguaje

\* @return Regresa un booleano indicando si el token es un comentario siendo

\* el booleano verdadero si es un comentario y falso en caso contrario.

\*/

public boolean automatonComments**(**String token**)** **{**

int state **=** 0**;**

**for** **(**int i **=** 0**;** i **<** token**.**length**();** i**++)** **{**

**switch** **(**state**)** **{**

**case** 0**:**

**if** **(**token**.**charAt**(**i**)** **==** '#'**)** **{**

state **=** 1**;**

**}** **else** **{**

**return** **false;**

**}**

**break;**

**case** 1**:**

**if** **(**token**.**charAt**(**i**)** **==** '#'**)** **{**

state **=** 2**;**

**}** **else** **if** **(**Character**.**isDefined**(**token**.**charAt**(**i**)))** **{**

**}** **else** **{**

**return** **false;**

**}**

**break;**

**case** 2**:**

**break;**

**default:**

**return** **false;**

**}**

**}**

**return** state **==** 2**;**

**}**

/\*\*

\* Analiza un símbolo y determina si es un número. Es un número si el

\* símbolo esta entre el <0-9>

\*

\* @param character

\* @return Regresa true si es un número y false en caso contrario

\*/

private boolean isNumber**(**char character**)** **{**

**return** character **>=** '0' **&&** character **<=** '9'**;**

**}**

/\*\*

\* Analiza un símbolo y determina si es una letra. Es una letra si el

\* símbolo se encuentra entre a-z o A-Z

\*

\* @param character

\* @return Regresa true si es una letra y false en caso contrario

\*/

private boolean isLetter**(**char character**)** **{**

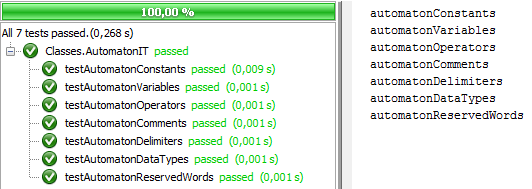
**return** character **>=** 'a' **&&** character **<=** 'z' **||** character **>=** 'A' **&&** character **<=** 'Z'**;**

**}**

**}**

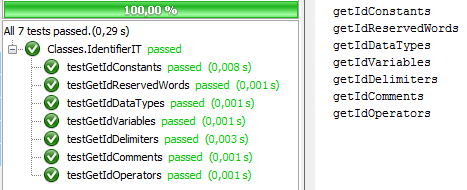
# Resultados JUnit

## Clase: Automaton



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NOMBRE: | Pruebas Automaton | |  |  |
| ACTIVIDADES | | Análisis de requerimientos del sistema | | |
| TIEMPO ESTIMADO | | 45 – 60 minutos por unidad | | |
| MÉTODOS O HERRAMIENTAS | | Netbeans junto con la librería JUnit | | |
| ENTREGABLES | | Resultado de pruebas unitarias | | |

## Clase: Identifier



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NOMBRE: | Pruebas Identifier | |  |  |
| ACTIVIDADES | | Análisis de requerimientos del sistema | | |
| TIEMPO ESTIMADO | | 45 – 60 minutos por unidad | | |
| MÉTODOS O HERRAMIENTAS | | Netbeans junto con la librería JUnit | | |
| ENTREGABLES | | Resultado de pruebas unitarias | | |

# Correcciones y cambios realizados

## Clase: Automaton

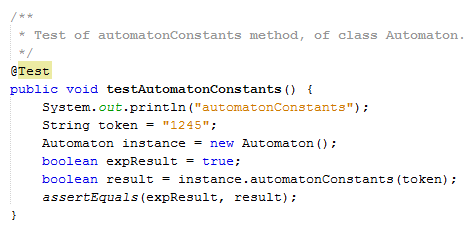
Se modificó cada método de prueba para que éste se acoplara a su método correspondiente en la clase Automaton. Por ejemplo, se asignó una cadena válida junto con su respectivo valor booleano

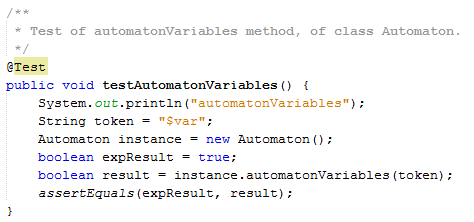
## Clase: Identifier

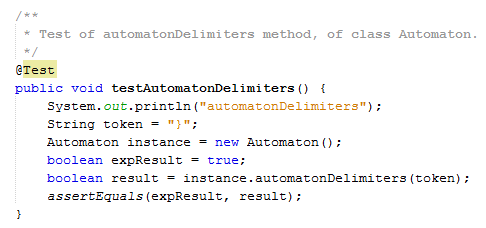
Se modificó cada método de prueba para que éste se acoplara a su método correspondiente en la clase Identifier. Por ejemplo, se asignó una cadena válida junto con su respectivo valor booleano

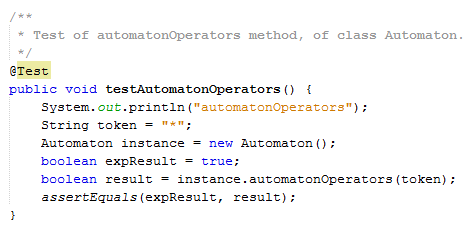
# Código corregido

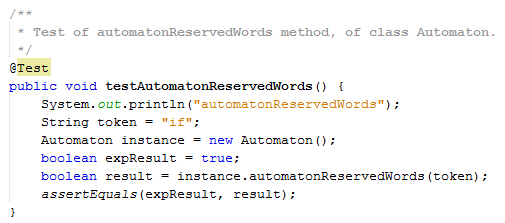
## Clase: Automaton

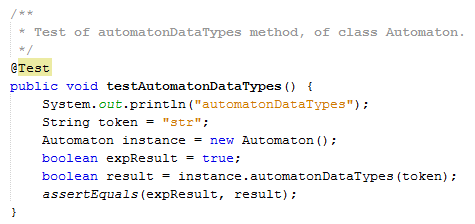


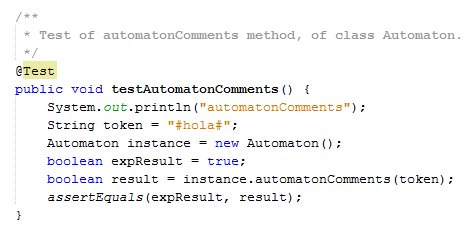




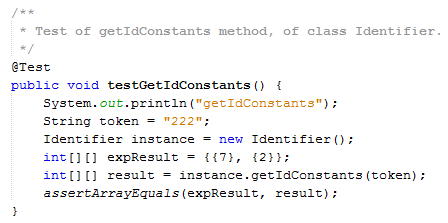


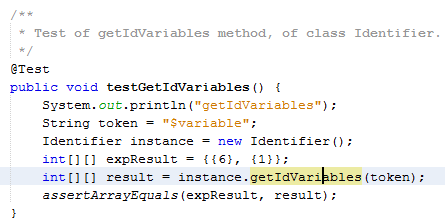


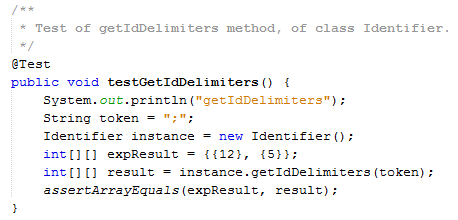


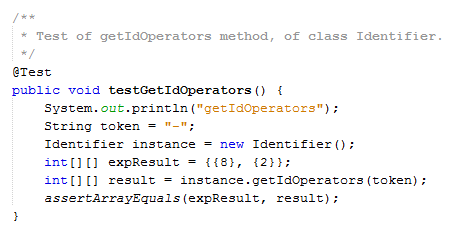


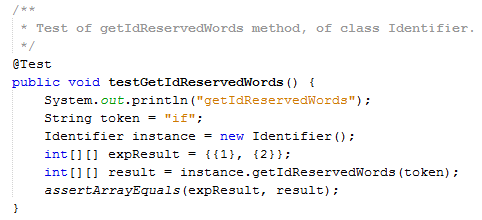
## Clase: Identifier

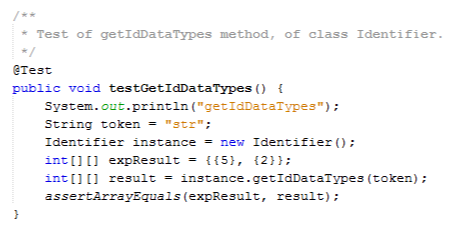


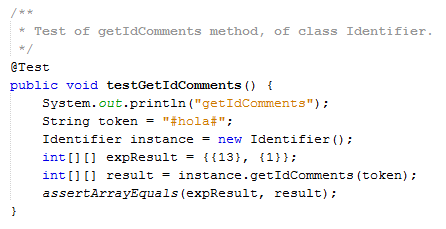












# Conclusión

Este trabajo ayudó para comprender que existen programas para poder testear programas, en este trabajo se realizaron principalmente pruebas unitarias con la librería JUnit, esta librería tiene muchas herramientas que son de gran utilidad al momento de ejecutar pruebas sobre los programas.

En un principio, al ejecutar las pruebas, éstas mostraron resultados con porcentajes iguales a 0 o menor que 50; sin embargo, esto se debió al hecho de que los métodos requieren parámetros específicos y con datos, por lo tanto se colocaron los parámetros necesarios para la ejecución del método y los resultados fueron los que se mostraron en apartados anteriores.

# Referencias

<https://es.wikipedia.org/wiki/Prueba_unitaria>

<https://www.youtube.com/watch?v=Z47B85vuxUg>

<http://www.pmoinformatica.com/2014/05/plan-de-pruebas-de-software.html>

<https://www.youtube.com/watch?v=fyPfiEXt5i8>

# Glosario

* **Automaton**

Palaba en inglés utilizada para definir una máquina de estados finitos no vacíos, es decir, un autómata finito.

* **Identifier**

Palabra en inglés utilizada para definir un identificador. En programación equivale al nombre asignado a una variable.