"UNIVERSIDAD DE HUELVA - UHU"

PROCESADORES DEL LENGUAJE - 2019

TRADUCTOR DE LOGO A SKETCH

JESÚS CAMPOS MÁRQUEZ

GRADO INGENIERÍA INFORMÁTICA
(ESPECIALIDAD DE COMPUTACIÓN)

INDICE

1. La Gramática de Logo
2. ¿Qué es y cómo funciona Sketch?
3. Java CC y LogoParser.jj
4. Expresiones aritméticas y jerarquía de llamadas
5. Comandos básicos
6. Trigonometría necesaria para la construcción de las instrucciones básicas
7. La tabla de símbolos
8. Comandos con variables
9. Control de flujo
10. Procedimientos
11. La máquina virtual
12. Logo Compiler
13. Desarrollo de la instrucción IF (Flujo)
14. Ejecución del jar
15. Webgrafía

LA GRAMATICA DE LOGO

La gramática de logo está compuesta por producciones en la forma EBNF, capaces de jerarquizar las llamadas necesarias a las funciones para llevar a cabo el traductor.

Podemos encontrar la gramática básica en el siguiente fichero: "Gramática de logo". proporcionada por Gonzalo Aranda Corral.

Para esta gramática se han realizado las tareas de análisis y de síntesis con JavaCC, generando el fichero LogoParser.jj, y con el conseguimos la ayuda para generar el código que puede ejecutar SKETCH.

¿QUÉ ES Y CÓMO FUNCIONA SKETCH?

SKETCH es una interfaz gráfica de java, creada por el compañero Javier Martín Moreno. Pueden encontrar el proyecto en su github.

La interfaz tiene como esquina superior izquierda las coordenadas (0,0) y como ángulo superior dentro de nuestro eje de coordenadas el -90.

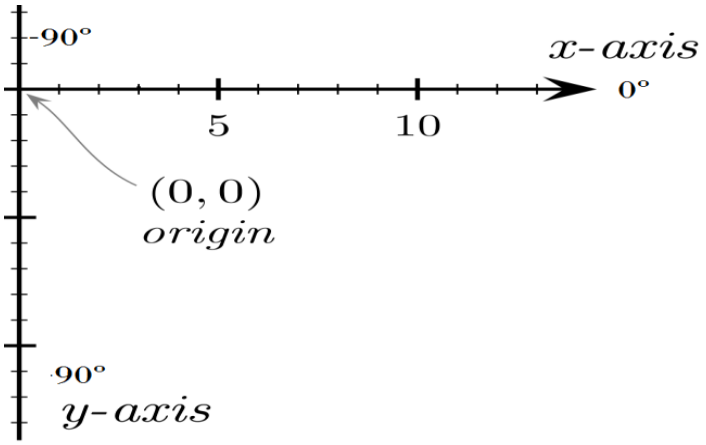


Figura 2.1 – Cuadrante y ángulos utilizados por SKETCH

SKECTH utiliza una lista de Instrucciones, que pueden ser:

- LINEA (xi,yi,xf,yf): Pinta una línea desde un punto inicial a un punto final, siendo su longitud la diferencia entre esos puntos.
- ARCO (centroX,centroY,Radio,alfai,alfaf): Pinta un arco desde unas coordenadas, con un radio desde un ángulo inicial a uno final.
- BORRAR (): Borra el contenido de la ventana.

Además, añade la funcionalidad de poder guardar lo pintado en pdf.

Podemos encontrar más información de cómo utilizarlo en su **manual**.

JAVA CC Y LOGOPARSER.JJ

JavaCC (Java Compiler Compiler) es una herramienta de generación automática de analizadores gramaticales basada en Java.

El funcionamiento de la herramienta consiste en analizar un fichero de entrada, que contiene la descripción de una gramática, y generar un conjunto de ficheros de salida, escritos en Java, que contienen la especificación de un analizador léxico y de un analizador sintáctico para la gramática especificada.

La especificación léxica se basa en expresiones regulares y la especificación sintáctica utiliza el formato EBNF.

Por defecto JavaCC analiza gramáticas de tipo LL(1), pero permite fijar un Lookahead mayor (para analizar gramáticas LL(k)) e incluso utilizar un Lookahead adaptativo.

Podemos encontrarlo en https://javacc.org/.

Por lo tanto, para poder llevar a cabo la práctica se ha generado un **LogoParser.jj**, que contendrá toda la especificación de la gramática, tanto tokens a consumir como llamadas a las funciones.

EXPRESIONES ARITMÉTICAS

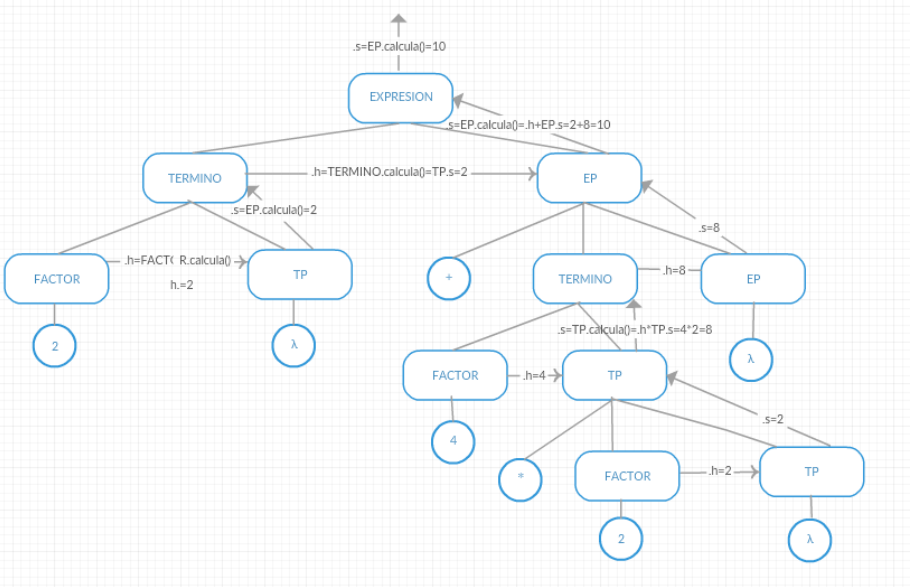
En logo se pueden realizar las operaciones comunes de **+**, **-**, ***** y **/**, de números enteros.

Para llevar a cabo estas expresiones nuestra gramática tiene una producción Expresion que genera todo lo que podemos ver en la imagen:

```
Expresion -->  Termino Ep
Ep  -->  (< MAS > | <  MENOS >) Termino  Ep )?
Termino -->  Factor Tp
Tp --> ((< PROD > | < DIV > ) Factor  Tp)?
Factor -->< NUMERO >
        | < VARIABLE >
        | < CADENA >
        | < NATIVA > Expresion
        | < PARABI > Expresion <PARCER>
```

Figura 4.1 – Gramática asociada a una Expresión

Veamos su jerarquía de forma más clara con un ejemplo. Supongamos la siguiente expresión: **2+4*2**, cuyo resultado debería de ser **10**. Consideramos que los atributos con **.h** son **heredados**/pasados por parámetro y los atributos con **.s** son **sintetizados**/devueltos por la



función. Además, los cálculos finales serán devueltos por las funciones EP y TP, ya que son las que contienen el tipo de operación **(+, -, *** y **/).** Ambas funciones realizan el cálculo en su función **calcula(),** y en el caso de que no tengan nada que calcular en sus hijos con su herencia, devuelven el resultado que heredan.

Figura 4.2 – Jerarquía de llamadas mediante ejemplo práctico simple

COMANDOS BÁSICOS

Los comandos básicos están formados por las instrucciones:

- fd (forward): Avanzar en el sentido de la orientación.
- bk (backward): Retroceder.
- arcr: avanza en forma de arco hacia la derecha
- arcl: avanza en forma de arco hacia la izquierda
- rt (right): Cambiar la orientación de la tortuga hacia la derecha.
- lt (left): ... a la izquierda.
- cs (clearscreen): Borrar la pantalla.
- pu (penup): Levanta el "bolígrafo" y todos los movimientos que realice a partir de ahí no pintaran (aunque se realizan).
- pd (pendown): Baja el "bolígrafo" y vuelve a pintar.
- ht (hideturtle): Oculta la tortuga
- st (showturtle): Muestra la tortuga
- home: Se mueve directamente a la posición inicial (0,0) y se pone con orientación inicial de 90º.
- setxy: Desplaza la tortuga al punto x,y que se le indique.

Además de las instrucciones **nativas** como:

- random: genera un numero aleatorio entre 0 y el numero indicado

- cos: devuelve el coseno de un número.
- sin: devuelve el seno de un número.
- mod: devuelve el módulo entre dos números.

Para poder utilizar funciones con más de una Expresion, se ha modificado la gramática inicial, creando un NATIVA2, con la que si es posible trabajar con dos Expresiones.

Para generar los comandos básicos se han tenido que crear Nodos de cada instrucción, que reciben por parámetro lo necesario para generar la instrucción de forma correcta.

Por ejemplo, el Nodo_FD, está compuesto por un Nodo_Exp, ya que le precede una expresión que habrá que calcularla, y pintará una línea con la longitud de la expresión devuelta.

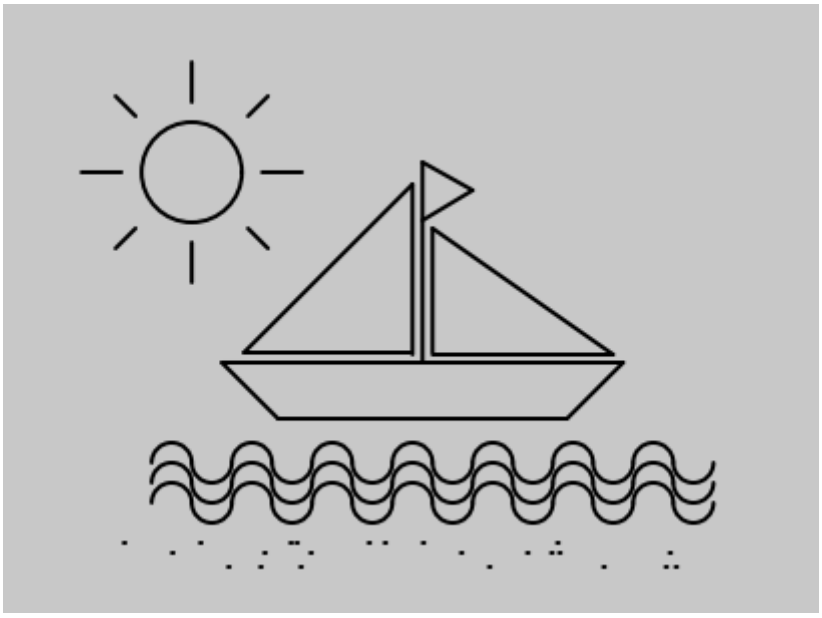


Figura 5.1 - Ejemplo de instrucciones básicas

CLICK PARA VER EL CODIGO

CLICK PARA VER EL PDF GENERADO CON SAVEALL

Todas las instrucciones básicas están dentro del package Instrucciones.

TRIGONOMETRÍA NECESARIA PARA LA CONTRUCCION DE LAS INSTRUCCIONES BÁSICAS

Para la construcción de las instrucciones, hay que tener en cuenta la orientación en la que se está. Por ejemplo, para hacer una línea tendremos que tener en cuenta cuanto nos desplazamos tanto en el eje X, como en el eje Y. Para ello utilizamos las fórmulas:

- $R \cdot \sin \alpha$
- $R \cdot \cos \alpha$

Siendo **R** la longitud o el radio en el caso de los arcos, y **α** la orientación en radianes. Teniendo en cuenta que nosotros trabajamos en grados Euler, vamos a hacer una transformación multiplicando los grados por $\frac{\pi}{180}$.

Además, un punto importante a destacar, es que, para el cálculo de los arcos, tenemos que tener en cuenta que, como nuestra orientación está en -90º inicialmente, tenemos que sumarle o restarle, dependiendo si es arcr o arcl, $\frac{\pi}{2}$ para desplazarnos bien en los ejes.

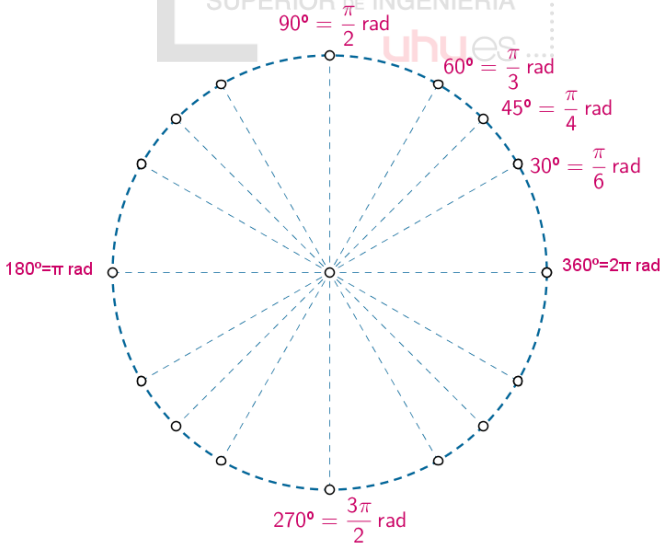


Figura 6.1 – Ángulos en radianes según cuadrantes.

LA TABLA DE SIMBOLOS

La tabla de símbolos nos proporciona la ayuda para la verificación semántica del lenguaje. Su funcionamiento se basa en una estructura que almacena símbolos con su correspondiente valor en diferentes ámbitos anidados. Dichos ámbitos deben implementarse mediante una pila donde se irán insertando y eliminando cada vez que se cree o se destruya uno.

Consideramos que un ámbito es cuando se define un bloque de programa, es decir, el programa principal tiene el ámbito_1, y un **procedimiento** creado, en su bloque tendrá el ámbito_2, el cual cuando acabe el **procedimiento** se destruirá, consiguiendo así una visibilidad hacia fuera.

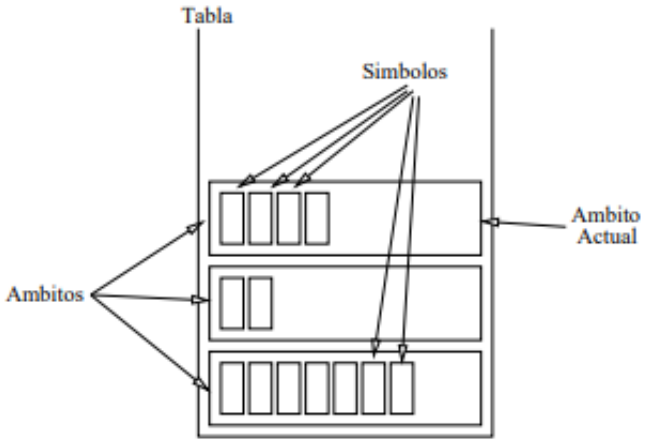


Figura 5.1

Está dentro del package tablaSimbolos.

COMANDOS CON VARIABLES

Logo permite la posibilidad de almacenar valores y reutilizarlos accediendo a ellos mediante variables.

Estas variables tienen nombres alfanuméricos (comenzando por letra), sin caracteres especiales, y para usarlas contendrán el carácter **":"** al principio del nombre de la variable.

Para su creación se utilizará la instrucción make “variable valor.

Está dentro del package Instrucciones.

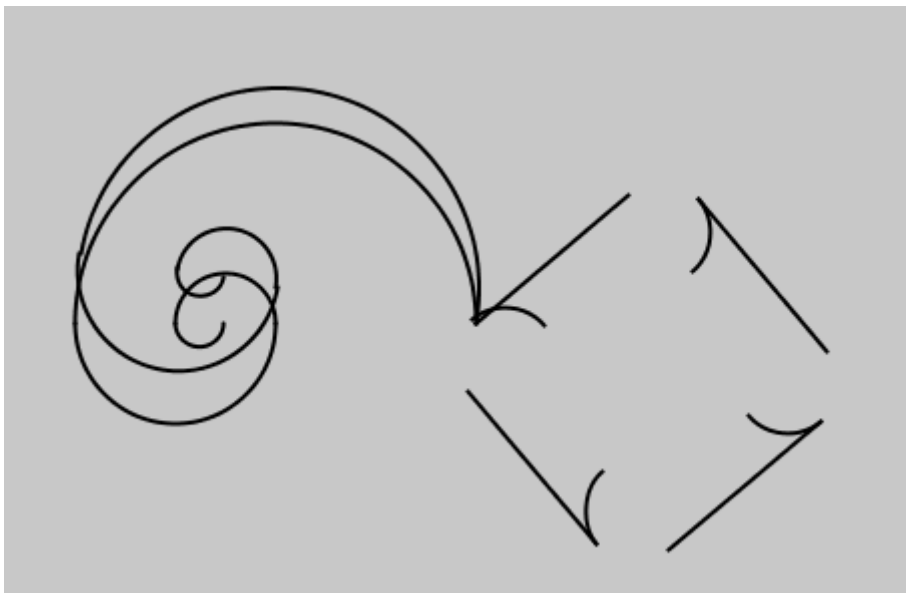


Figura 8.1. – Ejemplo 1 de instrucciones con variables

CLICK PARA VER EL CODIGO

CLICK PARA VER EL PDF GENERADO CON SAVEALL

CONTROL DE FLUJO

Para el control de flujo de ejecución del programa, tenemos 3 instrucciones:

- Repeat: realiza la repetición de Expresión veces el bloque de instrucciones dentro de su loque.

- If: Evalúa la condición y en caso de que sea cierta se ejecutará el bloque de instrucciones, y en caso contrario no.

- For: Repite para una variable, desde el valor inicial de la Expresion1, hasta la Expresion2, incrementando cada Expresion3 y repitiendo el bloque de instrucciones.

Están dentro del package flujo.

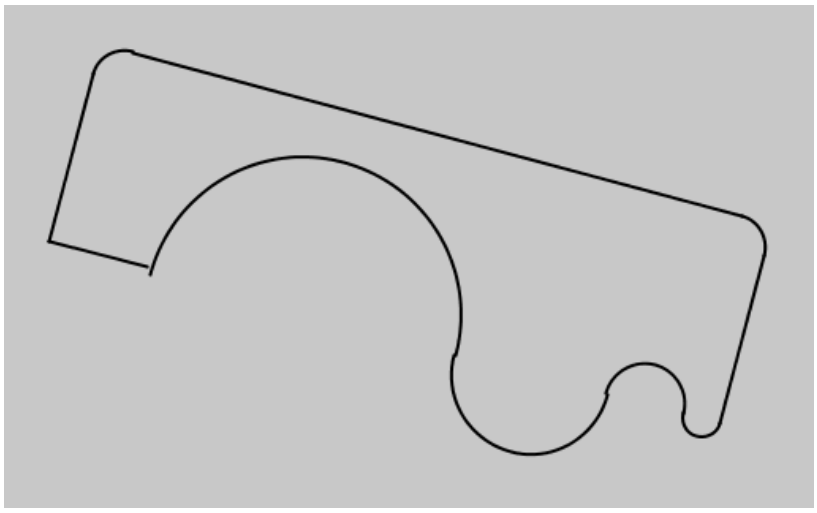


Figura 9.1 – Ejemplo de figura generada mediante instrucción repeat

CLICK PARA VER EL CODIGO

CLICK PARA VER EL PDF GENERADO CON SAVEALL

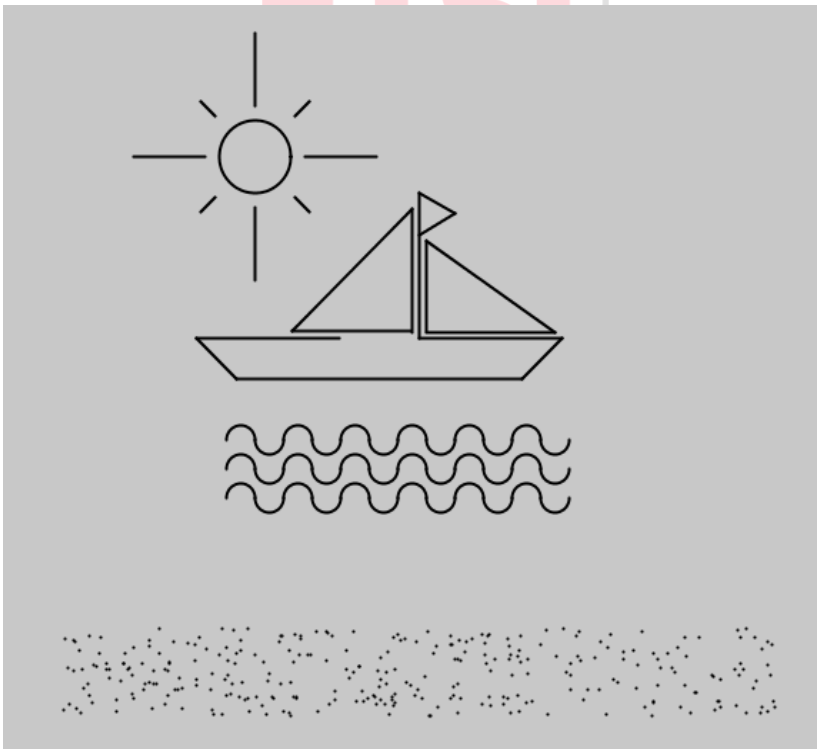


Figura 9.2 – Ejemplo de Figura generada mediante instrucción for

CLICK PARA VER EL CODIGO

CLICK PARA VER EL PDF GENERADO CON SAVEALL

PROCEDIMIENTOS

Los procedimientos en Logo están definidos como **TO *nombre_procedimientos parámetros*** y ***bloque_instrucciones***, y nos sirven para encapsular un bloque de instrucciones. Hemos tenido en cuenta que los procedimientos pueden tener o no parámetros y para ello cuando se crean los procedimientos tenemos que guardar esos parámetros para posteriormente en su llamada con un paso de parámetros asignarlos de forma correcta.

Como posible mejora seria poder implementar funciones con recursividad.

Además, es necesario hacer las llamadas a estas funciones sin intercalar creando nuevas, es decir, creamos todos los procedimientos y posteriormente los llamamos.

Por último, debemos de saber que los procedimientos no devuelven nada, y que, tras finalizar un procedimiento, la tortuga volverá al estado anterior.

Están dentro de nuestro package procedimientos.

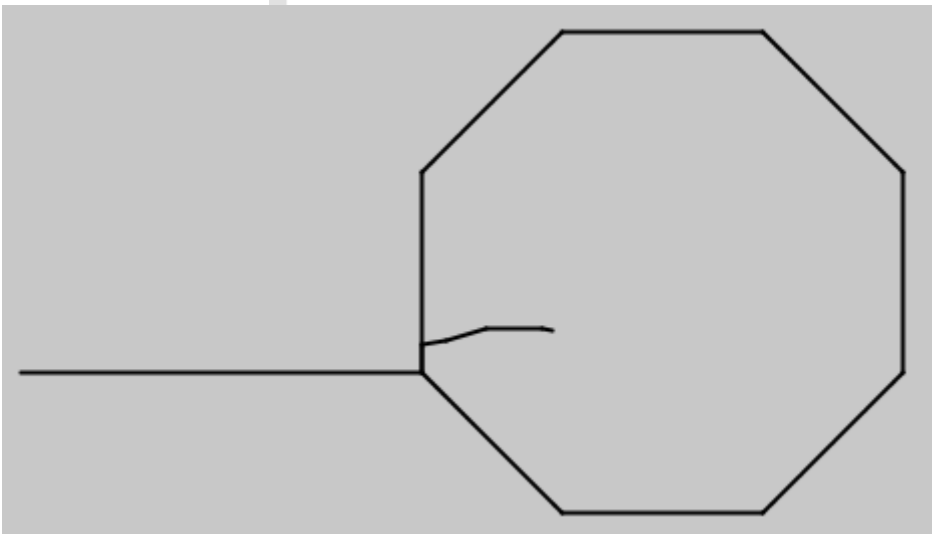


Figura 10.1 – Ejemplo de Figura generada con procedimientos

CLICK PARA VER EL CODIGO

CLICK PARA VER EL PDF GENERADO CON SAVEALL

LA MÁQUINA VIRTUAL

Para llevar a cabo todo el procedimiento de pintado, se ha creado una clase llamada MaquinaVirtual, que va guardando el estado en el que se encuentra nuestra tortuga, aunque esta o se vea y, además, indica su orientación.

La máquina virtual contiene la tabla de símbolos, para que en todo el procedimiento de la ejecución del programa se guarden las variables del fichero en su ámbito correspondiente, y aunque se haga un clearscreen, esto significa que se borra la pantalla, pero no las variables creadas con anterioridad.

Por lo tanto, nos ayuda a mantener los puntos exactos en los que nos encontramos, para poder pintar en SKECTH de manera correcta.

LOGO COMPILER

Es el programa principal, encargado de coger el fichero a ejecutar. Este fichero se enviará al LogoParser, creado automáticamente cuando se compila el código JavaCC (LogoParser.jj).

Posteriormente creará la tabla de símbolos y la maquina virtual, que contendrá las dimensiones del SKECTH.

Hay que tener en cuenta que, para poder obtener los procedimientos y las instrucciones de forma correcta se ha realizado una traducción en dos pasadas.

Tras esto se pintará todo aquello que se haya especificado en el fichero Main.logo, que contendrá todas las instrucciones que se quieran pintar, devolviendo una lista de instrucciones a ejecutar por SKECTH.

Se encuentra en el package logo junto con los Nodos principales que son Nodo_Programa, Nodo_BloquePrograma y Nodo_Procedimientos(No terminado en esta versión).

Cuando el fichero contiene fallo, se generará un LogoErrors.txt. Además, cuando se ejecuta se genera un LogoOutput.txt que indica si la ejecución fue "Correcta" o "Incorrecta".

DESARROLLO DE LA INSTRUCCIÓN IF

Cuando hablamos de una instrucción If, somos conscientes de que se trata que, si se cumple una cierta condición, se ejecuta un bloque de instrucciones, y en caso contrario no se hace. Para llevarlo a cabo tenemos un Nodo_Condición, que en su método calcula devolverá un 1 si la condición compuesta de la Expresion1 y la Expresion2 es correcta, y 0 en caso contrario.

Por lo tanto, nuestro Nodo_If, comprobará si su condición devuelve un 1 para ejecutar el bloque de instrucciones, y en caso de ser 0 no lo ejecutará.

EJECUCION DEL JAR

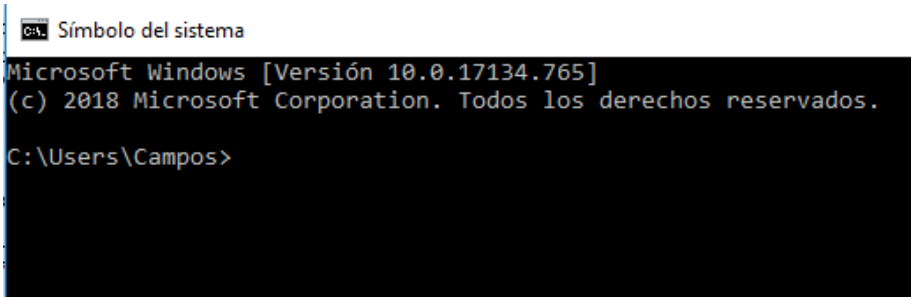
Para la ejecución del traductor, se ha generado un jar, aunque también está disponible todo el código fuente para poder verlo en profundidad. Recordar que el proyecto se ha creado con Eclipse para que se pueda abrir sin ningún tipo de problemas. En cuanto al ejecutable podemos dejarlo en cualquier sitio, es decir, podemos dejar el jar en el escritorio, pero si hacemos un saveAll o al generarse un fichero de OutputLogo.txt, este se generará también en el escritorio. Recomiendo meterlo en una carpeta.

Como primera opción, podemos ejecutar directamente el .jar dando doble click, pero este tendrá que estar en el mismo directorio que el Main.logo a ejecutar.

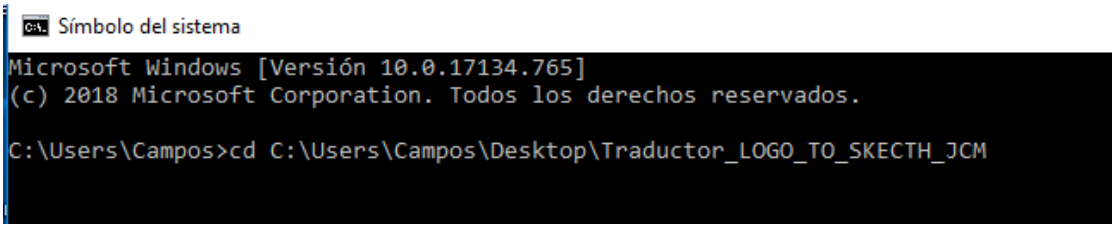
En cuanto a la segunda opción, la ejecución la vamos a realizar a través de la consola. Para ello tenemos que tener en cuenta que el jar recibe como parámetro la un string que corresponderá al directorio donde se encuentre el Main.Logo a ejecutar.

Para ello:

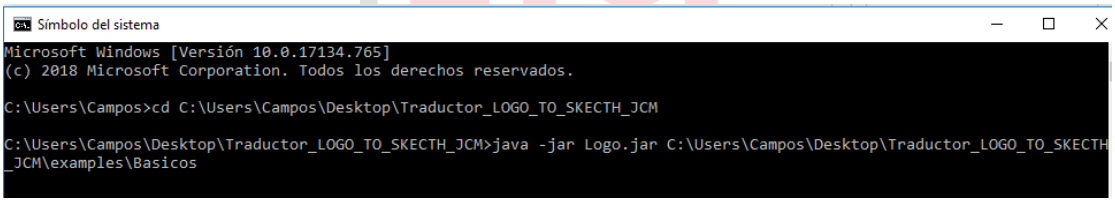
1. Abrir el cmd.



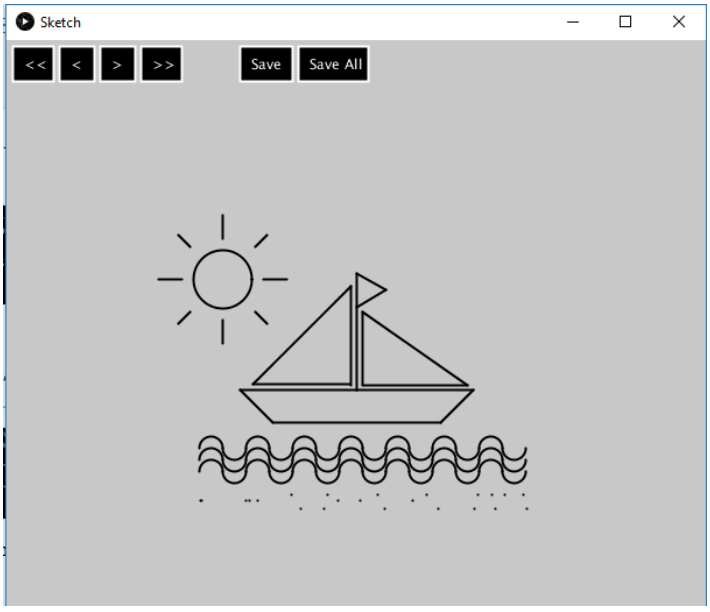
2. Cambiar el directorio hacia donde hallamos puesto Logo.jar.



3. Escribir: java -jar Logo.jar "Directorio_donde_este_el_Main.logo_sin_comillas".



4. La ejecución comienza y podemos ejecutar el SKECTH.



WEBGRAFÍA

https://www.google.com/search?q=cuadrante+radiane&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjbkdfIm-viAhVCQhoKHerVCLYQ_AUIECgB&biw=1366&bih=657#imgrc=UEl14iyKeWtxSM:

http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/41_TablasDeSimbolos.pdf

http://www.uhu.es/francisco.moreno/gii_pl/

http://people.eecs.berkeley.edu/~bh/logo.html

Página Moodle de la asignatura Procesadores del Lenguaje de la Universidad de Huelva – UHU