Universidad del Valle de Guatemala
Departamento de Matemática
Licenciatura en Matemática Aplicada

Estudiante: Rudik Roberto Rompich
E-mail: rom19857@uvg.edu.gt
Carné: 19857

CC2003 - Algoritmos y Estructuras de Datos - Catedrático: Melvin García
8 de abril de 2021

Intérprete Lisp

Dialecto elegido: *Common Lisp.*
Repositorio: https://github.com/RudiksChess/IntepreteLisp.git

1. Funciones a implementar:

(a) Operaciones aritméticas

Descripción. Se puede consultar acá:
click □

(b) Instrucción QUOTE o ‘ (single quote, para interrumpir el proceso de evaluación de expresiones)

Descripción. The quote special operator just returns object. Simply returns x. The object is not evaluated and may be any Lisp object whatsoever. This construct allows any Lisp object to be written as a constant value in a program. Ejemplos:

```
(setq a 1) =>  1
(quote (setq a 3)) =>  (SETQ A 3)
a =>  1
'a =>  A
''a =>  (QUOTE A)
'''a =>  (QUOTE (QUOTE A))
(setq a 43) =>  43
(list a (cons a 3)) =>  (43 (43 . 3))
(list (quote a) (quote (cons a 3))) =>  (A (CONS A 3))
1 =>  1
'1 =>  1
```

```
"foo" =>  "foo"
'"foo" =>  "foo"
(car '(a b)) =>  A
'(car '(a b)) =>  (CAR (QUOTE (A B)))
#(car '(a b)) =>  #(CAR (QUOTE (A B)))
'#(car '(a b)) =>  #(CAR (QUOTE (A B)))
```

□

(c) Definición de funciones (DEFUN)

Descripción. `(defun name (parameter-list) "Optional documentation string." body)`

□

(d) SETQ

Descripción. The special form (setq var1 form1 var2 form2 ...) is the "simple variable assignment statement" of Lisp. First form1 is evaluated and the result is stored in the variable var1, then form2 is evaluated and the result stored in var2, and so forth. The variables are represented as symbols, of course, and are interpreted as referring to static or dynamic instances according to the usual rules. Therefore setq may be used for assignment of both lexical and special variables. Ejemplos:

```
(setq a 1)
(setq b 2)
a => 1
b => 2
```

□

(e) Predicados (ATOM, LIST, EQUAL, $<$, $>$),

1. ATOM

   Descripción. The predicate atom is true if its argument is not a cons, and otherwise is false. [6] Note that:

   `(atom '()) is true, because ()=nil.`

   Ejemplos:

   ```
   (atom 'sss) =>  true
   (atom (cons 1 2)) =>  false
   (atom nil) =>  true
   (atom '()) =>  true
   (atom 3) =>  true
   ```

□

2. LIST

   Descripción. *List constructs and returns a list of its arguments* [6]. Ejemplos:

```
(list 3 4 'a (car '(b . c)) (+ 6 -2)) => (3 4 a b 4)
(list) => ()
(list (list 'a 'b) (list 'c 'd 'e)) => ((a b) (c d e))
```

□

3. EQUAL

   Descripción. *The equal predicate is true if its arguments are structurally similar (isomorphic) objects. A rough rule of thumb is that two objects are equal if and only if their printed representations are the same* [6]. Ejemplos:

```
(equal 'a 'b) is false.
(equal 'a 'a) is true.
(equal 3 3) is true.
(equal 3 3.0) is false.
(equal 3.0 3.0) is true.
(equal #c(3 -4) #c(3 -4)) is true.
(equal #c(3 -4.0) #c(3 -4)) is false.
(equal (cons 'a 'b) (cons 'a 'c)) is false.
(equal (cons 'a 'b) (cons 'a 'b)) is true.
(equal '(a . b) '(a . b)) is true.
(equal #\A #\A) is true.
(equal "Foo" "Foo") is true.
(equal "Foo" (copy-seq "Foo")) is true.
(equal "FOO" "foo") is false.
```

□

4. $<$

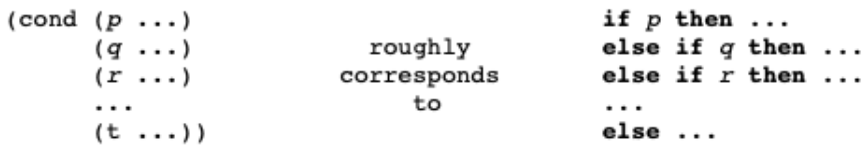
   Descripción. *It takes one or more argument and returns t if either there is a single argument or the arguments are successively smaller from left to right, or nil if otherwise.* □

5. $>$

   Descripción. *It takes one or more argument and returns t if either there is a single argument or the arguments are successively larger from left to right, or nil if otherwise.* □

(f) Condicionales (COND), nota: no es necesario implementar IF.

Descripción. Una imagen



□

(g) Paso de parámetros. (un parámetro puede ser incluso una función)

2. Funcionamiento del intérprete

Se usó como referencia principal [4] y [5].

1. Tokens...
2. Parse...

   Descripción. El parsing consiste en interpretar una serie de Strings. [1], [2], [3]. □

3. Intérprete...

2.1. Java Collection

1. Stack

   Descripción. Fue una forma eficiente para manipular los datos que entran y salen en base a las necesidades del código ingresado. □

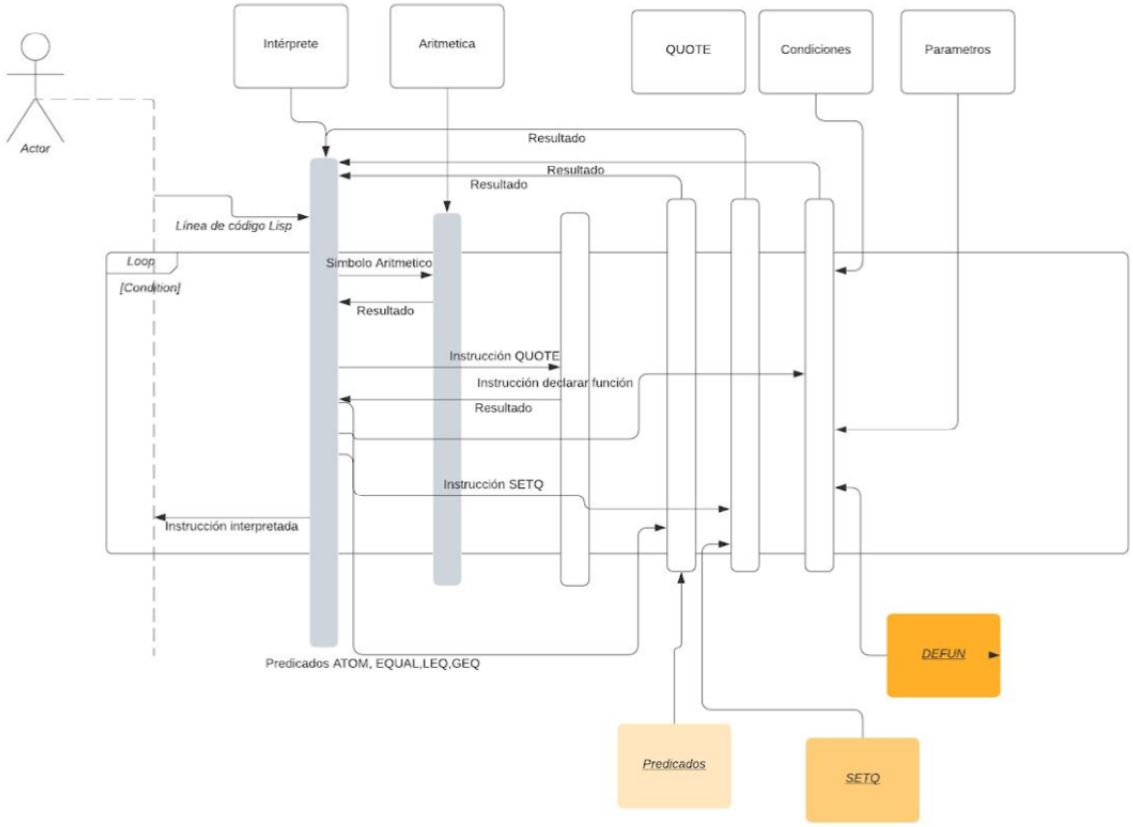
2. ArrayList

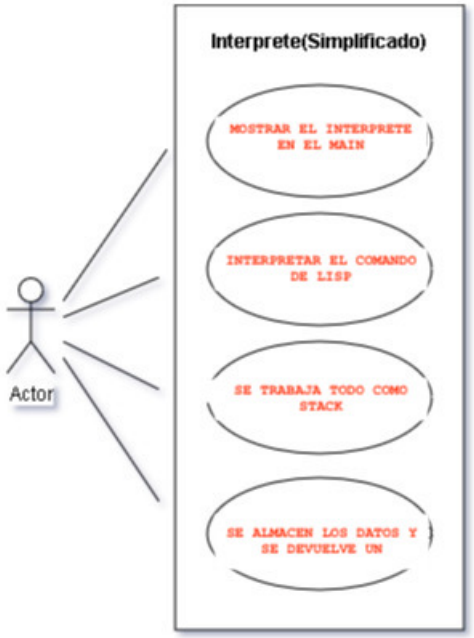
   Descripción. El ArrayList consistió en implementar los stacks a través de este Java Collection, por su vérsatilidad se eligió esta Colección. □

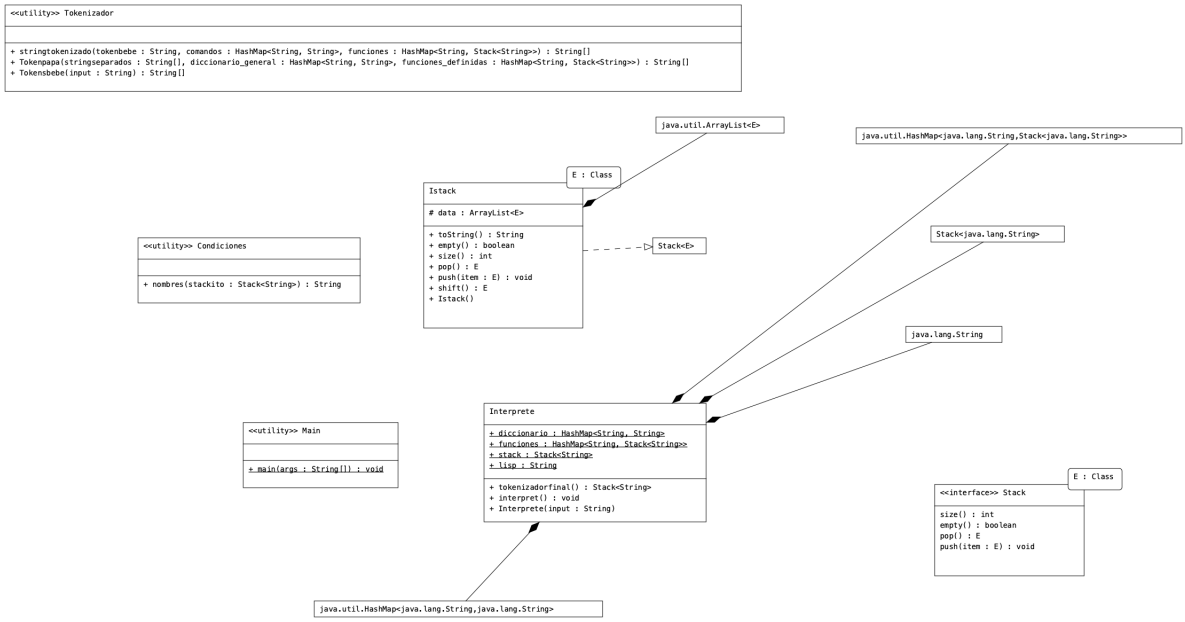
3. HashMap

   Descripción. Su principal usó fue almacenar stack dentro de los Hashmap utilizados. Ya que un simple key fue posible llamar a los stacks y su utilidad fue necesario para el correcto desarrollo del programa. □

3. Diagramas, UML, Secuencias







Referencias

[1] Computerphile. Functional parsing, 31-03-2021.

[2] Computerphile. Parsing, 31-03-2021.

[3] Computerphile. Parsing explained, 31-03-2021.

[4] Cook Maryrose. Little lisp interpreter.

[5] Walter Saldaña. Lisp interpreter.

[6] Carnegie Mellon University. Common lisp the language, 2nd edition.