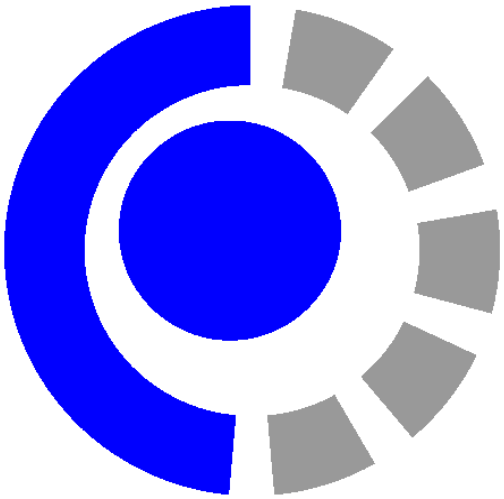


Facultad de Informática

UNIVERSIDAD NACIONAL DEL COMAHUE

Informe Trabajo Final POO 2019

Lucas San Segundo FAI-1921

Jeremías Zuliani FAI-2159

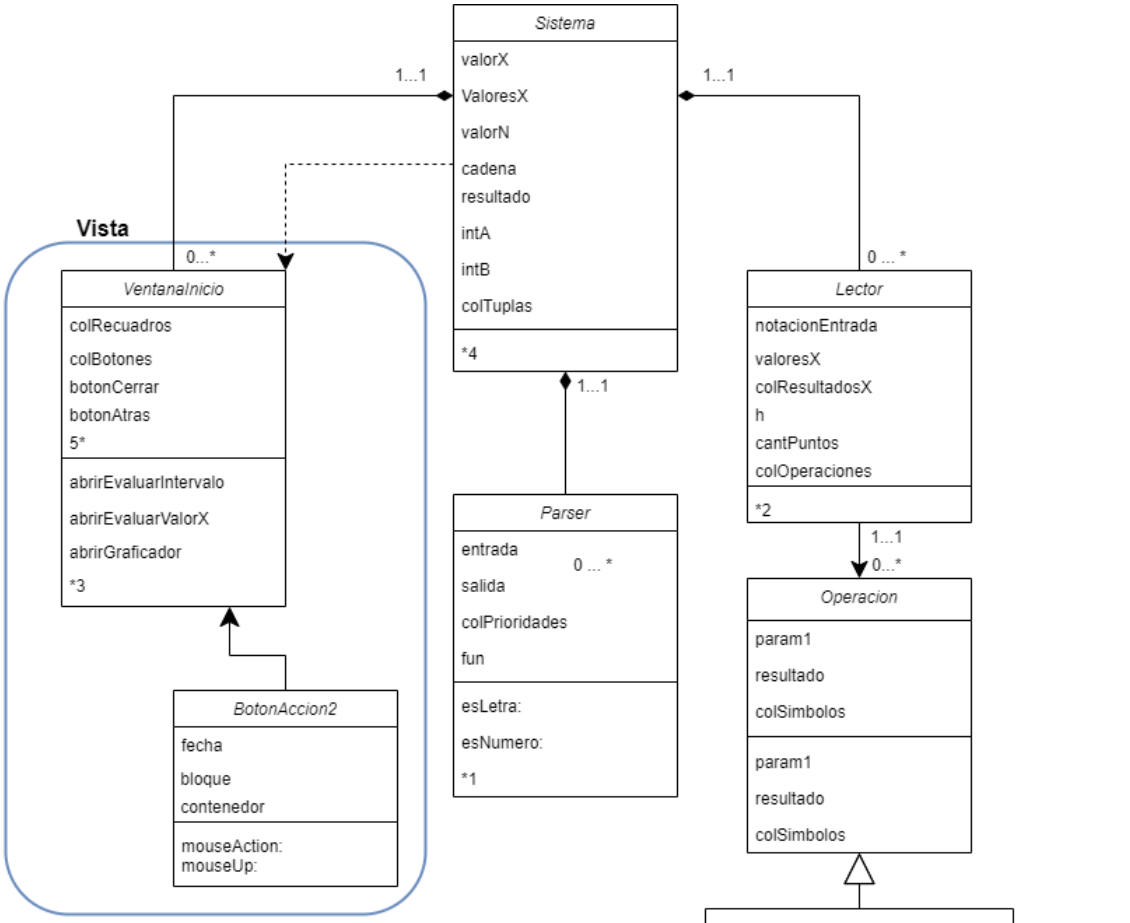
Leandro Nicolás Vallejos FAI-2003

18/02/2021

Introducción:

Durante el desarrollo de este informe, nos enfrentamos a descubrir la solución al problema que consiste en crear un sistema que tenga la capacidad de leer una cadena de texto que represente una ecuación del tipo y=f(x). A partir de dicha ecuación, el sistema deberá hacer operaciones sobre la misma. Dentro de este sistema incluimos un módulo (Parser) que convierte la cadena inicial a notación RPN (Reverse Polish Notation), luego otro (Lector), que permite evaluar y resolver la ecuación (en RPN) tanto para un valor de X como para un intervalo de valores de X. Finalmente contamos con un módulo visual que grafica la ecuación para un intervalo con N puntos. Estos últimos valores son ingresados por el usuario.

Desarrollo:



*1
Parentesis: pila1: pila2esOp
esSimbolo:
getEntrada
getSalida
inicializarParser
inicializarParser:
prioridades: cola: pila1: pila2
rpn
traducirCadena:

*2
analizarNotacion
esNumero:
getColResultados
getValoresX
inicializarLector: conX:
inicializarLector: inicioInterva

*3
abrirMenuPrincipal
cambiarFuncion
cerrar
colocarBotonesAux
inicializarBotonesyRecuadro
initialize
ponerCantPuntos
ponerIntervalo
ponerValorX
resolverValorX
update:
verificarDatosGrafico

*4
convertirTuplas
getCadena
getColRes
getColX
getIntervaloA
getIntervaloB
getTuplas
getValorN
getValorX
inicializarSistema
inicializarSistema:cantPunto
inicializarSistema:valorX:
obtenerResultadoMuchasX
obtenerResultadoUnaX
setCadena:
setIntervaloA:
setIntervaloB:
setValorN:
setValorX:

5*
texFuncion
valorX
resultadoValorX
origen funcion
cantPuntos
inicioInter
finInter intervalo
grafico sis
valorDeX
colTup

Para iniciar, tenemos una clase principal llamada “Sistema”, a la cual le ingresa una cadena f(x) y podemos operar sobre ella. Allí se almacenan valores claves para el problema, tales como la cadena de entrada, él o los resultados, la colección de valores de X (del intervalo), valorN (cantidad de puntos), el intervalo, entre otras. Aquí contamos con métodos para resolver las ecuaciones, como obtener el resultado para un valor de x, o para el intervalo. El modulo visual tiene dependencia con esta clase, de forma que el usuario solamente va a trabajar sobre Sistema. A la hora de operar una ecuación, debemos convertirla en una estructura que nos ayude a facilitar el cálculo, por eso, antes de resolver el problema, Sistema pide a Parser la notación RPN de la función que ingresó el usuario.

La clase Parser almacena la cadena de entrada (dato que le da el sistema), la de salida (cadena en RPN) y la colección de prioridades. El mecanismo utilizado para hacer la traducción es “Shunting-yard algorithm”, método para analizar expresiones matemáticas especificadas en notación infija, y producir una cadena en notación sufija, más conocida como RPN. El diseño de esta nueva cadena nos permite evaluar la función de manera más eficaz, por esto podemos resolver la ecuación haciendo solamente un recorrido por la cadena. El método que se encarga de esta traducción se llama “rpn”.

A la hora de armar la nueva cadena RPN, utilizamos una Cola y dos Pilas, las cuales fueron creadas como dos subclases de Orderedcollection. La variable colPropiedades es una colección que almacena tuplas, el símbolo de la operación y su respectiva prioridad, esto es clave para el desarrollo del algoritmo. Para dar un ejemplo, la operación suma tiene menos prioridad que la multiplicación, caso que se presenta en la siguiente función: *1+2*3;* el algoritmo tiene que saber que primero resolvemos la multiplicación y luego la suma. La notación RPN del ejemplo sería entonces: *1 2 3* * +. Una vez finalizado este proceso, se envía el resultado final a Sistema.

La clase sistema, ya con la cadena RPN, tiene que evaluarla, y para ello utilizamos la clase Lector. En ella encontramos variables como la notación de entrada, valores de X (puede ser uno solo, o varios dentro del intervalo), resultados, la cantidad de puntos, las operaciones disponibles y un valor h. Sea un intervalo [a, b] y c, cantidad de puntos, el valor h es el resultado de (b-a)/c. Este nos permite dividir en partes iguales el intervalo ingresado por el usuario. El funcionamiento del algoritmo que nos ayudara a resolver la ecuación se puede observar en la siguiente tabla, siguiendo el ejemplo anterior:

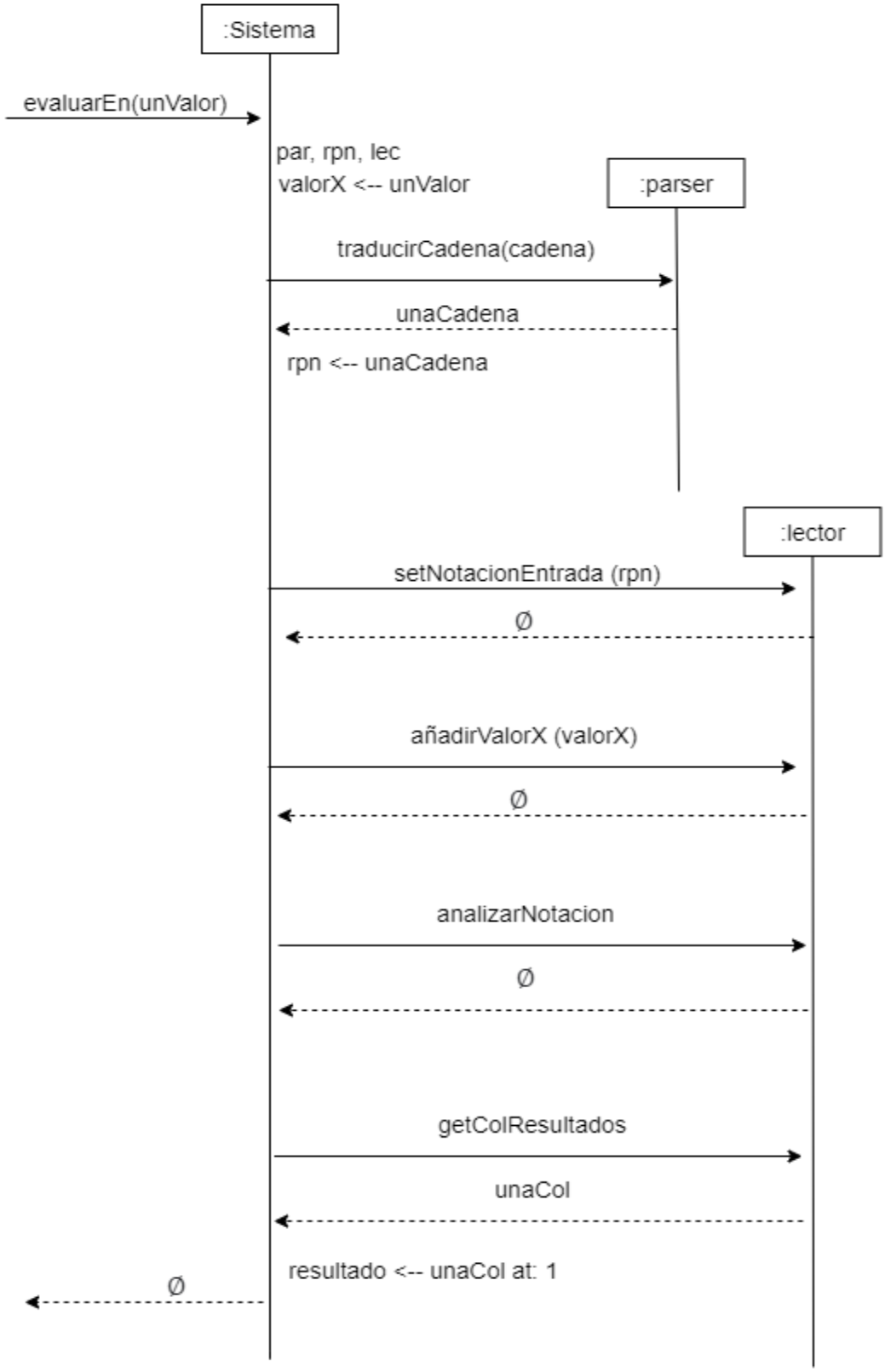
Cadena	Pila	Operación
1 2 3 * +	Vacía	
2 3 * +	1	
3 * +	1 2	
* +	1 2 3	
+	1 6	2 * 3
Fin recorrido	7	1 + 6

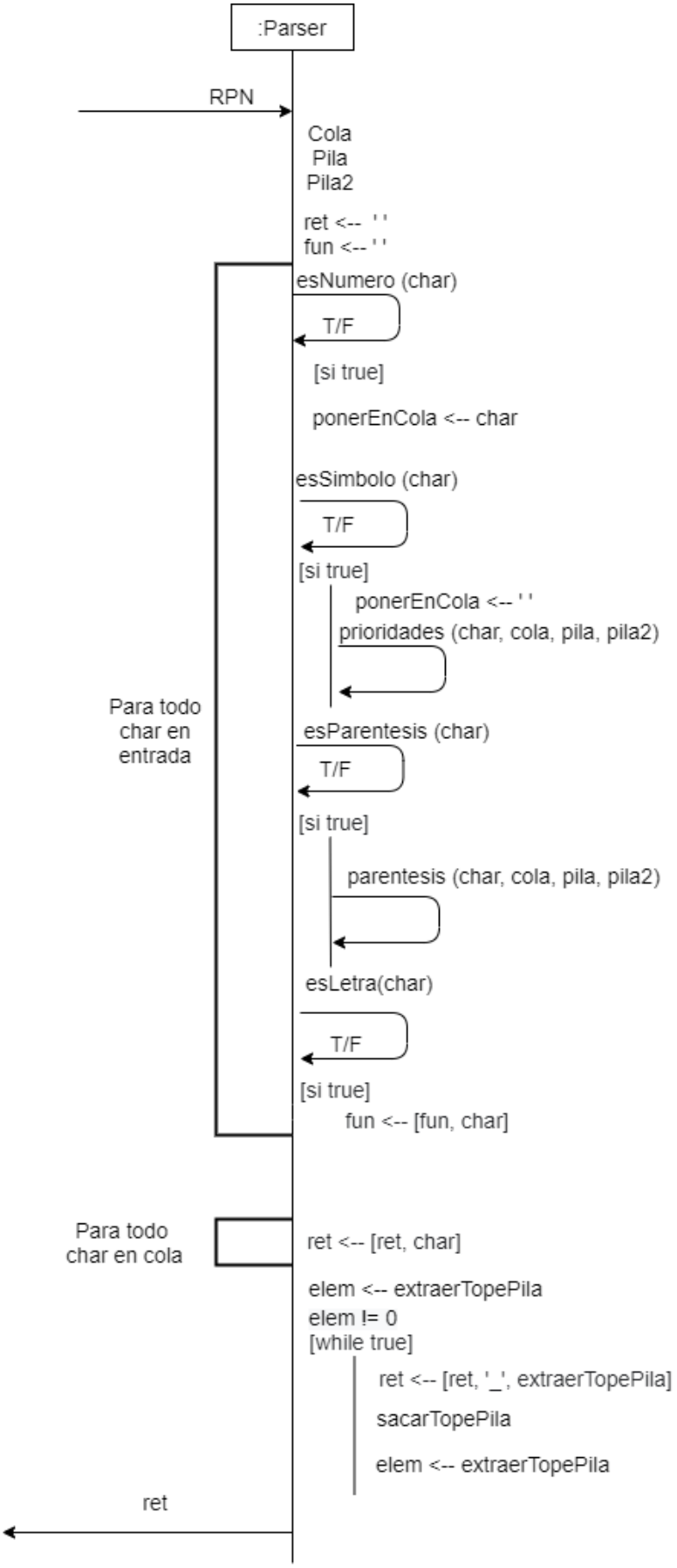
En este corto ejemplo se observa cómo se recorre toda la cadena tan solo una vez, evaluándola de izquierda a derecha. Cuando la pila se encuentra con un operador o con una función, opera con los primeros dos elementos del tope de la pila (operación con dos argumentos) o solamente con el tope (operación con un argumento, función). En el caso de que se encuentre una incógnita x en la cadena, el programa cambia su valor por el asignado por el usuario y lo coloca en la pila. El resultado de la operación es el único elemento que queda en el tope de la pila.

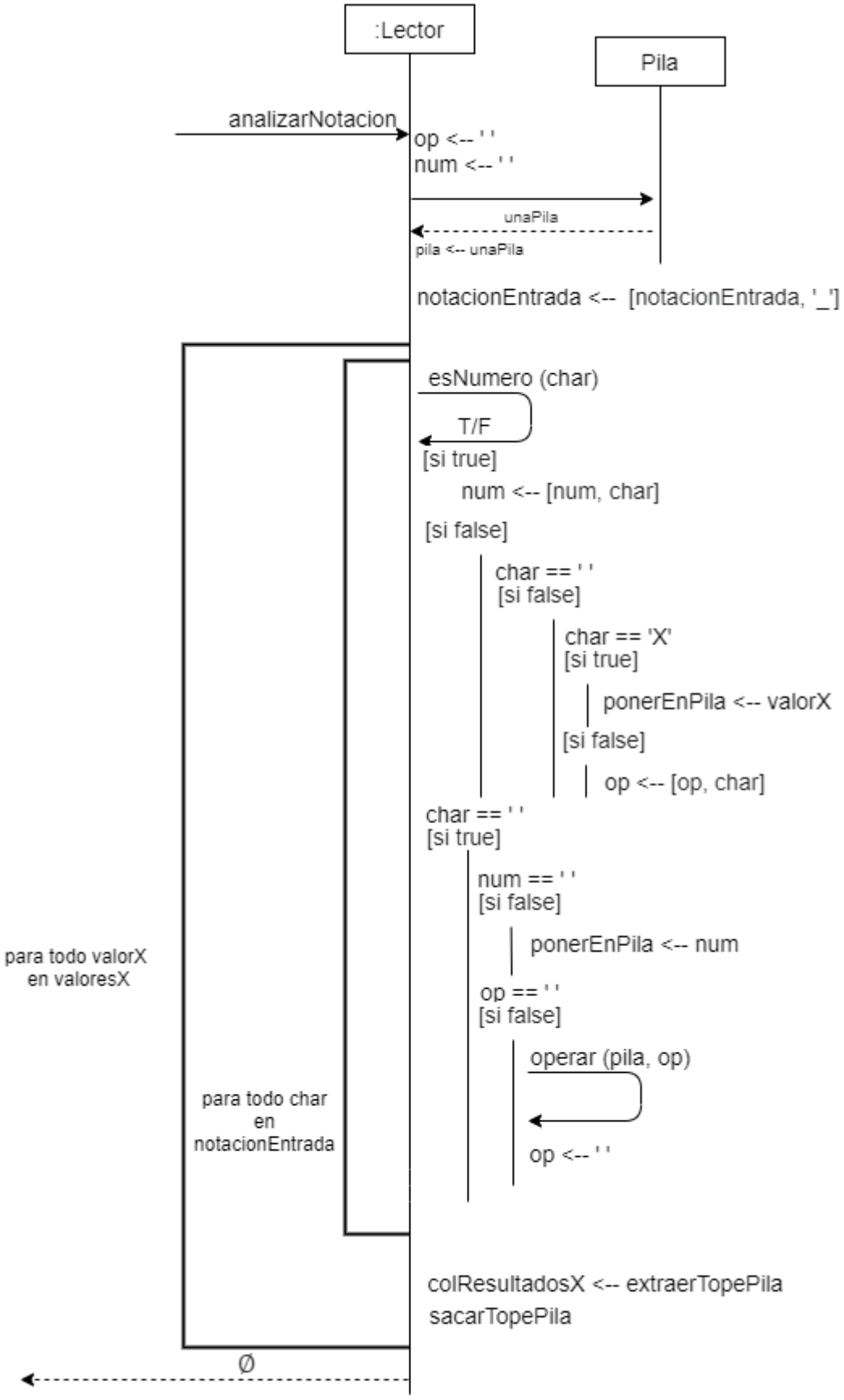
Lector no efectúa las operaciones, de esto se encarga la clase abstracta Operación, que está compuesta por sus dos subclases Doble y Simple. Tal como sus nombres indican, Doble se encarga de almacenar y operar con dos argumentos, por ejemplo la suma, resta, multiplicación, etc. Y Simple con un argumento, como la función coseno, seno, raíz cuadrada, etc. Dentro de Operación encontramos variables como el parámetro de la operación (siempre va a haber al menos uno), el resultado y la colección de símbolos, esta última va a variar dependiendo la subclase, tanto Doble como Simple tendrán símbolos distintos tales como los operadores “+”, “-“, etc. Y a funciones como “cos”, “sin”, “sqrt”, entre otras.

La clase Lector puede resolver para uno o más valores de X. En el caso para el valor único, concreta la operación anteriormente nombrada y almacena el resultado en la colección de resultados. En cambio, cuando evaluamos un intervalo, el constructor se encarga de crear todos los puntos (con ayuda de la variable h) y almacenarlos en la colección “valoresX”. A la hora de analizar la notación, operará todos los valores X y para cada uno almacenara los resultados en “colResultadosX”. Las dos colecciones tienen longitud similar, ya que para cada posición de un elemento de “valoresX” se le asigna la misma posición al resultado en “colResultadosX”. Por ejemplo, sea la colección de valores de X: (1 2 3) y la colección de resultados: (15 20 30), los puntos (x,y) de esta ecuación serían: (1,15), (2,20), (3,30). En definitiva, la colección de resultados almacena todos los valores de “Y”.

Este (o estos) resultado(s) ingresa(n) al sistema. Finalmente tenemos al módulo visual VentanaInicio, el cual tiene dependencia con Sistema. No solo permite ingresar datos sino que también comparte los resultados por pantalla de manera diversa. Mediante esta interfaz el usuario ingresa la ecuación y en base a eso decide si evaluar un solo valor de X o un intervalo de valores X. Si decide evaluar uno, ingresa el valor y el resultado aparece al instante, en cambio, si se desea evaluar el intervalo, se pide dato de intervalo y cantidad de puntos, con esto ya nos permite graficar la función. El gráfico se abre en una ventana aparte y nos encontramos con un mapa de ejes cartesianos en donde se ven incluidos los puntos designados por el usuario unido por rectas. Mediante pharo podemos acceder al programa copiando “VentanaInicio new openInWorld.” dentro del playground.


:Sistema
evaluarEn(unValor)
par, rpn, lec
valorX <-- unValor
:parser
traducirCadena(cadena)
unaCadena
rpn <-- unaCadena
:lector
setNotacionEntrada (rpn)
Ø
añadirValorX (valorX)
Ø
analizarNotacion
Ø
getColResultados
unaCol
resultado <-- unaCol at: 1
Ø


:Parser
RPN
Cola
Pila
Pila2
ret <-- ' '
fun <-- ' '
esNumero (char)
T/F
[si true]
ponerEnCola <-- char
esSimbolo (char)
T/F
[si true]
ponerEnCola <-- ' '
prioridades (char, cola, pila, pila2)
Para todo char en entrada
esParentesis (char)
T/F
[si true]
parentesis (char, cola, pila, pila2)
esLetra(char)
T/F
[si true]
fun <-- [fun, char]
Para todo char en cola
ret <-- [ret, char]
elem <-- extraerTopePila
elem != 0
[while true]
ret <-- [ret, '_', extraerTopePila]
sacarTopePila
elem <-- extraerTopePila
ret


:Lector
Pila
analizarNotacion
op <-- ' '
num <-- ' '
unaPila
pila <-- unaPila
notacionEntrada <-- [notacionEntrada, '_']
esNumero (char)
T/F
[si true]
num <-- [num, char]
[si false]
char == ' '
[si false]
char == 'X'
[si true]
ponerEnPila <-- valorX
[si false]
op <-- [op, char]
char == ' '
[si true]
num == ' '
[si false]
ponerEnPila <-- num
op == ' '
[si false]
operar (pila, op)
op <-- ' '
para todo valorX
en valoresX
para todo char
en
notacionEntrada
colResultadosX <-- extraerTopePila
sacarTopePila
Ø

Conclusión:

Este trabajo nos permite ver más allá de la programación orientada a objetos, ver cómo podemos resolver problemas de otra manera a la que estábamos acostumbrados. Estudiar la notación RPN nos permite operar ecuaciones complejas de la manera más eficaz permitiendo así una mejor performance en nuestro programa. A la hora de crear el sistema utilizando lenguaje smalltalk, logramos confeccionar soluciones más intuitivas gracias a la cantidad de recursos y métodos con los que cuenta el lenguaje.