UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Guatemala 16 de Enero de 2024
Adrián López - 231361
Luis Fernando Mendoza

Algoritmos y Estructura de Datos

Proyecto 1: Intérprete de LISP

Objetivos:

a. Utilizar el Java Collections Framework para desarrollo de aplicaciones.
b. Saber escoger estructuras de datos, con su complejidad en tiempo y espacio para la implementación
de aplicaciones.
c. Identificar los principales elementos y usos de la programación Funcional (Functional Programming).

"Desarrollo de un intérprete LISP para un subconjunto sencillo de instrucciones de alguno de los dos dialectos principales (Common LISP y Scheme). Deben implementarse como mínimo:
a) operaciones aritméticas.
b) instrucción QUOTE o ' (single quote, para interrumpir el proceso de evaluación de expresiones).
c) Definición de funciones (DEFUN).
d) SETQ.
e) predicados (ATOM, LIST, EQUAL, <, >).
f) condicionales (COND), nota: no es necesario implementar IF.
g) paso de parámetros. (un parámetro puede ser incluso una función)."

Evaluación de Expresiones: El programa puede evaluar expresiones Lisp proporcionadas por el usuario e identificar si son átomos o listas.

Operaciones Aritméticas: Permite realizar operaciones básicas como suma, resta, multiplicación y división entre números.

Comparaciones: Puede comparar valores numéricos para verificar si son iguales, mayores o menores que otros.

Verificación de Propiedades: Permite verificar si un valor es un átomo o una lista según la definición en Lisp.

¿Cómo funciona? El programa consta de varias clases que se encargan de diferentes aspectos de la evaluación de expresiones Lisp:

Main: La clase Main sirve como punto de entrada del programa. Coordina la interacción con el usuario, lee las expresiones a evaluar y las pasa al Interprete para su procesamiento.

Interprete: La clase Interprete interpreta las expresiones recibidas del usuario, realiza operaciones y gestiona variables y funciones. Utiliza instancias de las clases PrefixCalc, Operator, Tokenizer y otras estructuras de datos como HashMap y ArrayList.

Reader: La clase Reader se encarga de leer y analizar archivos para convertir su contenido en instrucciones comprensibles por el intérprete. Proporciona el método Parse para realizar esta tarea.

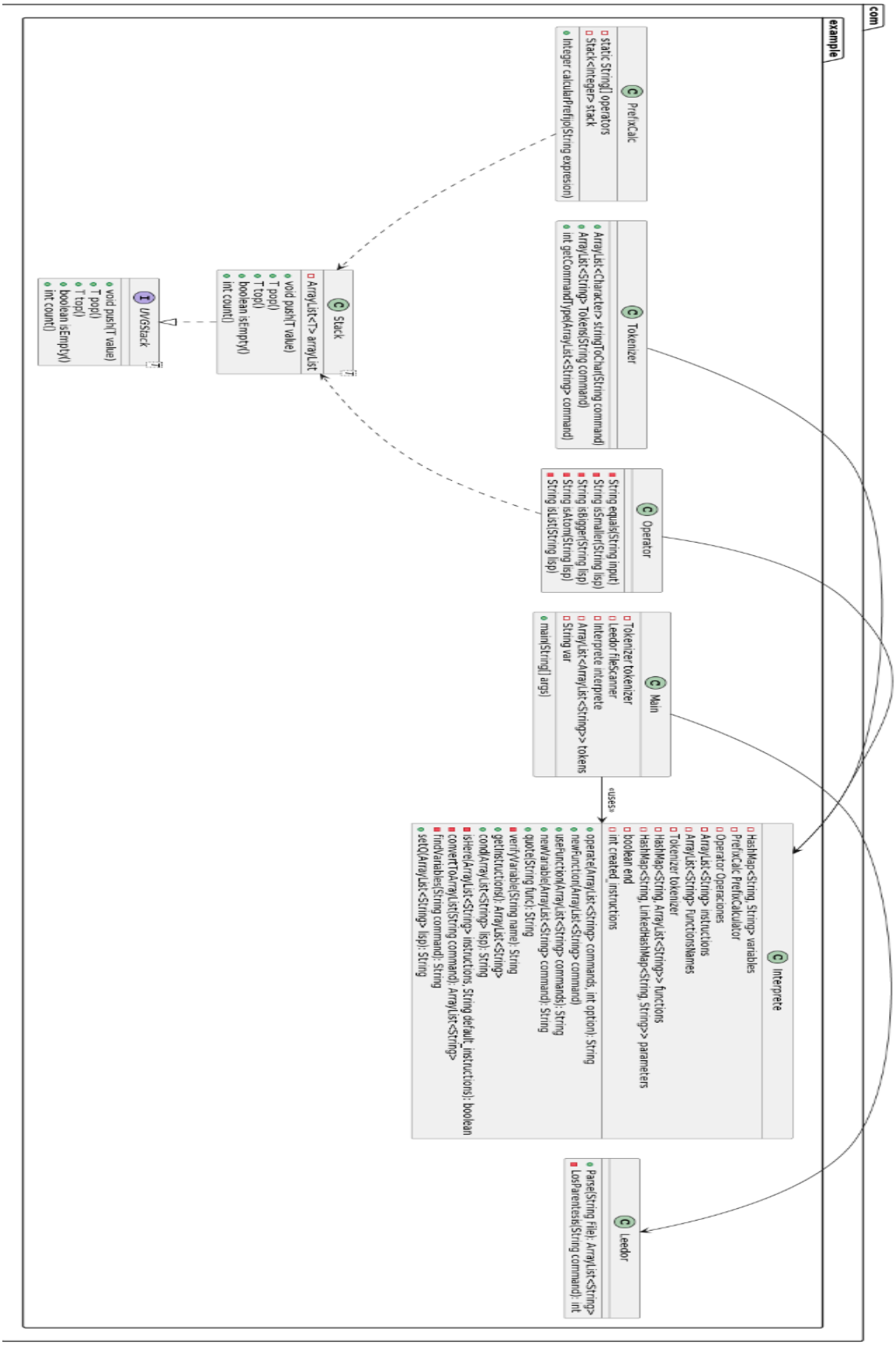
Operator: La clase Operator contiene métodos para realizar operaciones específicas, como comparar valores, verificar si un valor es un átomo o una lista en el contexto de Lisp.

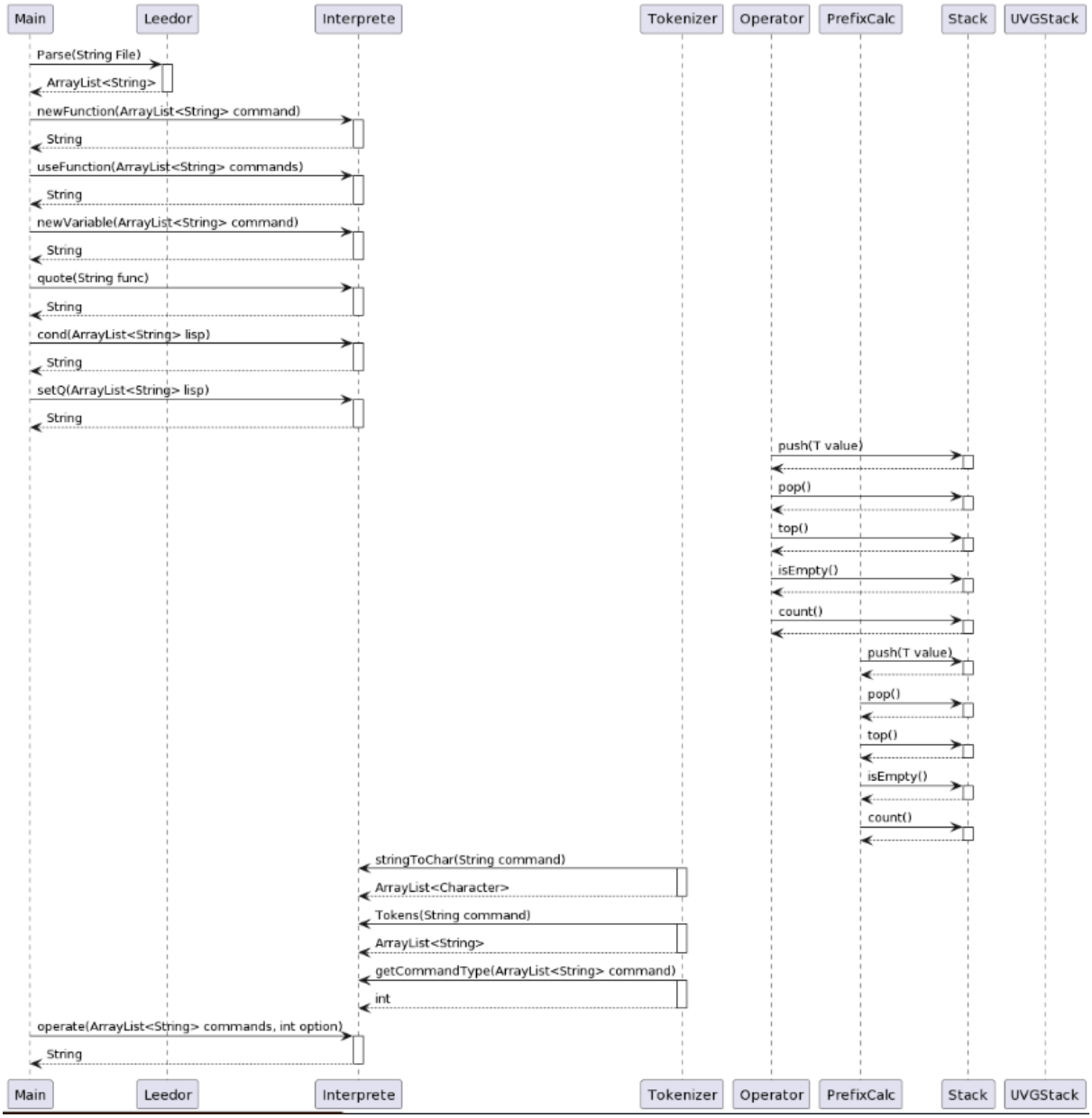
PrefixCalc: La clase PrefixCalc proporciona funcionalidad para calcular expresiones en notación prefija, como sumar, restar, multiplicar y dividir. Utiliza una estructura de datos Stack para realizar los cálculos.

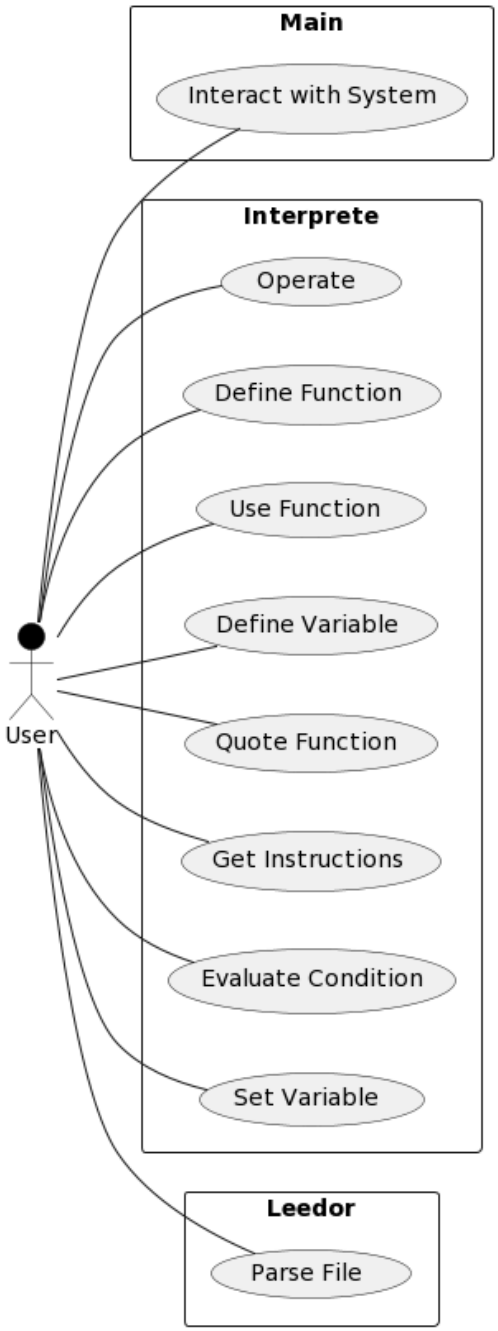
Stack: La clase Stack implementa una pila genérica utilizando un ArrayList y proporciona operaciones básicas de pila como push, pop, top, isEmpty y count.

Tokenizer: La clase Tokenizer se encarga de tokenizar comandos de Lisp y determinar su tipo de comando. Convierte comandos de texto en listas de tokens y proporciona métodos para esta tarea.

a. Diagramas diseño final de intérprete LISP:




Main
Leedor
Interprete
Tokenizer
Operator
PrefixCalc
Stack
UVGStack
Parse(String File)
ArrayList<String>
newFunction(ArrayList<String> command)
String
useFunction(ArrayList<String> commands)
String
newVariable(ArrayList<String> command)
String
quote(String func)
String
cond(ArrayList<String> lisp)
String
setQ(ArrayList<String> lisp)
String
push(T value)
pop()
top()
isEmpty()
count()
push(T value)
pop()
top()
isEmpty()
count()
stringToChar(String command)
ArrayList<Character>
Tokens(String command)
ArrayList<String>
getCommandType(ArrayList<String> command)
int
operate(ArrayList<String> commands, int option)
String
Main
Leedor
Interprete
Tokenizer
Operator
PrefixCalc
Stack
UVGStack



b. Control de versiones (GitHub):
https://github.com/adirnnn/Proyecto-1-Fase-2-LISP.git

c. Prueba unitaria JUnit:
Se han hecho JUnit tests para cada clase importante:

- InterpreteTest
- OperatorTest
- PrefixCalcTest
- StackTest
- TokenizerTest