Angel Antonio Armas Hernández - 24714
Julio Reynaldo Pellecer Morales- 241071
Anthony Emanuel Barrios de león -241301

Proyecto 1

Fase 1

Lisp fue uno de los primeros lenguajes de programación que se crearon. Fue desarrollado en 1958 por John McCarthy un informático de gran renombre en el tema de la inteligencia artificial. John McCarthy desarrolló este lenguaje de programación mientras trabajaba en un proyecto del MIT, necesitaba que las computadoras de la época fueran capaces de procesar grandes cantidades de datos numéricos e información, con la finaldad de poder imitar la capacidad humana de análisis de información o razonamiento. Su solución fue desarrollar un medio de comunicación entre el usuario y el equipo con el que pudiera darle instrucciones para ordenar los datos en un formato de listas para que el sistema pudiera posteriormente navegar entre un conjunto ordenado de listas donde se almacenaban diferentes tipos de datos para cumplir un requerimiento solicitado por el usuario, por ejemplo responder una pregunta.

El lenguaje de programación lisp basa su funcionamiento en el cálculo lambda, un concepto matemático utilizado en informática para describir el comportamiento de los elementos principales de la programación (las funciones y sus respectivas variables). El lenguaje de programación lisp implementa el cálculo lambda junto con otro concepto de programación conocido como meta-programación permitiendo que el sistema utilice el código que se desarrolle en este lenguaje como nuevos datos que le permiten realizar modificaciones usando una herramienta llamada macros que consisten en pequeños fragmentos de código que analizan el funcionamiento del código desarrollado en lisp y pueden modificar su estructura mientras este se encuentra en proceso de compilación.
La Java Collections Framework proporciona un conjunto de interfaces y clases que permiten gestionar colecciones de datos de manera eficiente. En el diseño del intérprete mostrado en el diagrama, se pueden aprovechar diversas estructuras de datos para optimizar la manipulación de tokens, nodos del árbol sintáctico y almacenamiento de cadenas.

Clases y su Relación con la Java Collections Framework:

1. Interfaz Expression

Define el contrato para cualquier expresión evaluable dentro del intérprete.

- Método:
    - evaluate(Environment env): Evalúa la expresión en un entorno dado y devuelve un resultado.

2. LiteralExpression

Angel Antonio Armas Hernández - 24714
Julio Reynaldo Pellecer Morales- 241071
Anthony Emanuel Barrios de león -241301

Representa valores literales como números o cadenas.

- Atributo:
  - value: almacena el valor literal.
- Métodos:
  - evaluate(Environment env): Retorna el valor literal.
  - toString(): Devuelve la representación en texto del valor.

3. SymbolExpression

Representa símbolos (identificadores de variables o nombres de funciones).

- Atributo:
  - name: almacena el nombre del símbolo.
- Métodos:
  - evaluate(Environment env):
    - Si name es una variable definida en env, retorna su valor.
    - Si no, retorna el nombre del símbolo.
  - getName(): Devuelve el nombre del símbolo.
  - toString(): Representación en texto.

4. ListExpression

Representa listas de expresiones.

- Atributo:
  - elements: lista de expresiones.
- Métodos:
  - evaluate(Environment env):
    - Si la lista está vacía, retorna una lista vacía.
    - Si el primer elemento es un símbolo, se trata de una función u operador y se ejecuta.
    - Si no es un operador conocido, evalúa cada elemento de la lista.
  - getElements(): Devuelve los elementos de la lista.
  - toString(): Devuelve la representación en texto de la lista.

5. Interfaz OperatorStrategy

Define una estrategia para operadores.

- Método:
  - execute(List<Expression> args, Environment env): Recibe una lista de argumentos y devuelve el resultado.

6. Clase Environment

Maneja variables, funciones y operadores.

Angel Antonio Armas Hernández - 24714
Julio Reynaldo Pellecer Morales- 241071
Anthony Emanuel Barrios de león -241301

- Atributos:
    - variables: almacena variables definidas.
    - functions: almacena funciones definidas por el usuario.
    - operators: almacena operadores predefinidos.
    - parent: referencia a un entorno padre (para anidamiento de entornos).
- Métodos clave:
    - registerBuiltInOperators(): Registra operadores (+, -, *, /, setq, quote, etc.).
    - isVariable(name), getVariable(name), setVariable(name, value): Manejo de variables.
    - isFunction(name), defineFunction(name, Function f), callFunction(name, args, callerEnv): Manejo de funciones.
    - getOperator(name): Retorna el operador si está definido.

7. Clase Function

Representa una función definida por el usuario.

- Atributos:
    - parameters: Lista de parámetros de la función.
    - body: Cuerpo de la función.
- Métodos:
    - getParameters(): Devuelve los parámetros.
    - getBody(): Devuelve el cuerpo de la función.

8. Clase LispParser

Convierte una cadena de entrada en una estructura de expresiones evaluables.

- Métodos clave:
    - parse(String input): Parsea la entrada y devuelve una Expression.
    - tokenize(String input): Divide la entrada en tokens.
    - parseExpression(List<String> tokens, int[] position): Convierte tokens en expresiones (LiteralExpression, SymbolExpression o ListExpression).

9. Clase LispInterpreter

Punto de entrada del intérprete.

- Atributos:
    - globalEnv: Entorno global.
    - parser: Instancia del parser.
- Métodos clave:
    - evaluate(String input): Evalúa una expresión LISP.
- Método main(String[] args):
    - Inicia un bucle interactivo (REPL).
    - Permite al usuario ingresar expresiones y ver los resultados.

Angel Antonio Armas Hernández - 24714
Julio Reynaldo Pellecer Morales- 241071
Anthony Emanuel Barrios de león -241301

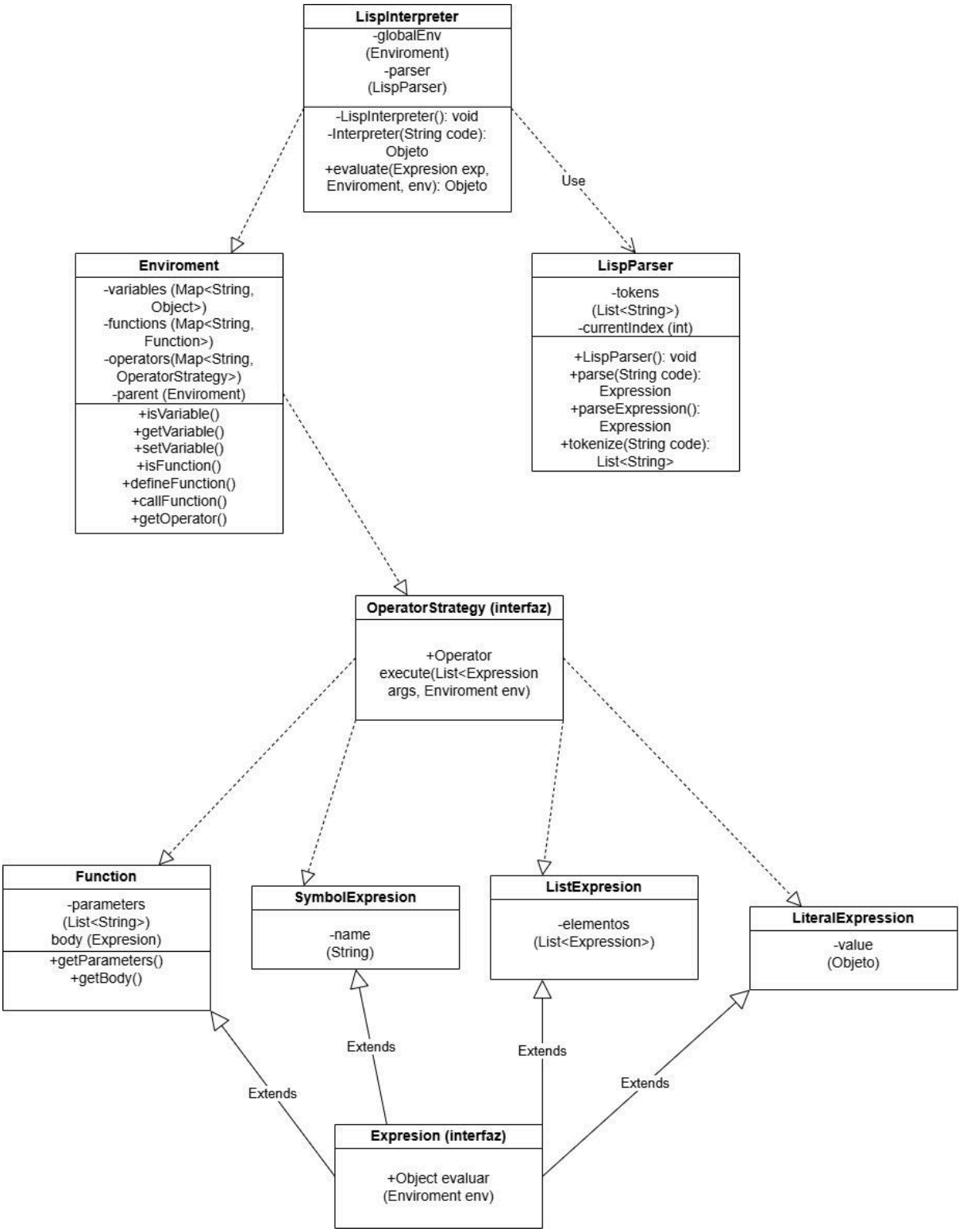
10. Requisitos funcionales

El intérprete soporta:

- Evaluación de expresiones numéricas (+, -, *, /).
- Definición y asignación de variables (setq).
- Evaluación de expresiones lógicas (<, >, equal).
- Definición de funciones (defun).
- Condiciones (if, cond).
- Funciones de lista (list, quote, ').
- Impresión en consola (print).
- Soporte para recursión (mediante defun y callFunction).

DIAGRAMA UML (abajo):

Angel Antonio Armas Hernández - 24714
Julio Reynaldo Pellecer Morales- 241071
Anthony Emanuel Barrios de león -241301



Angel Antonio Armas Hernández - 24714
Julio Reynaldo Pellecer Morales- 241071
Anthony Emanuel Barrios de león -241301

Diagrama de Secuencia: