UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Guatemala 22 de Marzo de 2024
Oscar Escribá - 21474

Algoritmos y estructura de datos

Entrega Proyecto Fase 2

Indicación de las estructuras del Java Collections Framework utilizadas, indicando la razón y referencias para su uso.

Las Java Collections Framework (JCF) son un conjunto esencial de herramientas en Java, proporcionando una variedad de clases e interfaces para manejar datos de manera efectiva. Estas estructuras de datos son vitales en el desarrollo de software, ya que simplifican la gestión de información al permitir operaciones como búsqueda, inserción, eliminación y recorrido de elementos de manera eficiente.
En resumen, las Java Collections son fundamentales en el desarrollo de software en Java porque simplifica la manipulación de datos, ofrecen eficiencia en el acceso y manipulación de datos, abstraen las estructuras de datos subyacentes, ofrecen flexibilidad y estandarización, y facilitan la interoperabilidad entre diferentes partes del programa.

Implementación:
En la implementación del intérprete de Lisp, se emplean diversas estructuras del Java Collections Framework para gestionar eficazmente distintos aspectos del código. Estas elecciones se basan en las necesidades específicas de acceso y organización de datos dentro del intérprete.

En la clase Environment, se utiliza un HashMap<String, Object> para almacenar variables y sus valores correspondientes. Esta elección se debe a que HashMap permite un acceso rápido a las variables a través de sus nombres, lo cual es esencial para la manipulación dinámica de variables durante la ejecución de expresiones Lisp.

Asimismo, en la misma clase, se emplea un HashMap<String>, FunctionExpression> para almacenar funciones definidas por el usuario y sus expresiones asociadas. De nuevo, la elección de HashMap se justifica por su eficiente acceso a las funciones a través de sus nombres, lo que resulta crucial para evaluar las llamadas a funciones en Lisp.

Por otro lado, en la clase ListExpression, se utiliza un ArrayList<Expression> para almacenar una lista de expresiones dentro de una expresión de lista en el código Lisp. Esta decisión se basa en la necesidad de un almacenamiento dinámico de elementos y un acceso rápido por índice para mantener el orden de las expresiones en la lista y acceder a ellas durante la evaluación.

En cuanto al análisis sintáctico realizado en la clase Parser, se emplea una List<Token> para almacenar los tokens generados por el lexer y procesarlos durante el análisis sintáctico. Se opta por List debido a la necesidad de mantener un orden específico de los tokens para este proceso, y List proporciona una estructura adecuada para mantener esta secuencia de tokens.

Por último, en la clase Lexer, se utiliza un StringBuilder para construir cadenas de caracteres de manera eficiente al procesar tokens. Esta elección se prefiere sobre la concatenación de cadenas directamente debido a la eficiencia que ofrece StringBuilder al construir cadenas en un bucle, evitando la creación innecesaria de múltiples objetos de cadena.

En resumen, estas estructuras del Java Collections Framework se emplean en el intérprete de Lisp para almacenar y manipular datos de manera eficiente, adaptándose a los requisitos específicos de acceso y orden necesarios para su correcto funcionamiento.

Descripción de cada una de las clases:

Lexer: La clase Lexer se encarga de analizar una cadena de entrada en Lisp y producir una secuencia de tokens. Estos tokens representan los componentes básicos de un programa Lisp, como paréntesis, números, símbolos y otros caracteres especiales.

Parser: La clase Parser se encarga de tomar la secuencia de tokens generada por el Lexer y construir una estructura de árbol que representa la sintaxis del programa Lisp. Para hacer esto, el Parser utiliza una técnica llamada análisis sintáctico recursivo descendente.

Token: La clase Token representa un token generado por el Lexer. Cada token tiene un tipo (como número, símbolo, paréntesis de apertura, paréntesis de cierre o fin de archivo) y un valor asociado que representa el contenido del token.

TokenType: La enumeración TokenType define los tipos de tokens que pueden ser generados por el Lexer, como símbolos, números, paréntesis de apertura, paréntesis de cierre y fin de archivo.

Expression: La interfaz Expression es una interfaz marcadora que indica que una clase es una expresión válida en Lisp. Las clases que implementan esta interfaz son utilizadas para representar diferentes tipos de expresiones en Lisp, como números, símbolos, listas y funciones.

ListExpression: La clase ListExpression representa una lista de expresiones en Lisp. Se utiliza para representar listas de valores, argumentos de funciones y otros elementos que requieren una estructura de lista.

SymbolExpression: La clase SymbolExpression representa un símbolo en Lisp. Se utiliza para representar nombres de variables, funciones y otros identificadores en un programa Lisp.

NumberExpression: La clase NumberExpression representa un número entero en Lisp. Se utiliza para representar valores numéricos en un programa Lisp.

FunctionExpression: La clase FunctionExpression representa una función definida por el usuario en Lisp. Contiene una lista de parámetros y una lista de expresiones que forman el cuerpo de la función.