Justificación:
de las estructuras del java Collections Framework

Saúl Contreras
Roberto Figueroa
Michele Benvenuto

Estructura de Datos
Segundo Ciclo
Intérprete de Lisp

A continuación se muestra los Java Collections Frameworks que fueron utilizados y la razón por la cual fueron utilizados en el Intérprete de Lisp realizado: El codigo para el intérprete se encuentra en:

https://github.com/suulcoder/LispInterpreter/

1. Map y HashMap: El map que permite instanciar un HashTable, fue utilizado para guardar las funciones tanto del Intérprete como las definidas por el usuario, y las variables tanto las propias del intérprete como las definidas por el usuario. De la interfaz Map se eligió HashMap sobre otras clases que implementan Map por las siguientes razones:

    1) usa los métodos equals y hashCode para identificar la llave. Lo cual nos beneficia para encontrar efectivamente la variable o la función.

    2) no mantiene un orden secuencial de los datos registrados, lo cual nos conviene ya que no necesitamos ese orden para nuestras variables y/o funciones.

2. List, Abstract List y Double Linked List: Se utilizó el Java Collection Framework List, para almacenar datos en secuencia, como los parámetros de una función o la función escrita por el usuario. Siendo la implementación de esta el Abstract List y la Double linked List, que fueron utilizadas por las siguientes razones:

1) Permite llamar tanto a los nodos anteriores y posteriores al actual.
2) Permite realizar iteraciones más eficientes que Circular list y Single Linked List..

3. Collection, Iterator y ListIterator: Estas tres Java Collection Framework fueron utilizadas para definir los métodos de la Double Linked List, La collection Iterator nos permite iterar como su nombre lo indica y se implementó ListIterator porque nos da ventajas para interar listas, en especial la double Linked List. Por otro lado Collection fue utilizado para obtener toda la colección de nodos.

4. BufferedReader: Está Collection Framework nos permite leer de una manera tipo buffer, es decir que se carga el contenido, parte por parte, para así poder trabajar con la información recibida.

5. IOException: Está collection nos permitió regresar al usuario el error obtenido y solicitarle a cualquier main que implemente nuestro intérprete que utilice un Try-Catch para implementarlo. Des esta manera aseguramos la seguridad de nuestro programa.

6. InputStreamReader: Está Collection Framework, nos permite leer la información proporcionada por el usuario.

7. Arrays: Fue utilizada sobre una ArrayList, o cualquier tipo de list, ya que sabíamos que no iba a mutar nuestros datos. Además fue utilizada sobre un List<T>, debido a que permite que no se consuma menos RAM durante el proceso.

8. BinaryOperator: Nos permitió definir las funciones independientes del intérprete como lo es la suma o la multiplicación, y agregarlos al hashmap de una manera eficiente. e introducir todo el método como un BinaryOperator.

9. UnaryOperator: Nos permitió definir variables, y poderlas diferenciar de un valor no definido.

Nota: Se utilizó el tipo Object, para luego identificar el tipo con el método propio de Java instanceOf, esto nos permitió hacer más efectivo el código.