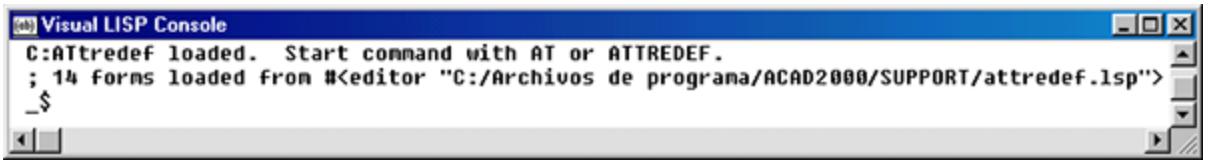
A. Investigación corta de Lisp, su historia, sus características, donde es empleado.

Historia de Lisp

Es un lenguaje que muestra la información estructurada en listas en las que se pueden gestionar la información que estas contienen. De ahí el nombre que se le dió a este lenguaje, Lisp (List-Processing), porque fue creado principalmente para el procesamiento de listas. Es un lenguaje funcional que se apoya en la utilización de funciones matemáticas para el control de los datos. Pero el elemento fundamental en el Lisp es la lista. Y desde el punto de vista más amplio del término. Cada función del lisp y cada programa que se genera con él vienen dado en forma de lista. Por esta razón los datos no se pueden diferenciar sintácticamente de los programas.
A este tipo de lenguaje se les denomina aplicativos o funcionales porque se basan en la aplicación de funciones a los datos. El lisp diferencia dos tipos de elementos básicos: El átomo, datos elementales de varios tipos como números, símbolos, caracteres y cadenas de caracteres. Y las Listas, entre las que podemos nombrar a una en especial, la lista “nil”, que es una lista nula que no tiene ningún elemento (EcuRed, 2011).
Las formas se *evalúan* (en relación con determinado contexto) para producir valores y efectos colaterales. Las funciones se invocan *aplicándolas* a argumentos. (Guy L. Steele Jr., Common Lisp the Language, capítulo 5). En el entorno Visual LISP se identifican como FORMAS las funciones de usuario definidas dentro de un fichero fuente LSP. Al cargar un programa se recibe la confirmación de la carga exitosa del mismo mediante el siguiente mensaje:



Las funciones son conocidas en matemáticas. Mediante ellas se describe algún tipo de relación entre un grupo de valores. La adición y la multiplicación son funciones simples. Una función más compleja sería: f(x)=3x2+2x-15
Esta última función también describe una regla de cálculo o algoritmo. Las funciones realizan los cálculos (en términos LISP, evaluación) utilizando para ello sus argumentos y devuelven un resultado. A iguales argumentos corresponderá iguales resultados.

En LISP, los programas se construyen a partir de la composición de funciones. Esto permite que tales programas expresen sus propósitos más claramente que los programas

convencionales, y que resulten más fáciles de entender y de mantener, además de ser más fáciles de construir.
AutoCAD identifica sus funciones LISP primitivas como el tipo de dato SUBR, mientras que las formas definidas a partir de ellas pertenecen al tipo de dato USUBR o "sub-rutina de usuario" La función type devuelve los tipos correspondientes. La llamada a una función toma la forma de una lista cuyo primer elemento es un átomo simbólico que representa a la función llamada. El resto de los elementos de esa lista pueden ser átomos y otras listas. Estas sublistas se consideran también llamadas a funciones y se evalúan para que su valor resultante pueda ser pasado como argumento a la función que las contiene.

FUNCIONES PRIMITIVAS

Describimos una serie de funciones como ‘primitivas’ en el sentido de que están definidas en la norma del lenguaje, para distinguirlas de las funciones creadas por el usuario a partir de aquéllas.
En LISP se llama a una función mediante la siguiente sintaxis:
(NOMBREFUNCION <Argumento_1> ... <Argumento_n>)
La suma de dos números sería (+ 5 1)
Para evaluarla LISP procede de la siguiente manera:
a. Lee la expresión completa (+ 5 1)
b. La interpreta como una llamada a una función y la identifica como SUMAR <+>
c. Interpreta 5 como primer argumento y 1 como segundo. El paréntesis de cierre le indica que no hay más argumentos.
d. La función <+> se evalúa para 5 y 1, devolviendo 6 como resultado, que a falta de otro destino es impreso en pantalla.

Programar LISP significa llamar a funciones. Básicamente esto se hace usando el tipo de dato LISTA. Cualquier lista que no tenga otra interpretación como forma especial se considerará una llamada a una función, donde el primer término se tomará como el nombre de la función y el resto como sus argumentos. Las listas de llamadas a función pueden estar anidadas, es decir, que una llamada a función se puede estar utilizando como argumento en otra lista que corresponda a una llamada a otra función. El resultado de la evaluación de cada nivel de anidación es devuelto al nivel de superior, hasta llegar al nivel más alto, cuyo valor devuelto se imprimiría en la pantalla de texto o la línea de comandos.

ARGUMENTOS FUNCIONALES

De lo expuesto más arriba se concluye que en LISP una función es además un objeto de datos que puede ser suministrado a otra función como argumento. Esta posibilidad contribuye a la facilidad con que LISP se puede adaptar a las necesidades de cualquier programa mediante la incorporación de nuevas funciones que en su comportamiento resultan idénticas a las primitivas.

Un programa que admite funciones como datos debe también suministrar alguna manera de invocarlas. Esto se logra en Visual LISP mediante la función APPLY.

APPLY
(apply función lista-args)

APPLY, como su nombre en inglés indica, aplica una función (que recibe como primer argumento) a una lista de argumentos. La función puede ser un objeto de código compilado, una expresión-lambda, o un símbolo. En este último caso se utiliza el valor funcional global de dicho símbolo, aunque éste no puede ser una forma especial.

```
_$ (apply '+ '(2 4 6))
12
_$ (apply '(lambda (x y z)(+ x y z)) '(2 4 6))
12
_$ (apply 'quote '(2 4 6))
; error: bad QUOTE syntax: ((QUOTE 2) (QUOTE 4) (QUOTE 6))
_1$
; reset after error
```

Obsérvese el error provocado por utilizar la forma especial QUOTE.

Otras muchas funciones LISP requieren argumentos funcionales. Entre las de uso más frecuente están las funciones de mapeado. MAPCAR, por ejemplo toma dos o más argumentos: una función y una o más listas (tantas como parámetros requiera la función) y aplica la función sucesivamente a los elementos de cada lista, devolviendo una lista con los resultados.

```
_$ (mapcar '+ '(1 2 3) '(10 100 1000))
(11 102 1003)
```

Otras muchas funciones admiten funciones como argumentos. Entre las más utilizadas tenemos VL-SORT y VL-REMOVE-IF. La primera es una función de ordenación de uso general. Requiere una lista y un predicado, y devuelve una lista ordenada pasando sus elementos dos a dos al predicado.

```
_$ (vl-sort '(3 2 1 3) '<)
(1 2 3)
```

VL-REMOVE-IF acepta una función y una lista, y devuelve todos los elementos de la lista para los cuales la función devuelve NIL (falso).

```
_$ (vl-remove-if 'numberp '("a" 3 4 "c" 5 "d" "e"))
("a" "c" "d" "e")
```

El programar nuevas funciones utilitarias que aceptan argumentos funcionales es una parte importante del estilo de programación funcional que caracteriza a LISP.

FUNCIONES ARITMÉTICAS BÁSICAS

Las funciones aritméticas que tienen como argumento un número devuelven distintos valores dependiendo de que el argumento proporcionado sea un número entero o real. Si todos los argumentos son enteros, el valor devuelto será entero. Por el contrario, si alguno o todos los argumentos son reales, el valor devuelto será un número real. Por ejemplo:
(/ 12 5) devuelve 2, mientras que (/ 12.0 5) devuelve 2.4
Los argumentos de una función aritmética no son necesariamente números. Cualquier otra función que devuelva un número como resultado es admisible como argumento de una función aritmética. Ejemplo (* (+ 1 5)(- 20 10))

FUNCIONES BÁSICAS DE TRATAMIENTO DE CADENAS

Para el tratamiento de cadenas tenemos funciones que permiten unificar cadenas diferentes, extraer subcadenas de una cadena mayor, determinar cuántos caracteres hay en una cadena y transformar los caracteres a mayúsculas o minúsculas. El predicado WCMATCH permite determinar la semejanza de cadenas utilizando comodines.

FUNCIONES BÁSICAS DE ACCESO A LISTAS

Una lista es una manera de representar un conjunto de átomos y de otras listas. Una lista tiene la forma de un paréntesis de apertura "(" seguido de una serie de átomos o listas, seguido por otro paréntesis de cierre ")". De manera que cualquier cosa encerrada entre paréntesis será considerada una lista. Una lista pasada al evaluador LISP será tratada como una expresión

simbólica (S-expresión), es decir, una llamada a función y se considerará el primer término de la lista como el nombre de la función. Para que una lista sea tratada como dato y no como una expresión simbólica debe estar contenida en la forma especial QUOTE. Las funciones básicas cuya comprensión es imprescindible para el acceso a la información contenida en listas se pueden reducir a cuatro: QUOTE, CAR, CDR, y NTH.

FUNCIONES DE CONSTRUCCIÓN DE LISTAS

Con las funciones del epígrafe anterior podemos descomponer listas, accediendo a sus componentes a distintos niveles de anidación. Existen otras funciones que podemos utilizar para componer nuevas listas a base de elementos individuales, ya sean átomos u otras listas. CONS, LIST y APPEND son funciones que construyen listas por distintos procedimientos y con resultados diversos, por lo que es necesario distinguirlas bien.

(Togores, 2019)

Características

- LISP posee un manejo de memoria automático que libera el espacio utilizado por los objetos que dejan de ser necesitados (EcuRed, 2011).
- Incluye un mecanismo bastante simple para utilizar evaluación perezosa de expresiones (EcuRed, 2011).
- LISP no posee un sistema de tipos estáticos como puede ocurrir en c/c++. LISP asocia los tipos a los valores en vez que a las variables, por ello los errores de mal uso de tipos solo puedan ser detectados en tiempo de ejecución y no de compilación (EcuRed, 2011).
- Las implementaciones en LISP suelen programarse de manera que cálculos iterativos puedan realizarse en un espacio constante (en memoria) aunque hayan sido descritos mediante el uso de la recursividad (EcuRed, 2011).
- LISP es interpretado y usa una estructura de gestión de almacenamiento en montículo con recolección de basura como almacenamiento primario para datos y programas (Desarrollo Web, 2004).

¿Donde es empleado?

Desde su concepción en 1955, Lisp fue el primer lenguaje de inteligencia artificial y, además, introdujo una serie de nuevos conceptos que, hoy en día, son de uso común como

la recursividad, los procesos por listas, las estructuras de datos en forma de árbol, el manejo de almacenamiento automático o los tipos dinámicos de datos. Las sentencias IF, THEN y ELSE que, prácticamente, vemos en cualquier lenguaje de programación son un invento de McCarthy para Lisp.

Desde sus inicios, Lisp estuvo estrechamente vinculado a los equipos de investigación en el campo de la inteligencia artificial, principalmente, en sistemas basados en el computador PDP-10. El lenguaje de programación Micro Planner, base del sistema SHRDLU de inteligencia artificial, está basado en el Lisp. La primera implementación práctica de Lisp se debe a Steve

Russel que fue capaz de implementar algunas funciones mediante el código máquina de un IBM 704 y, al final, desarrolló un intérprete Lisp plenamente funcional que podía evaluar expresiones Lisp. En 1974, Tim Hart y Mike Levin en MIT fueron capaces de desarrollar un compilador completo de Lisp que introdujo en Lisp la compilación incremental, es decir, la posibilidad de mezclar libremente funciones compiladas e interpretadas (Velasco, J , 2011).

B. Investigación corta sobre Java Collections Framework (especialmente la jerarquía de interfaces e implementaciones). Indicar cuáles se usarán en el proyecto, indicando cómo se emplearán.

Java Collections Framework o “estructura de colecciones de Java” se trata de un conjunto de clases e interfaces que mejoran notablemente las capacidades del lenguaje respecto a estructuras de datos. También se conoce a la librería de clases contenedoras de Java que podemos encontrar en el paquete estándar java.util. Además, constituyen un excelente ejemplo de aplicación de los conceptos propios de la programación orientada a objetos (Javincho, 2007).

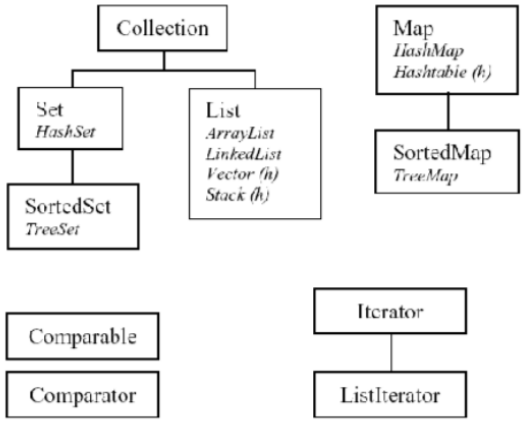


Imagen 1.

(Hans, 2019)

La imagen#1 nos presenta las jerarquías de interfaces y las clases que implemente el Java Collection Framework.

Interfaces de la JCF: Constituyen el elemento central de la JCF.

1. Collection: Define métodos para tratar una colección genérica de elementos
2. Set : Colección que no admite elementos repetidos
3. SortedSet : Set cuyos elementos se mantienen ordenados según el criterio establecido
4. List : Admite elementos repetidos y mantiene un orden inicial
5. Map: Conjunto de pares clave/valor, sin repetición de claves
6. SortedMap: Map cuyos elementos se mantienen ordenados según el criterio establecido.

Interfaces de soporte:

1. Iterator : Sustituye a la interfaz Enumeration. Dispone de métodos para recorrer una colección y para borrar elementos.
2. ListIterator : Deriva de Iterator y permite recorrer lists en ambos sentidos.
3. Comparable: Declara el método compareTo() que permite ordenar las distintas colecciones según un orden natural (String, Date, Integer, Double, …).
4. Comparator : Declara el método compare() y se utiliza en lugar de Comparable cuando se desea ordenar objetos no estándar o sustituir a dicha interfaz.

Clases de propósito general: Son las implementaciones de las interfaces de la JFC.

1. HashSet : Interfaz Set implementada mediante una hash table.
2. TreeSet : Interfaz SortedSet implementada mediante un árbol binario ordenado.
3. ArrayList : Interfaz List implementada mediante un array.
4. LinkedList : Interfaz List implementada mediante una lista vinculada.
5. HashMap: Interfaz Map implementada mediante una hash table.

(Hans, 2019)

Las interfaces que pensamos utilizar para nuestro proyecto podrían ser..

- SortedSet: Para llevar un orden e ir calculando.
- List: Mantiene el orden entre los cálculos utilizados dentro del programa.

Las interfaces de soporte que pensamos utilizar para nuestro proyecto podrían ser..

- Comparable: Hay un cálculo específico para cierto dato, el orden es lo más importante ya que si no no se confundieran los datos y operaciones.

Las clases de propósito general que pensamos utilizar en nuestro proyecto podrían ser..

- ArrayList: Es el concepto más básico y con el que ya hemos trabajado anteriormente.
- LinkedList: Este podría ser otra forma de probar algo nuevo e implementar más cambios y/o innovaciones al proyecto.

A. Bibliografía.

Cortés, Ulises y Sierra, Carlos. LISP. Editorial Marcombo, Barcelona-Máxico, 1987. pág. 44

Desarrollo Web. (2004). Qué es Lisp. 2 de febrero del 2020, Sitio web: https://desarrolloweb.com/articulos/1561.php

EcuRed. (2011). LISP. 2 de febrero del 2020, de EcuRed Sitio web: https://www.ecured.cu/LISP

Hans, A. (2019). Libro de estructura de datos en java, Apuntes de Estructuras de Datos y Algoritmos. 5 de febrero del 2020, de Docsity Sitio web: https://www.docsity.com/es/libro-de-estructura-de-datos-en-java/5078025/

Javincho. (2007). Estructura de Colecciones de Java, Apuntes de Programación Lineal. 5 de febrero del 2020, de Docsity Sitio web: https://www.docsity.com/es/estructura-de-colecciones-de-java/2755126/

Togores, R. (2019). 2.2. Funciones. 12 de Febrero del 2020, de Formas&Funciones Sitio Web: http://www.togores.net/vl/curso/lisp/bases/funciones

Velasco, J. (2011). Historia de la tecnología: Lisp. 2 febrero del 2020, de Hipertextual Sitio web: https://hipertextual.com/2011/10/historia-de-la-tecnologia-lisp