**UNIVERSIDAD DE CARTAGENA**

**PROGRAMA**

**INGENIERIA DE SOFTWARE**

**MATERIA**

**ESTRUCTURA DE DATOS**

**EVALUACION FINAL**

**DOCENTE**

**JHON ARRIETA ARRIETA**

**INTEGRANTES**

**JUAN CAMILO LENTINO**

**SERGIO LEMA**

**CARTAGENA DE INDIAS D.TYC**

**17-01-2023**

Tabla de contenido

[Introducción vi](#_Toc125046277)

[Objetivos vii](#_Toc125046278)

[Justificación viii](#_Toc125046279)

[Desarrollo 1](#_Toc125046280)

[Dictionary 1](#_Toc125046281)

[Definición y objetivo 1](#_Toc125046282)

[Versión de java aparecido 1](#_Toc125046283)

[Paquete al que pertenece 1](#_Toc125046284)

[Métodos 1](#_Toc125046285)

[Ejemplo practico 2](#_Toc125046286)

[HashTable 2](#_Toc125046287)

[Definición y objetivo 2](#_Toc125046288)

[Versión de java aparecido 3](#_Toc125046289)

[Paquete al que pertenece 3](#_Toc125046290)

[Métodos 3](#_Toc125046291)

[Ejemplo practico 4](#_Toc125046292)

[Property y Properties 4](#_Toc125046293)

[Definición y objetivo 4](#_Toc125046294)

[Versión de java aparecido 5](#_Toc125046295)

[Paquete al que pertenece 5](#_Toc125046296)

[Métodos 5](#_Toc125046297)

[Ejemplo practico 7](#_Toc125046298)

[Vector 7](#_Toc125046299)

[Definición y objetivo 7](#_Toc125046300)

[Versión de java aparecido 8](#_Toc125046301)

[Paquete al que pertenece 8](#_Toc125046302)

[Métodos 8](#_Toc125046303)

[Ejemplo practico 9](#_Toc125046304)

[Enumeration 10](#_Toc125046305)

[Definición y objetivo 10](#_Toc125046306)

[Versión de java aparecido 10](#_Toc125046307)

[Paquete al que pertenece 10](#_Toc125046308)

[Métodos 10](#_Toc125046309)

[Ejemplo practico 11](#_Toc125046310)

[Collection y Collections 11](#_Toc125046311)

[Definición y objetivo 11](#_Toc125046312)

[Versión de java aparecido 12](#_Toc125046313)

[Paquete al que pertenece 12](#_Toc125046314)

[Métodos 12](#_Toc125046315)

[Ejemplo practico 13](#_Toc125046316)

[Arrays 14](#_Toc125046317)

[Definición y objetivo 14](#_Toc125046318)

[Versión de java aparecido 14](#_Toc125046319)

[Paquete al que pertenece 14](#_Toc125046320)

[Métodos 14](#_Toc125046321)

[Ejemplo practico 15](#_Toc125046322)

[List 15](#_Toc125046323)

[Definición y objetivo 15](#_Toc125046324)

[Versión de java aparecido 16](#_Toc125046325)

[Paquete al que pertenece 16](#_Toc125046326)

[Métodos 16](#_Toc125046327)

[Ejemplo practico 17](#_Toc125046328)

[ArrayList 17](#_Toc125046329)

[Definición y objetivo 17](#_Toc125046330)

[Versión de java aparecido 17](#_Toc125046331)

[Paquete al que pertenece 18](#_Toc125046332)

[Métodos 18](#_Toc125046333)

[Ejemplo practico 19](#_Toc125046334)

[LinkedList 19](#_Toc125046335)

[Definición y objetivo 19](#_Toc125046336)

[Versión de java aparecido 20](#_Toc125046337)

[Paquete al que pertenece 20](#_Toc125046338)

[Métodos 20](#_Toc125046339)

[Ejemplo practico 21](#_Toc125046340)

[Set 21](#_Toc125046341)

[Definición y objetivo 21](#_Toc125046342)

[Versión de java aparecido 21](#_Toc125046343)

[Paquete al que pertenece 22](#_Toc125046344)

[Métodos 22](#_Toc125046345)

[Ejemplo practico 22](#_Toc125046346)

[SortedSet 23](#_Toc125046347)

[Definición y objetivo 23](#_Toc125046348)

[Versión de java aparecido 23](#_Toc125046349)

[Paquete al que pertenece 23](#_Toc125046350)

[Métodos 23](#_Toc125046351)

[Ejemplo practico 24](#_Toc125046352)

[HashSet 24](#_Toc125046353)

[Definición y objetivo 24](#_Toc125046354)

[Versión de java aparecido 24](#_Toc125046355)

[Paquete al que pertenece 25](#_Toc125046356)

[Métodos 25](#_Toc125046357)

[Ejemplo practico 25](#_Toc125046358)

[TreeSet 26](#_Toc125046359)

[Definición y objetivo 26](#_Toc125046360)

[Versión de java aparecido 26](#_Toc125046361)

[Paquete al que pertenece 26](#_Toc125046362)

[Métodos 26](#_Toc125046363)

[Ejemplo practico 27](#_Toc125046364)

[Map 27](#_Toc125046365)

[Definición y objetivo 27](#_Toc125046366)

[Versión de java aparecido 27](#_Toc125046367)

[Paquete al que pertenece 28](#_Toc125046368)

[Métodos 28](#_Toc125046369)

[Ejemplo practico 29](#_Toc125046370)

[HashMap 29](#_Toc125046371)

[Definición y objetivo 29](#_Toc125046372)

[Versión de java aparecido 29](#_Toc125046373)

[Paquete al que pertenece 29](#_Toc125046374)

[Métodos 29](#_Toc125046375)

[Ejemplo practico 30](#_Toc125046376)

[TreeMap 31](#_Toc125046377)

[Definición y objetivo 31](#_Toc125046378)

[Versión de java aparecido 31](#_Toc125046379)

[Paquete al que pertenece 31](#_Toc125046380)

[Métodos 31](#_Toc125046381)

[Ejemplo practico 32](#_Toc125046382)

[LinkedHashMap 33](#_Toc125046383)

[Definición y objetivo 33](#_Toc125046384)

[Versión de java aparecido 33](#_Toc125046385)

[Paquete al que pertenece 33](#_Toc125046386)

[Métodos 33](#_Toc125046387)

[Ejemplo practico 35](#_Toc125046388)

[Ejercicios prácticos con List, Set y Map 35](#_Toc125046389)

[Síntesis y argumentación individual 36](#_Toc125046390)

[Bibliografía 37](#_Toc125046391)

# Introducción

El presente trabajo consta de una serie de ejercicios y mucha investigación a nivel teórico sobre los distintos tipos de colecciones que tiene el lenguaje de programación java, mediante pequeños y sencillos ejercicios se demuestra el uso de estas clases y cómo funcionan algunos de sus métodos principales permitiéndonos conocer, ampliar, practicar y explorar nuevas herramientas que pueden ser útiles dependiendo de los diferentes tipos de situaciones que se nos presenten en problemas futuros de nuestro día a día como desarrolladores

# Objetivos

El presente trabajo tiene como objetivo dar a conocer, entender y practicar con los diferentes tipos de herramientas que nos ofrece la utilería de java, trabajar a un nivel básico con estas mismas y poder ver sus diferentes usos para poder en futuras ocasiones ver si es factible implementarlas y hacer uso de estas mismas.

# Justificación

Para el presente trabajo se realizó una investigación detallada en diferentes sitios de cada una de las clases o interfaces y se practicó mucho con ejercicios viendo su uso y al final se crearon pequeños y muy sencillos ejercicios productos de la imaginación donde se puede ver el uso de diferentes métodos que componen estas clases o interfaces y un caso de uso práctico del uso de estas mismas.

# Desarrollo

## Dictionary

### Definición y objetivo

La clase Dictionary en Java es una interfaz que proporciona una estructura de datos de diccionario genérica. El objetivo de esta clase es permitir el almacenamiento de pares clave-valor, donde las claves son únicas y los valores pueden ser recuperados mediante su clave correspondiente. Esta clase se utiliza a menudo para implementar algoritmos de búsqueda y tablas de hash.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la interfaz Dictionary es Java 1.0

### Paquete al que pertenece

La clase Dictionary pertenece al paquete java.util en Java.

### Métodos

Los métodos de la interfaz Dictionary en Java incluyen:

enumitems(): Devuelve los valores de enumeración del diccionario.

get Object (Object key): Devuelve este diccionario en el que la clave está asignada a los valores.

boolean isEmpty(): Prueba de este diccionario se asigna al valor de la clave no existe.

enum keys(): Devuelve una enumeración de las claves de diccionario.

put Object(Object key, Object value): Mapea la clave especificada en este diccionario se especifica valor.

Object remove (Object key): Elimina de esta clave de diccionario (y su valor correspondiente).

int size(): Devuelve el número (claves distintas) las entradas del diccionario.

### Ejemplo practico

El ejemplo se encuentra en el paquete Dictionary en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y a sus métodos.

## HashTable

### Definición y objetivo

La clase Hashtable en Java es una implementación de la interfaz Map, que proporciona una estructura de datos de tabla hash genérica. El objetivo de esta clase es permitir el almacenamiento de pares clave-valor, donde las claves son únicas y los valores pueden ser recuperados mediante su clave correspondiente. Los elementos son almacenados en una tabla hash, lo que permite una rápida recuperación de los valores mediante su clave. La clase Hashtable es similar a la clase HashMap, pero es segura para subprocesos.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase HashTable es Java 1.0

### Paquete al que pertenece

La clase HashTable pertenece al paquete java.util en Java.

### Métodos

Los métodos de la clase Hashtable en Java incluyen:

void clear(): elimina todos los elementos de la tabla hash.

Object clone(): devuelve una copia superficial de la tabla hash.

boolean contains(Object value): determina si la tabla hash contiene algún valor específico.

boolean containsKey(Object key): determina si la tabla hash contiene una clave específica.

boolean containsValue(Object value): determina si la tabla hash contiene algún valor específico.

Enumeration elements(): devuelve una enumeración de los valores de la tabla hash.

Object get(Object key): devuelve el valor asociado a la clave especificada.

boolean isEmpty(): determina si la tabla hash está vacía.

Enumeration keys(): devuelve una enumeración de las claves de la tabla hash.

Object put(Object key, Object value): agrega un par clave-valor a la tabla hash.

Object remove(Object key): elimina el par clave-valor de la tabla hash.

int size(): devuelve el número de pares clave-valor en la tabla hash.

Collection values(): devuelve una colección de los valores de la tabla hash.

### Ejemplo practico

El ejemplo se encuentra en el paquete HashTable en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y a sus métodos.

## Property y Properties

### Definición y objetivo

La clase Properties es una subclase de Hashtable que se utiliza para almacenar pares clave-valor en formato de cadena. Esta clase se utiliza a menudo para almacenar configuraciones de aplicaciones o para pasar parámetros a un programa a través de un archivo de propiedades.

La clase Property es una clase auxiliar que proporciona métodos estáticos para trabajar con archivos de propiedades. Por ejemplo, tiene métodos para cargar y guardar archivos de propiedades en disco, y también tiene métodos para obtener y establecer valores específicos en un objeto Properties.

En resumen, Properties se utiliza para almacenar un conjunto de pares clave-valor en formato de cadena, mientras que Property proporciona métodos para trabajar con archivos de propiedades.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase Property y Properties es Java 1.0

### Paquete al que pertenece

La clase Property y Properties pertenece al paquete java.util en Java.

### Métodos

La clase Properties tiene los siguientes métodos comunes:

void load(InputStream stream): Carga las propiedades desde un flujo de entrada.

void store(OutputStream out, String comments): Almacena las propiedades en un flujo de salida, opcionalmente con un comentario.

setProperty(String key, String value): Establece el valor de una propiedad dada su clave.

getProperty(String key): Obtiene el valor de una propiedad dada su clave.

stringPropertyNames(): Devuelve un conjunto de todas las claves de las propiedades en el objeto Properties.

clear(): Elimina todas las propiedades del objeto Properties.

Properties(): Crea un nuevo objeto Properties con ninguna propiedad.

Properties(Properties defaults): Crea un nuevo objeto Properties con las propiedades predeterminadas especificadas.

Enumeration<?> propertyNames(): Devuelve un enumerado de todas las claves de las propiedades en el objeto Properties.

void list(PrintStream out): Imprime las propiedades y sus valores en un flujo de salida.

void list(PrintWriter out): Imprime las propiedades y sus valores en un escritor de salida.

void loadFromXML(InputStream in): Carga las propiedades desde un flujo de entrada en formato XML.

void storeToXML(OutputStream os, String comment): Almacena las propiedades en un flujo de salida en formato XML, opcionalmente con un comentario.

void storeToXML(OutputStream os, String comment, String encoding): Almacena las propiedades en un flujo de salida en formato XML, opcionalmente con un comentario y una codificación específica.

La clase Property tiene los siguientes métodos comunes:

load(InputStream stream): Carga las propiedades desde un flujo de entrada.

store(OutputStream out, String comments): Almacena las propiedades en un flujo de salida, opcionalmente con un comentario.

setProperty(String key, String value): Establece el valor de una propiedad dada su clave.

getProperty(String key): Obtiene el valor de una propiedad dada su clave.

stringPropertyNames(): Devuelve un conjunto de todas las claves de las propiedades en el objeto Properties.

clear(): Elimina todas las propiedades del objeto Properties.

Enumeration<?> propertyNames(): Devuelve un enumerado de todas las claves de las propiedades en el objeto Properties..

### Ejemplo practico

El ejemplo se encuentra en el paquete Property\_Properties en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y a sus métodos.

## Vector

### Definición y objetivo

La clase Vector en Java es una clase de la biblioteca estándar de Java que proporciona una implementación de una lista de objetos que pueden cambiar de tamaño dinámicamente. Los objetos en un vector se almacenan en orden y se pueden acceder mediante un índice numérico.

La clase Vector proporciona una serie de métodos para agregar, eliminar, insertar, buscar y recuperar elementos de la lista, así como para obtener información sobre el tamaño, capacidad y estado de la lista.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase Vector es Java 1.0

### Paquete al que pertenece

La clase Vector pertenece al paquete java.util en Java.

### Métodos

La clase Vector tiene los siguientes métodos comunes:

add(E e): Añade un elemento al final del vector.

add(int index, E element): Inserta un elemento en la posición especificada.

addElement(E obj): Añade un elemento al final del vector.

capacity(): Devuelve la capacidad actual del vector.

clear(): Elimina todos los elementos del vector.

clone(): Devuelve una copia superficial del vector.

contains(Object o): Devuelve true si el vector contiene el elemento especificado.

elementAt(int index): Devuelve el elemento en la posición especificada.

elements(): Devuelve un enumerado de los elementos del vector.

firstElement(): Devuelve el primer elemento del vector.

get(int index): Devuelve el elemento en la posición especificada.

indexOf(Object o): Devuelve la posición del primer elemento especificado en el vector o -1 si no se encuentra.

insertElementAt(E obj, int index): Inserta un elemento en la posición especificada.

isEmpty(): Devuelve true si el vector está vacío.

lastElement(): Devuelve el último elemento del vector.

remove(int index): Elimina el elemento en la posición especificada.

remove(Object o): Elimina la primera ocurrencia del elemento especificado en el vector.

removeAllElements(): Elimina todos los elementos del vector.

removeElement(Object obj): Elimina la primera ocurrencia del elemento especificado en el vector.

removeElementAt(int index): Elimina el elemento en la posición especificada.

set(int index, E element): Reemplaza el elemento en la posición especificada con el elemento especificado.

setElementAt(E obj, int index): Reemplaza el elemento en la posición especificada con el elemento especificado.

setSize(int newSize): Establece el tamaño del vector.

size(): Devuelve el número de elementos en el vector.

toArray(): Devuelve un arreglo que contiene todos los elementos del vector en el orden correcto.

### Ejemplo practico

El ejemplo se encuentra en el paquete Vector en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## Enumeration

### Definición y objetivo

La clase Enumeration en Java proporciona una forma de recorrer los elementos de una colección, como un vector o una lista, de manera similar a un iterador. Sin embargo, a diferencia de un iterador, una enumeración no permite eliminar elementos de una colección mientras se está recorriendo. El objetivo de la clase Enumeration es proporcionar una forma segura y sencilla de recorrer los elementos de una colección.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase Enumeration es Java 1.0

### Paquete al que pertenece

La clase Vector pertenece al paquete java.util en Java.

### Métodos

Los métodos de la clase Enumeration incluyen:

hasMoreElements(): devuelve verdadero si hay más elementos en la enumeración.

nextElement(): devuelve el siguiente elemento de la enumeración.

hasMoreElements() y nextElement() son los dos métodos más comunes utilizados al trabajar con objetos Enumeration.

Además de estos métodos, Enumeration también hereda los métodos de la clase Object, como equals(), hashCode() y toString().

### Ejemplo practico

El ejemplo se encuentra en el paquete Enumeration en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## Collection y Collections

### Definición y objetivo

La clase Collection es una interfaz en Java que define un conjunto de métodos que deben ser implementados por las clases que la implementan. Esta interfaz proporciona una manera estandarizada de trabajar con colecciones de elementos en Java, como Listas, Conjuntos y Mapas. Por ejemplo, proporciona métodos para agregar y eliminar elementos de una colección, así como para determinar si un elemento está contenido en una colección.

La clase Collections es una clase de utilidad en Java que proporciona métodos estáticos útiles para trabajar con colecciones. Por ejemplo, proporciona métodos para ordenar y buscar elementos en una colección, así como para crear colecciones inmutables y sincronizadas.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase Collection y la interfaz Collections es Java 1.2

### Paquete al que pertenece

La clase Collection y la interfaz Collections pertenecen al paquete java.util en Java.

### Métodos

La interfaz Collection define los siguientes métodos:

int size(): devuelve el número de elementos en la colección.

boolean isEmpty(): devuelve verdadero si la colección está vacía, de lo contrario, falso.

boolean contains(Object o): devuelve verdadero si la colección contiene el elemento especificado, de lo contrario, falso.

Iterator iterator(): devuelve un iterador para recorrer los elementos en la colección.

Object[] toArray(): devuelve un arreglo que contiene todos los elementos de la colección en el orden en que son devueltos por el iterador.

boolean add(E e): agrega el elemento especificado a la colección.

boolean remove(Object o): elimina el primer elemento que sea igual al elemento especificado de la colección si existe.

void clear(): elimina todos los elementos de la colección.

La clase Collections tiene un gran número de métodos estáticos, algunos de los cuales son:

sort(List<T> list): ordena los elementos de la lista especificada en orden natural.

binarySearch(List<? extends Comparable<? super T>> list, T key): realiza una búsqueda binaria de la clave especificada en la lista especificada.

reverse(List<?> list): invierte el orden de los elementos en la lista especificada.

shuffle(List<?> list): desordena aleatoriamente los elementos en la lista especificada.

max(Collection<? extends T> coll, Comparator<? super T> comp): devuelve el elemento máximo en la colección especificada según el comparador especificado.

min(Collection<? extends T> coll, Comparator<? super T> comp): devuelve el elemento mínimo en la colección especificada según el comparador especificado.

copy(List<? super T> dest, List<? extends T> src): copia los elementos de la lista de origen especificada a la lista de destino especificada.

### Ejemplo practico

El ejemplo se encuentra en el paquete Collection\_Collections en los archivos CollectionMain.java y CollectionsMain.java, estos archivos contienen un ejemplo sencillo donde se hace uso de estas clases y tienen comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a estas clases y sus métodos.

## Arrays

### Definición y objetivo

La clase Arrays en Java proporciona métodos estáticos para trabajar con arreglos. la clase Arrays es una de las clases utilitarias que ofrece el lenguaje Java, que permite realizar operaciones básicas sobre arreglos, como ordenarlos, buscar elementos, compararlos, llenarlos, copiarlos y mostrarlos.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase Arrays es Java 1.2

### Paquete al que pertenece

La clase Arrays pertenecen al paquete java.util en Java.

### Métodos

La clase Arrays tiene varios métodos estáticos para trabajar con arreglos, algunos de los más comunes son:

sort(T[] a): Ordena los elementos de un arreglo de forma ascendente según su orden natural.

sort(T[] a, Comparator<? super T> c): Ordena los elementos de un arreglo según el comparador especificado.

binarySearch(T[] a, T key): Busca un elemento específico en un arreglo ordenado, utilizando el algoritmo de búsqueda binaria.

equals(T[] a, T[] a2): Compara dos arreglos y determina si son iguales.

fill(T[] a, T val): Llena un arreglo con un valor específico.

copyOf(T[] original, int newLength): Crea una copia de un arreglo en otro de un tamaño específico.

toString(T[] a): Devuelve una cadena de caracteres que representa los elementos de un arreglo.

asList(T... a): Devuelve una lista vista del arreglo especificado.

hashCode(T[] a): devuelve un hash code para el arreglo especificado.

deepEquals(Object[] a1, Object[] a2): devuelve true si dos arreglos profundamente iguales.

deepHashCode(Object[] a): devuelve un hash code para el arreglo especificado..

### Ejemplo practico

El ejemplo se encuentra en el paquete Arrays en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## List

### Definición y objetivo

La interfaz List en Java es una de las interfaces más utilizadas para trabajar con colecciones de datos. Proporciona una estructura de datos de lista, que es similar a un arreglo pero con capacidades adicionales.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase List es Java 1.2

### Paquete al que pertenece

La clase List pertenecen al paquete java.util en Java.

### Métodos

La interfaz List define varios métodos para trabajar con listas, algunos de los más comunes son:

add(E e): Agrega un elemento al final de la lista.

add(int index, E element): Inserta un elemento en una posición específica de la lista.

remove(int index): Elimina el elemento en una posición específica de la lista.

get(int index): Devuelve el elemento en una posición específica de la lista.

size(): Devuelve el número de elementos en la lista.

clear(): Elimina todos los elementos de la lista.

contains(Object o): Devuelve verdadero si la lista contiene el elemento especificado.

indexOf(Object o): Devuelve el índice de la primera ocurrencia del elemento especificado en la lista.

### Ejemplo practico

El ejemplo se encuentra en el paquete List en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## ArrayList

### Definición y objetivo

La clase ArrayList en Java, es una clase que permite almacenar datos en memoria de forma similar a los Arrays, con la ventaja de que el número de elementos que almacena, lo hace de forma dinámica, es decir, que no es necesario declarar su tamaño como pasa con los Arrays.

Un ArrayList se trata de un arreglo en el cual se en listan referencias a objetos mediante índices, el índice es la posición que ocupa un determinado objeto, como en todo arreglo la posición inicial o índice es 0. Cuando se agregan elementos a un objeto ArrayList su capacidad aumenta de forma automática.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase ArrayList es Java 1.2

### Paquete al que pertenece

La clase ArrayList pertenecen al paquete java.util en Java.

### Métodos

La clase ArrayList define varios métodos para trabajar con listas, algunos de los más comunes son:

add(E e): Agrega un elemento al final de la lista.

add(int index, E element): Inserta un elemento en una posición específica de la lista.

get(int index): Devuelve el elemento en una posición específica de la lista.

remove(int index): Elimina el elemento en una posición específica de la lista.

set(int index, E element): Asigna un elemento específico a una posición específica de la lista, reemplazando el elemento anterior.

clear(): Elimina todos los elementos de la lista.

isEmpty(): Devuelve verdadero si la lista está vacía, falso de lo contrario.

size(): Devuelve el número de elementos en la lista.

contains(Object o): Devuelve verdadero si la lista contiene el elemento especificado, falso de lo contrario.

indexOf(Object o): Devuelve el índice de la primera ocurrencia del elemento especificado en la lista, o -1 si no se encuentra.

ensureCapacity(int minCapacity): Aumenta la capacidad de la lista si es necesario para asegurar que pueda almacenar al menos el número especificado de elementos.

trimToSize(): Ajusta el tamaño del arreglo subyacente para que sea igual al número de elementos en la lista.

clone(): Devuelve una copia superficial de la lista.

toArray(): Devuelve un arreglo que contiene todos los elementos en la lista en el orden correcto.

subList(int fromIndex, int toIndex): Devuelve una vista de la porción de la lista entre las posiciones especificadas.

### Ejemplo practico

El ejemplo se encuentra en el paquete ArrayList en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## LinkedList

### Definición y objetivo

La clase LinkedList es una implementación de la interfaz List en Java. El objetivo principal de esta clase es proporcionar una estructura de datos de lista enlazada. A diferencia de ArrayList, LinkedList almacena los elementos en una estructura de datos enlazada, donde cada elemento de la lista apunta al siguiente y al anterior elemento en lugar de estar almacenado en un arreglo.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase LinkedList es Java 1.2

### Paquete al que pertenece

La clase LinkedList pertenecen al paquete java.util en Java.

### Métodos

La clase LinkedList tiene varios métodos, algunos de los más comunes son:

void add(E element): agrega el elemento especificado al final de esta lista.

void addFirst(E element): inserta el elemento especificado al principio de esta lista.

void addLast(E element): agrega el elemento especificado al final de esta lista.

void clear(): vacía la lista.

Object clone(): devuelve una copia superficial de esta lista.

boolean contains(Object o): devuelve true si esta lista contiene el elemento especificado.

E getFirst(): devuelve el primer elemento de esta lista.

E getLast(): devuelve el último elemento de esta lista.

int indexOf(Object o): devuelve el índice de la primera ocurrencia del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.

int lastIndexOf(Object o): devuelve el índice de la última ocurrencia del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.

E remove(): elimina y devuelve el primer elemento de esta lista.

E remove(int index): elimina el elemento en la posición especificada en esta lista.

E removeFirst(): elimina y devuelve el primer elemento de esta lista.

boolean removeLast(): elimina y devuelve el último elemento de esta lista.

E set(int index, E element): reemplaza el elemento en la posición especificada en esta lista con el elemento especificado.

int size(): devuelve el número de elementos en esta lista.

Object[] toArray(): devuelve un array que contiene todos los elementos en esta lista en orden secuencial.

### Ejemplo practico

El ejemplo se encuentra en el paquete LinkedList en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## Set

### Definición y objetivo

El objetivo de la interfaz Set en Java es proporcionar un conjunto de elementos únicos, es decir, que no se permiten elementos duplicados. Un conjunto es una colección que no tiene elementos repetidos.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la interface Set es Java 1.2

### Paquete al que pertenece

La interface Set pertenecen al paquete java.util en Java.

### Métodos

La interface Set tiene varios métodos, algunos de los más comunes son:

add(E e): agrega el elemento especificado al conjunto si aún no está presente.

clear(): elimina todos los elementos de este conjunto.

contains(Object o): devuelve true si este conjunto contiene el elemento especificado.

isEmpty(): devuelve true si este conjunto no contiene elementos.

iterator(): devuelve un iterador sobre los elementos de este conjunto.

remove(Object o): elimina el elemento especificado de este conjunto si está presente (si se permite la eliminación).

size(): devuelve el número de elementos en este conjunto.

toArray(): devuelve un array que contiene todos los elementos en este conjunto en orden arbitrario.

### Ejemplo practico

El ejemplo se encuentra en el paquete Set en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## SortedSet

### Definición y objetivo

El objetivo de la clase SortedSet en Java es proporcionar un conjunto ordenado de elementos únicos. Es decir, un conjunto que no permite elementos duplicados y mantiene el orden natural de los elementos.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase SortedSet es Java 1.2

### Paquete al que pertenece

La clase SortedSet pertenecen al paquete java.util en Java.

### Métodos

La clase SortedSet tiene varios métodos, algunos de los más comunes son:

first(): devuelve el primer elemento del conjunto.

last(): devuelve el último elemento del conjunto.

subSet(E fromElement, E toElement): devuelve una vista del conjunto que contiene elementos desde el elemento especificado hasta el elemento especificado.

headSet(E toElement): devuelve una vista del conjunto que contiene elementos menores al elemento especificado.

tailSet(E fromElement): devuelve una vista del conjunto que contiene elementos mayores o iguales al elemento especificado.

comparator(): devuelve el comparador utilizado para ordenar el conjunto, o null si el orden natural de los elementos es utilizado.

### Ejemplo practico

El ejemplo se encuentra en el paquete SortedSet en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## HashSet

### Definición y objetivo

El objetivo de la clase HashSet en Java es proporcionar una implementación de la interfaz Set que utiliza una tabla de hash para almacenar los elementos. Esto permite un acceso rápido a los elementos del conjunto y no permite elementos duplicados.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase HashSet es Java 1.2

### Paquete al que pertenece

La clase HashSet pertenecen al paquete java.util en Java.

### Métodos

La clase HashSet tiene varios métodos, algunos de los más comunes son:

add(E e): agrega el elemento especificado al conjunto si aún no está presente.

clear(): elimina todos los elementos de este conjunto.

contains(Object o): devuelve true si este conjunto contiene el elemento especificado.

isEmpty(): devuelve true si este conjunto no contiene elementos.

iterator(): devuelve un iterador sobre los elementos de este conjunto.

remove(Object o): elimina el elemento especificado de este conjunto si está presente (si se permite la eliminación).

size(): devuelve el número de elementos en este conjunto.

toArray(): devuelve un array que contiene todos los elementos en este conjunto en orden arbitrario.

### Ejemplo practico

El ejemplo se encuentra en el paquete HashSet en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## TreeSet

### Definición y objetivo

El objetivo de la clase TreeSet en Java es proporcionar una implementación de la interfaz SortedSet que utiliza un árbol binario para almacenar los elementos. Esto permite un acceso rápido a los elementos del conjunto y mantiene el orden natural de los elementos.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase TreeSet es Java 1.2

### Paquete al que pertenece

La clase TreeSet pertenecen al paquete java.util en Java.

### Métodos

La clase TreeSet tiene varios métodos, algunos de los más comunes son:

first(): devuelve el primer elemento del conjunto.

last(): devuelve el último elemento del conjunto.

subSet(E fromElement, E toElement): devuelve una vista del conjunto que contiene elementos desde el elemento especificado hasta el elemento especificado.

headSet(E toElement): devuelve una vista del conjunto que contiene elementos menores al elemento especificado.

tailSet(E fromElement): devuelve una vista del conjunto que contiene elementos mayores o iguales al elemento especificado.

comparator(): devuelve el comparador utilizado para ordenar el conjunto, o null si el orden natural de los elementos es utilizado.

descendingSet(): Devuelve una vista inversa del conjunto.

pollFirst(): Devuelve y elimina el primer elemento del conjunto.

pollLast(): Devuelve y elimina el último elemento del conjunto.

### Ejemplo practico

El ejemplo se encuentra en el paquete TreeSet en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## Map

### Definición y objetivo

El objetivo de la interfaz Map en Java es proporcionar una estructura de datos para almacenar pares clave-valor, donde cada clave es única y se utiliza para acceder al valor correspondiente.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase Map es Java 1.2

### Paquete al que pertenece

La clase Map pertenecen al paquete java.util en Java.

### Métodos

La clase Map tiene varios métodos, algunos de los más comunes son:

void clear(): elimina todos los elementos de la mapa.

boolean containsKey(Object key): devuelve verdadero si la mapa contiene la clave especificada.

boolean containsValue(Object value): devuelve verdadero si la mapa contiene el valor especificado.

Set<Map.Entry<K,V>> entrySet(): devuelve un conjunto de entradas (claves y valores) en la mapa.

V get(Object key): devuelve el valor asociado a la clave especificada.

boolean isEmpty(): devuelve verdadero si la mapa está vacía.

Set<K> keySet(): devuelve un conjunto de las claves en la mapa.

V put(K key, V value): agrega una entrada (clave y valor) a la mapa.

void putAll(Map<? extends K,? extends V> m): agrega todas las entradas de otra mapa a esta mapa.

V remove(Object key): elimina la entrada con la clave especificada de la mapa.

int size(): devuelve el número de entradas (claves y valores) en la mapa.

Collection<V> values(): devuelve una colección de los valores en la mapa.

### Ejemplo practico

El ejemplo se encuentra en el paquete Map en el archivo Contactos.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## HashMap

### Definición y objetivo

El objetivo de la clase HashMap en Java es proporcionar una implementación de la interfaz Map que utiliza una tabla hash para almacenar pares clave-valor. Un HashMap permite nulos y no se encuentra ordenada.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase HashMap es Java 1.2

### Paquete al que pertenece

La clase HashMap pertenecen al paquete java.util en Java.

### Métodos

La clase TreeSet tiene varios métodos, algunos de los más comunes son:

void clear(): Elimina todos los elementos de la mapa.

Object clone(): Devuelve una copia superficial de esta HashMap.

boolean containsKey(Object key): Devuelve verdadero si la mapa contiene la clave especificada.

boolean containsValue(Object value): Devuelve verdadero si la mapa contiene el valor especificado.

Set<Map.Entry<K,V>> entrySet(): Devuelve un conjunto de entradas (claves y valores) en la mapa.

V get(Object key): Devuelve el valor asociado a la clave especificada.

V getOrDefault(Object key, V defaultValue): Devuelve el valor asociado a la clave especificada o el valor por defecto si no se encuentra.

boolean isEmpty(): Devuelve verdadero si la mapa está vacía.

Set<K> keySet(): Devuelve un conjunto de las claves en la mapa.

V put(K key, V value): Agrega una entrada (clave y valor) a la mapa.

void putAll(Map<? extends K,? extends V> m): Agrega todas las entradas de otra mapa a esta mapa.

V remove(Object key): Elimina la entrada con la clave especificada de la mapa.

int size(): Devuelve el número de entradas (claves y valores) en la mapa.

Collection<V> values(): Devuelve una colección de los valores en la mapa.

### Ejemplo practico

El ejemplo se encuentra en el paquete HashMap en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## TreeMap

### Definición y objetivo

El objetivo de la clase TreeMap en Java es proporcionar una implementación de la interfaz SortedMap (que extiende a Map) que almacena sus elementos en un árbol de búsqueda binaria. Los elementos se almacenan en orden natural o según un comparador proporcionado, lo que permite una eficiente implementación de operaciones como get() y put() con un tiempo de acceso logarítmico (O(log n)) en promedio.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la clase TreeMap es Java 1.2

### Paquete al que pertenece

La clase TreeMap pertenecen al paquete java.util en Java.

### Métodos

La clase TreeMap tiene varios métodos, algunos de los más comunes son:

void clear(): Elimina todos los elementos de la mapa.

Object clone(): Devuelve una copia superficial de esta TreeMap.

boolean containsKey(Object key): Devuelve verdadero si la mapa contiene la clave especificada.

boolean containsValue(Object value): Devuelve verdadero si la mapa contiene el valor especificado.

Set<Map.Entry<K,V>> entrySet(): Devuelve un conjunto de entradas (claves y valores) en la mapa.

V get(Object key): Devuelve el valor asociado a la clave especificada.

V getOrDefault(Object key, V defaultValue): Devuelve el valor asociado a la clave especificada o el valor por defecto si no se encuentra.

boolean isEmpty(): Devuelve verdadero si la mapa está vacía.

Set<K> keySet(): Devuelve un conjunto de las claves en la mapa.

V put(K key, V value): Agrega una entrada (clave y valor) a la mapa.

void putAll(Map<? extends K,? extends V> m): Agrega todas las entradas de otra mapa a esta mapa.

V remove(Object key): Elimina la entrada con la clave especificada de la mapa.

int size(): Devuelve el número de entradas (claves y valores) en la mapa.

Collection<V> values(): Devuelve una colección de los valores en la mapa.

### Ejemplo practico

El ejemplo se encuentra en el paquete TreeMap en el archivo Main.java, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

## LinkedHashMap

### Definición y objetivo

El objetivo de la clase LinkedHashMap en Java es proporcionar una implementación de la interfaz Map que utiliza una tabla hash para almacenar pares clave-valor, así como una lista doblemente enlazada para recordar el orden en que se insertaron los elementos. Esto permite un tiempo de acceso constante (O(1)) para operaciones como get() y put() y mantener el orden de inserción de los elementos.

### Versión de java aparecido

La versión de Java en la que apareció por primera vez la interface LinkedHashMap es Java 1.4

### Paquete al que pertenece

La clase TreeMap pertenecen al paquete java.util en Java.

### Métodos

La clase LinkedHashMap tiene varios métodos, algunos de los más comunes son:

void clear(): Elimina todos los elementos de la mapa.

Object clone(): Devuelve una copia superficial de esta LinkedHashMap.

boolean containsKey(Object key): Devuelve verdadero si la mapa contiene la clave especificada.

boolean containsValue(Object value): Devuelve verdadero si la mapa contiene el valor especificado.

Set<Map.Entry<K,V>> entrySet(): Devuelve un conjunto de entradas (claves y valores) en la mapa.

V get(Object key): Devuelve el valor asociado a la clave especificada.

V getOrDefault(Object key, V defaultValue): Devuelve el valor asociado a la clave especificada o el valor por defecto si no se encuentra.

boolean isEmpty(): Devuelve verdadero si la mapa está vacía.

Set<K> keySet(): Devuelve un conjunto de las claves en la mapa.

V put(K key, V value): Agrega una entrada (clave y valor) a la mapa.

void putAll(Map<? extends K,? extends V> m): Agrega todas las entradas de otra mapa a esta mapa.

remove(Object key): Elimina la entrada con la clave especificada de la mapa.

int size(): Devuelve el número de entradas (claves y valores) en la mapa.

Collection<V> values(): Devuelve una colección de los valores en la mapa.

void forEach(BiConsumer<? super K, ? super V> action): Realiza una acción para cada entrada en la mapa.

boolean removeEldestEntry(Map.Entry<K,V> eldest): Define la política para eliminar la entrada más antigua cuando la mapa se llena.

### Ejemplo practico

El ejemplo se encuentra en el paquete LinkedHashMap en el archivo Main.java con un archivo Libro.java auxiliar, este archivo contiene un ejemplo sencillo donde se hace uso de esta clase y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran el uso que se le da a esta clase y sus métodos.

# Ejercicios prácticos con List, Set y Map

Todos los ejercicios prácticos desde el punto A al punto M se encuentran en el paquete List\_Set\_Map con archivos PuntoA.java, PuntoB.java, PuntoC.java, PuntoD.java, PuntoE.java, PuntoF.java, PuntoG.java, PuntoH.java, PuntoI.java, PuntoJ.java, PuntoK.java, PuntoL.java y PuntoM.java. Cada uno de estos archivos contiene la solución a lo solicitado en el punto y tiene comentarios tanto en el código como en la ejecución del mismo que demuestran la correcta solución a los problemas y ejercicios planteados.

# 

# Síntesis y argumentación individual

Conocer los diferentes tipos de herramientas como son estos tipos de clases o interfaces nos ayuda mucho a ampliar nuestro stack y el conocerlas y haber trabajado con ellas nos permite tener un mejor panorama, ya que sabiendo esto y el alcance o problema que estas herramientas resuelven podremos implementarlas según la necesidad que tengamos, sin mencionar que estaríamos dominando nuevas estructuras de datos que nos permitirán mejorar nuestra lógica y ser mejores desarrolladores.

# 

# Bibliografía

<https://www.w3big.com/es/java/java-dictionary-class.html#gsc.tab=0>

https://lineadecodigo.com/java/usar-una-hashtable-java/#:~:text=Una%20Hashtable%20Java%20es%20una,%C3%ADndice%20de%20nuestra%20Hashtable%20Java.

http://dis.um.es/~lopezquesada/documentos/IES\_1516/IAW/curso/UT3/ActividadesAlumnos/grupo9/16.html#:~:text=Un%20vector%20es%20similar%20a,elementos%20entre%20otros%20dos%20existentes.

https://es.stackoverflow.com/questions/72644/cu%C3%A1l-es-la-funci%C3%B3n-de-la-clase-collections-en-java

http://dis.um.es/~lopezquesada/documentos/IES\_1516/IAW/curso/UT3/ActividadesAlumnos/grupo7/pdf/punto16.pdf

https://refactorizando.com/metodos-hashmap-java/#:~:text=Un%20HashMap%20en%20Java%20es,y%20no%20se%20encuentra%20ordenada.