**ESTRUCTURA DE DATOS ACT. 4**

John Jairo Pérez Peña

Universidad de Cartagena

Programa de Ingeniería de Software

**Tabla de Contenidos**

[Introducción 1](#_Toc140247312)

[Objetivos 2](#_Toc140247313)

[Generales 2](#_Toc140247314)

[Específicos 2](#_Toc140247315)

[Justificación 3](#_Toc140247316)

[Desarrollo API Colecciones de Java 4](#_Toc140247317)

[Clase Dictionary 5](#_Toc140247318)

[Clase HashTable 7](#_Toc140247319)

[Clase Properties 9](#_Toc140247320)

[Clase Vector 11](#_Toc140247321)

[Interfaz Enumeration 15](#_Toc140247322)

[Interfaz Collection 15](#_Toc140247323)

[Clase Collections 18](#_Toc140247324)

[Clase Arrays 20](#_Toc140247325)

[Interfaz List 22](#_Toc140247326)

[Clase ArrayList 25](#_Toc140247327)

[Clase LinkedList 27](#_Toc140247328)

[Interfaz Set 30](#_Toc140247329)

[Interfaz SortedSet 32](#_Toc140247330)

[Clase HashSet 34](#_Toc140247331)

[Clase TreeSet 36](#_Toc140247332)

[Interfaz Map 39](#_Toc140247333)

[Clase HashMap 42](#_Toc140247334)

[Clase TreeMap 44](#_Toc140247335)

[Clase LinkedHashMap 47](#_Toc140247336)

[Interfaz Stream 49](#_Toc140247337)

[Desarrollo Creación de un Programa 51](#_Toc140247338)

[Lista de referencias 52](#_Toc140247339)

**Lista de tablas**

**Tabla** 1 Métodos de la clase Dictionary 5

**Tabla 2** Metodos de la clase HashTable 7

**Tabla 3** Metodos de la clase Properties 9

**Tabla 4** Metodos de la clase Vector 11

**Tabla 5** Metodos de la interfaz Enumeration 15

**Tabla 6** Metodos de la interfaz Collection 16

**Tabla 7** Metodos de la clase Collections 18

**Tabla 8** Metodos de la clase Arrays 20

**Tabla 9** Métodos de la interfaz List 22

**Tabla 10** Métodos de la clase ArrayList 25

**Tabla 11** Métodos de la clase LinkedList 28

**Tabla 12** Métodos de la interfaz Set 31

**Tabla 13** Métodos de la interfaz SortedSet 33

**Tabla 14** Métodos de la clase HashSet 35

**Tabla 15** Métodos de la clase TreeSet 37

**Tabla 16** Métodos de la interfaz Map 40

**Tabla 17** Metodos de la clase HashMap 43

**Tabla 18** Metodos de la clase TreeMap 45

**Tabla 19** Metodos de la clase LinkedHashMap 48

**Tabla 20** Metodos de la interfaz Stream 50

**Lista de figuras**

**Figura 1**. Ejemplo en código de la clase Dictionary 6

**Figura 2**. Ejemplo en código de la clase HashTable 8

**Figura 3**. Ejemplo en código de la clase Properties 10

**Figura 4**. Ejemplo en código de la clase Vector 14

**Figura 5.** Ejemplo en código de la interfaz Enumeration 15

**Figura 6.** Ejemplo en código de la interfaz Collection 17

**Figura 7**. Ejemplo en código de la clase Collections. 19

**Figura 8.** Ejemplo en código de la clase Arrays 21

**Figura 9.** Ejemplo en código de la interfaz List 24

**Figura 10**. Ejemplo en código de la clase ArrayList 27

**Figura 11.** Ejemplo en código de la clase LinkedList 30

**Figura 12.** Ejemplo en código de la interfaz Set 32

**Figura 13.** Ejemplo en código de la interfaz SortedSet 34

**Figura 14.** Ejemplo en código de la clase HashSet 36

**Figura 15.** Ejemplo en código de la clase HashSet 39

**Figura 16.** Ejemplo en código de la interfaz Map 42

**Figura 17.** Ejemplo en codigo de la clase HashMap 44

**Figura 18.** Ejemplo en codigo de la clase TreeMap 47

**Figura 19.** Ejemplo en codigo de la clase LinkedHashMap 49

**Figura 20.** Ejemplo en codigo de la interfaz Stream 51

# Introducción

El presente trabajo ilustra una amplia investigación sobre las distintas estructuras de datos que nos suministra Java, haciendo énfasis en las librerías proporcionada por el JDK (Java Development Kit) definiendo las posibles aplicaciones y sus objetivos, así como la dotación de un trasfondo histórico y una breve implementación de las interfaces o clases.

En el marco de la investigación se ha explorado el API de Colecciones de Java, el cual es una poderosa herramienta que ofrece una amplia gama de estructuras de datos y algoritmos para manipular y almacenar colecciones de objetos. Estas colecciones son utilizadas en numerosas aplicaciones y desempeñan un papel crucial en el desarrollo de software eficiente y escalable.

La información que se suministrará en este documento es su totalidad parte de la documentación oficial de Java dotada por Oracle, donde se exponen todas las clases y sus respectivos usos, importancia y optimizaciones.

# Objetivos

## Generales

* Conocer y analizar el API de Colecciones de Java, y sus aplicaciones en la manipulación y almacenamiento de datos.

## Específicos

* Investigar el API de Colecciones de Java para comprender sus características y funcionalidades.
* Analizar las interfaces y clases clave dentro del API de Colecciones, incluyendo su propósito y casos de uso.
* Conocer la evolución histórica del API de Colecciones.
* Describir los métodos proporcionados por el API de Colecciones y sus aplicaciones.
* Demostrar el uso de una clase o interfaz específica del API de Colecciones mediante un ejemplo práctico y original.
* Diferenciar las situaciones en las que es óptimo utilizar una estructura de datos en comparación con otra, en el contexto del API de Colecciones de Java.

# Justificación

La investigación que se realizó con respecto a las Colecciones de la API de Java es sumamente importante para entender la jerarquía que manejan las clases en Java, así como el funcionamiento interno de los métodos de obtención de datos en listas, vectores, diccionarios, maps, entre otros. Teniendo en cuenta que se propuso la generación de tablas, para la visualización de los métodos, el aprendizaje y la forma de entender el funcionamiento de estas es mas evidente y eficaz, los ejemplos son claros ya que comparten entre ellos una similitud y un avance en el rendimiento dependiendo de a que factor o sistema se quiera aplicar, por ello es importante resaltar que no siempre existe una única opción para realizar un tratamiento a los datos, ya sea ordenamiento, búsqueda o en sí la manipulación completa.

En este documento se presentan la mayor parte de las clases e interfaces del paquete java.util, en donde desde la versión primordial de java se encuentran estas estructuras de datos y hasta el día de hoy siguen siendo totalmente funcionales y actualizadas.

# Desarrollo API Colecciones de Java

La API o Framework de Colecciones de Java viene siendo un conjunto de clases e interfaces, que proporciona estructuras de datos y algoritmos para el almacenamiento y manipulación eficiente de grupos de objetos, diseñadas para ser flexibles y extensibles.

***Clases e Interfaces contenidas:***

* Clase Dictionary
* Clase HashTable
* Clase Properties
* Clase Vector
* Interfaz Enumeration
* Interfaz Collection
* Clase Collections
* Clase Arrays
* Interfaz List
* Clase ArrayList
* Clase LinkedList
* Interfaz Set
* Interfaz SortedSet
* Clase HashSet
* Clase TreeSet
* Interfaz Map
* Clase HashMap
* Clase TreeMap
* Clase LinkedHashMap
* Interfaz Stream

## Clase Dictionary

Su objetivo principal es proporcionar una interfaz para asociar claves con valores, teniendo en cuenta que es una clase abstracta que tiene como propósito almacenar datos donde las claves son únicas. Sin embargo, la clase Dictionary ha sido reemplazada en versiones posteriores a partir de Java 1.2 es recomendable utilizar Map.

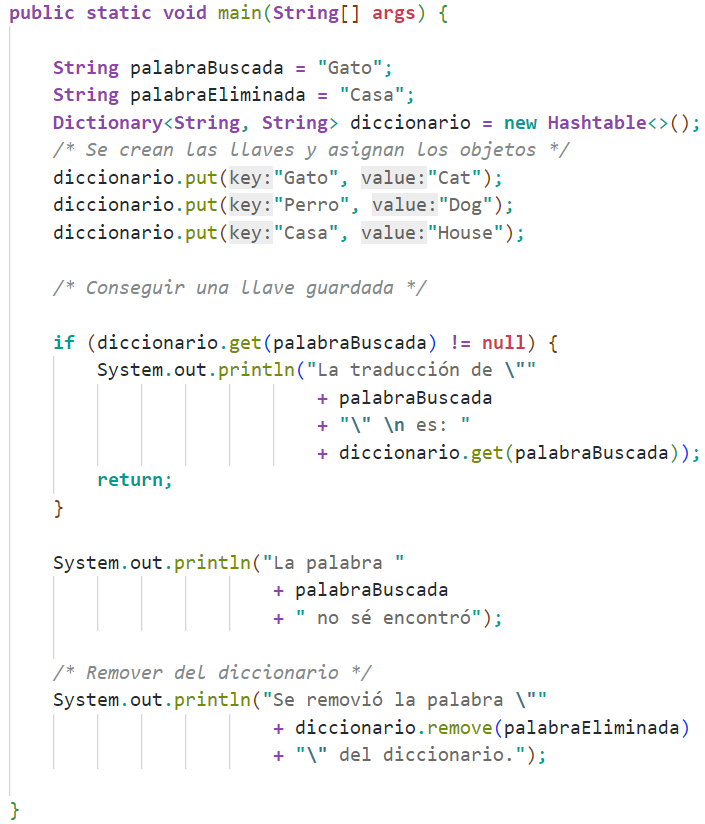
Paquete: java.util

Versión de Java: 1.0

**Tabla 1**  
Métodos de la clase Dictionary

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| abstract Enumeration<V> elements() | Retorna una enumeración de los valores de este diccionario. |
| abstract V get(Object key) | Retorna el valor al que se asigna la clave en este diccionario. |
| abstract boolean isEmpty() | Comprueba si este diccionario no asigna claves al valor. |
| abstract Enumeration<K> keys() | Retorna una enumeración de las claves de este diccionario. |
| abstract V put​(K key, V value) | Asigna la clave especificada al valor especificado en este diccionario. |
| abstract V remove​(Object key) | Elimina la clave (y su valor correspondiente) de este diccionario. |
| abstract int size() | Retorna el número de entradas (claves distintas) en este diccionario. |

*Nota.* Dictionary al ser una clase abstracta, no tiene implementaciones concretas de métodos propios, no obstante, las clases que heredan de esta proporcionan métodos específicos como Hashtable y Properties.



**Figura 1**. Ejemplo en código de la clase Dictionary

## Clase HashTable

Su objetivo principal es la sincronización durante Multi-Threads lo que permite que varias operaciones concurrentes puedan acceder y modificar la tabla hash de forma segura, la funcionalidad es similar a Dictionary, teniendo así la posibilidad de manejar clave-valor únicas, sin embargo, es recomendable utilizar su versión actualizada, la clase HashMap aunque esta no esta sincronizada.

Paquete: java.util

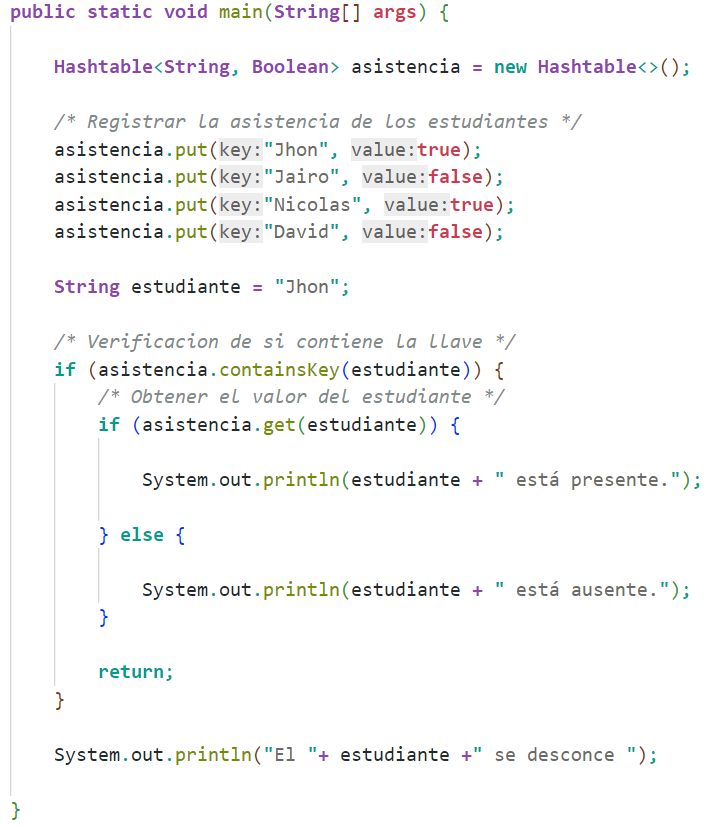
Versión de Java: 1.0

Tabla 2   
*Metodos de la clase HashTable*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| void clear() | Borra esta tabla hash para que no contenga claves. |
| Object clone() | Crea una copia superficial de esta tabla hash. |
| boolean contains​(Object value) | Comprueba si algunas claves se asignan al valor especificado en esta tabla hash. |
| boolean containsKey​(Object key) | Comprueba si el objeto especificado es una clave en esta tabla hash. |
| boolean containsValue​(Object value) | Retorna verdadero si esta tabla hash asigna una o más claves a este valor. |
| Enumeration<V> elements() | Retorna una enumeración de los valores de esta tabla hash. |
| Set<Map.Entry<K,​V>> entrySet() | Retorna una Setvista de las asignaciones contenidas en este map. |
| boolean equals​(Object o) | Compara el objeto especificado con este map para la igualdad, según la definición en la interfaz del map. |
| V get​(Object key) | Retorna el valor al que se asigna la clave especificada, o null si este map no contiene ninguna asignación para la clave. |
| int hashCode() | Retorna el valor del código hash para este map según la definición en la interfaz del map. |
| boolean isEmpty() | Comprueba si esta tabla hash no asigna claves a valores. |
| Enumeration<K> keys() | Retorna una enumeración de las claves en esta tabla hash. |
| Set<K> keySet() | Retorna una Set vista de las claves contenidas en este map. |
| V put​(K key, V value) | Asigna lo especificado key a lo especificado value en esta tabla hash. |
| void putAll​(Map<? extends K,​? extends V> t) | Copia todas las asignaciones del map especificado a esta tabla hash. |
| protected void rehash() | Aumenta la capacidad y reorganiza internamente esta tabla hash, para acomodar y acceder a sus entradas de manera más eficiente. |
| V remove​(Object key) | Elimina la clave (y su valor correspondiente) de esta tabla hash. |
| int size() | Retorna el número de claves en esta tabla hash. |
| String toString() | Retorna una representación del objeto mediante una cadena. |
| Collection<V> values() | Retorna una Collection vista de los valores contenidos en este map. |

*Nota.* Se han omitido unos cuantos métodos, debido a que son bastante complejos en su utilización, aquellos como: *merge, compute, computeIfAbsent, computeIfPresent*.

**Figura 2**. Ejemplo en código de la clase HashTable



## Clase Properties

Properties es una subclase de Hashtable y se utiliza para administrar pares clave-valor en forma de propiedades. Su objetivo principal es proporcionar una forma conveniente de trabajar para cargar y guardar archivos de propiedades en formato de texto, suelen tener una estructura en donde cada línea representa un par clave=valor.

Paquete: java.util

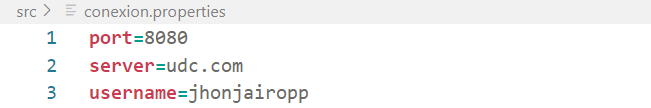
Versión de Java: 1.0

Tabla 3  
 *Metodos de la clase Properties*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| String getProperty​(String key) | Busca la propiedad con la clave especificada en esta lista de propiedades. |
| String getProperty​(String key, String defaultValue) | Busca la propiedad con la clave especificada en esta lista de propiedades. |
| void list​(PrintStream out) | Imprime esta lista de propiedades en el flujo de salida especificado. |
| void list​(PrintWriter out) | Imprime esta lista de propiedades en el flujo de salida especificado. |
| void load​(InputStream inStream) | Lee una lista de propiedades (pares de clave y elemento) del flujo de bytes de entrada. |
| void load​(Reader reader) | Lee una lista de propiedades (pares de clave y elemento) en un formato simple orientado a líneas. |
| void loadFromXML​(InputStream in) | Carga todas las propiedades representadas por el documento XML en el flujo de entrada especificado en esta tabla de propiedades. |
| Enumeration<?> propertyNames() | Retorna una enumeración de todas las claves en esta lista de propiedades |
| Object setProperty​(String key, String value) | Llama al Hashtable método put(). |
| void store​(OutputStream out, String comments) | Escribe esta lista de propiedades en un archivo con formato adecuado para cargar por load(InputStream) |
| void store​(Writer writer, String comments) | Escribe esta lista de propiedades en un archivo con formato adecuado para cargar por load(Reader) |
| void storeToXML​(OutputStream os, String comment) | Emite un documento XML que representa todas las propiedades contenidas en esta tabla. |
| storeToXML​(OutputStream os, String comment, Charset charset) | Emite un documento XML que representa todas las propiedades contenidas en esta tabla, utilizando la codificación especificada. |
| Set<String> stringPropertyNames() | Retorna un conjunto de claves no modificables de esta lista de propiedades donde la clave y su valor correspondiente son cadenas |

*Nota.* Existen más métodos heredados y también se han omitido un método sobrecargado y otro obsoleto son: *storeXML, save.*

**Figura 3**. Ejemplo en código de la clase Properties



## Clase Vector

Vector implementa una lista dinámica que puede crecer o disminuir según sea necesario, es similar a un arreglo, pero tiene la capacidad de ajustar su tamaño automáticamente, siendo también una clase sincronizada, lo que significa que es segura para utilizar durante operaciones concurrentes, no obstante, en versiones más recientes de Java, se recomienda utilizar la clase ArrayList en lugar de Vector debido a mejoras de rendimiento.

Paquete: java.util

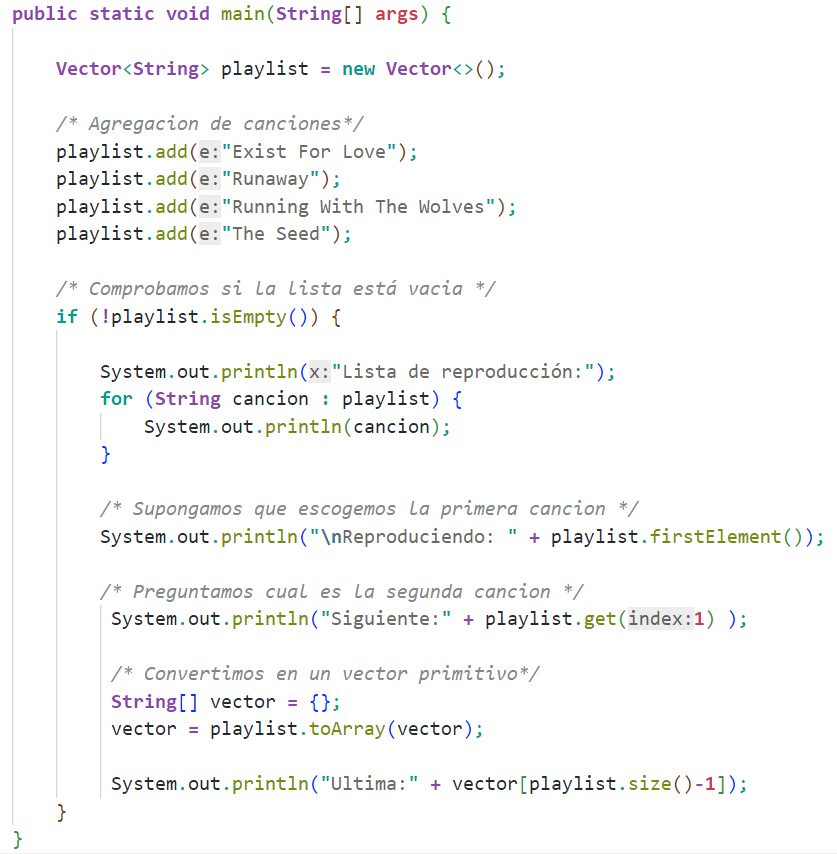
Versión de Java: 1.0

Tabla 4   
*Metodos de la clase Vector*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| void add​(int index, E element) | Inserta el elemento especificado en la posición especificada en este Vector. |
| boolean add​(E e) | Agrega el elemento especificado al final de este Vector. |
| boolean addAll​(int index, Collection<? extends E> c) | Inserta todos los elementos de la Colección especificada en este Vector en la posición especificada. |
| boolean addAll​(Collection<? extends E> c) | Agrega todos los elementos de la colección especificada al final de este vector. |
| void addElement​(E obj) | Agrega el componente especificado al final de este vector, aumentando su tamaño en uno. |
| int capacity() | Devuelve la capacidad actual de este vector. |
| void clear() | Elimina todos los elementos de este Vector. |
| Object clone() | Devuelve un clon de este vector. |
| boolean contains​(Object o) | Devuelve truesi este vector contiene el elemento especificado. |
| boolean containsAll​(Collection<?> c) | Devuelve verdadero si este Vector contiene todos los elementos de la Colección especificada. |
| void copyInto​(Object[] anArray) | Copia los componentes de este vector en la matriz especificada. |
| E elementAt​(int index) | Devuelve el componente en el índice especificado. |
| Enumeration<E> elements() | Devuelve una enumeración de los componentes de este vector. |
| void ensureCapacity​(int minCapacity) | Aumenta la capacidad de este vector, si es necesario. |
| boolean equals​(Object o) | Compara el Objeto especificado con este Vector para la igualdad. |
| E firstElement() | Devuelve el primer componente (el elemento en el índice 0) de este vector. |
| void forEach​(Consumer<? super E> action) | Realiza la acción dada para cada elemento de Iterable hasta que se hayan procesado todos los elementos. |
| E get​(int index) | Devuelve el elemento en la posición especificada en este Vector. |
| int hashCode() | Devuelve el valor del código hash para este Vector. |
| int indexOf​(Object o) | Devuelve el índice de la primera aparición del elemento especificado en este vector, o -1 si este vector no contiene el elemento. |
| int indexOf​(Object o, int index) | Devuelve el índice de la primera aparición del elemento especificado en este vector, buscando hacia delante desde index, o devuelve -1 si no se encuentra el elemento. |
| void insertElementAt​(E obj, int index) | Inserta el objeto especificado como un componente en este vector en el especificado index. |
| boolean isEmpty() | Comprueba si este vector no tiene componentes. |
| Iterator<E> iterator() | Devuelve un iterador sobre los elementos de esta lista en la secuencia adecuada. |
| E lastElement() | Devuelve el último componente del vector. |
| int lastIndexOf​(Object o) | Devuelve el índice de la última aparición del elemento especificado en este vector, o -1 si este vector no contiene el elemento. |
| int lastIndexOf​(Object o, int index) | Devuelve el índice de la última aparición del elemento especificado en este vector, buscando hacia atrás desde index, o devuelve -1 si no se encuentra el elemento. |
| E remove​(int index) | Elimina el elemento en la posición especificada en este Vector. |
| ListIterator<E> listIterator() | Devuelve un iterador de lista sobre los elementos de esta lista (en la secuencia adecuada). |
| ListIterator<E> listIterator​(int index) | Devuelve un iterador de lista sobre los elementos de esta lista (en la secuencia adecuada), comenzando en la posición especificada en la lista. |
| boolean remove​(Object o) | Elimina la primera aparición del elemento especificado en este Vector. |
| boolean removeAll​(Collection<?> c) | Elimina de este Vector todos sus elementos que están contenidos en la Colección especificada. |
| void removeAllElements() | Elimina todos los componentes de este vector y establece su tamaño en cero. |
| boolean removeElement​(Object obj) | Elimina la primera ocurrencia (índice más bajo) del argumento de este vector. |
| void removeElementAt​(int index) | Elimina el componente en el índice especificado. |
| boolean removeIf​(Predicate<? super E> filter) | Elimina todos los elementos de esta colección que satisfacen el predicado dado. |
| protected void removeRange​(int fromIndex, int toIndex) | Elimina de esta lista todos los elementos cuyo índice está entre fromIndex, inclusive, y toIndex, exclusivo. |
| void replaceAll​(UnaryOperator<E> operator) | Reemplaza cada elemento de esta lista con el resultado de aplicar el operador a ese elemento. |
| boolean retainAll​(Collection<?> c) | Retiene solo los elementos de este Vector que están contenidos en la Colección especificada. |
| E set​(int index, E element) | Reemplaza el elemento en la posición especificada en este Vector con el elemento especificado. |
| void setElementAt​(E obj, int index) | Establece el componente en el especificado indexde este vector para que sea el objeto especificado. |
| void setSize​(int newSize) | Establece el tamaño de este vector. |
| int size() | Devuelve el número de componentes de este vector. |
| List<E> subList​(int fromIndex, int toIndex) | Devuelve una vista de la parte de esta lista entre fromIndex, inclusive, y toIndex, exclusivo. |
| <T> T[] toArray​(T[] a) | Devuelve una matriz que contiene todos los elementos de este Vector en el orden correcto; el tipo de tiempo de ejecución de la matriz devuelta es el de la matriz especificada. |
| String toString() | Devuelve una representación de cadena de este vector, que contiene la representación de cadena de cada elemento. |
| void trimToSize() | Recorta la capacidad de este vector para que tenga el tamaño actual del vector. |

*Nota.* Se omitieron algunos métodos heredados y uno propio el cual es*: spliterator().*

**Figura 4**. Ejemplo en código de la clase Vector



## Interfaz Enumeration

El objetivo de la interfaz Enumeration es proporcionar un mecanismo simple y eficiente para iterar secuencialmente elementos en una colección como un Vector o un Hashtable, sin permitir modificaciones en la colección subyacente, cabe resaltar que la interfaz Enumeration es una versión anterior a la interfaz Iterator.

Paquete: java.util

Versión de Java: 1.0

Tabla 5  
*Metodos de la interfaz Enumeration*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| default Iterator<E> asIterator() | Retorna un Iterator que atraviesa los elementos restantes cubiertos por esta enumeración. |
| boolean hasMoreElements() | Comprueba si esta enumeración contiene más elementos. |
| E nextElement() | Retorna el siguiente elemento de esta enumeración si este objeto de enumeración tiene al menos un elemento más para proporcionar. |

## Interfaz Collection

**Figura 5.** Ejemplo en código de la interfaz Enumeration



Collection es la raíz de la jerarquía de las colecciones en Java, definiendo las operaciones básicas que se pueden realizar en una colección de objetos, como agregar elementos, eliminar elementos, verificar la presencia de un elemento, entre otros, se convierte en una interfaz genérica que se puede implementar en diversas clases concretas, como ArrayList, LinkedList, HashSet, entre otras.

Paquete: java.util

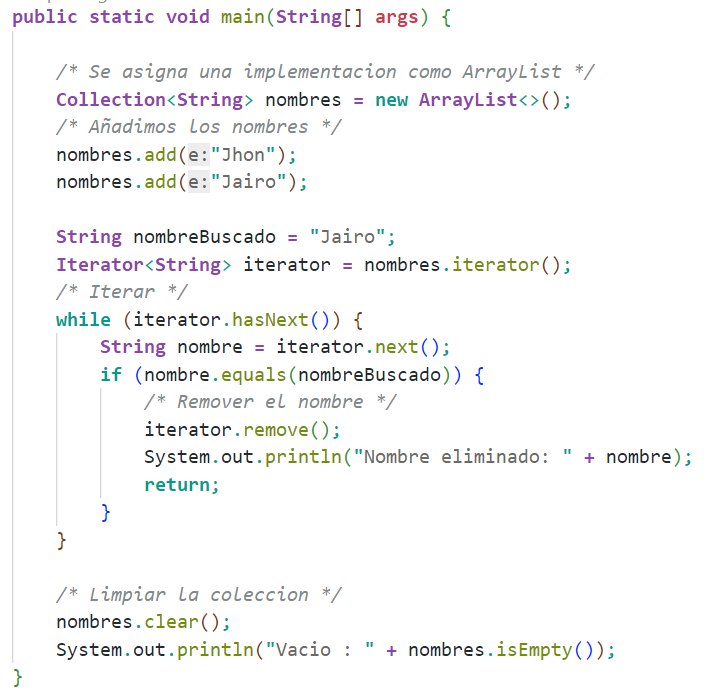
Versión de Java: 1.2

Tabla 6  
*Metodos de la interfaz Collection*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| boolean add​(E e) | Garantiza que esta colección contenga el elemento especificado (operación opcional). |
| boolean addAll​(Collection<? extends E> c) | Agrega todos los elementos de la colección especificada a esta colección (operación opcional). |
| void clear() | Elimina todos los elementos de esta colección (operación opcional). |
| boolean contains​(Object o) | Retorna true si esta colección contiene el elemento especificado. |
| boolean containsAll​(Collection<?> c) | Retorna true si esta colección contiene todos los elementos de la colección especificada. |
| boolean equals​(Object o) | Compara el objeto especificado con esta colección para la igualdad. |
| int hashCode() | Retorna el valor del código hash para esta colección. |
| boolean isEmpty() | Retorna true si esta colección no contiene elementos. |
| Iterator<E> iterator() | Retorna un iterador sobre los elementos de esta colección. |
| default Stream<E> parallelStream() | Retorna un posible paralelo Stream con esta colección como fuente. |
| boolean remove​(Object o) | Elimina una sola instancia del elemento especificado de esta colección |
| boolean removeAll​(Collection<?> c) | Elimina todos los elementos de esta colección que también están contenidos en la colección especificada |
| default boolean removeIf​(Predicate<? super E> filter) | Elimina todos los elementos de esta colección que satisfacen el predicado dado. |
| boolean retainAll​(Collection<?> c) | Retiene solo los elementos de esta colección que están contenidos en la colección especificada |
| int size() | Retorna el número de elementos de esta colección. |
| default Stream<E> stream() | Retorna un secuencial Stream con esta colección como fuente. |
| Object[] toArray() | Retorna una matriz que contiene todos los elementos de esta colección. |
| default <T> T[] toArray​(IntFunction<T[]> generator) | Retorna una matriz que contiene todos los elementos de esta colección, utilizando la generator función proporcionada para asignar la matriz devuelta. |
| <T> T[] toArray​(T[] a) | Retorna una matriz que contiene todos los elementos de esta colección; el tipo de tiempo de ejecución de la matriz devuelta es el de la matriz especificada. |

*Nota.* Se omitieron algunos métodos heredados y uno propio el cual es*: spliterator().*

**Figura 6.** Ejemplo en código de la interfaz Collection



## Clase Collections

El objetivo principal de la clase Collections es proporcionar una biblioteca de métodos estáticos que amplían las capacidades de las colecciones estándar en Java y permiten realizar operaciones comunes de manera sencilla y eficiente, todos sus métodos son estáticos por lo tanto no se puede instanciar.

Paquete: java.util

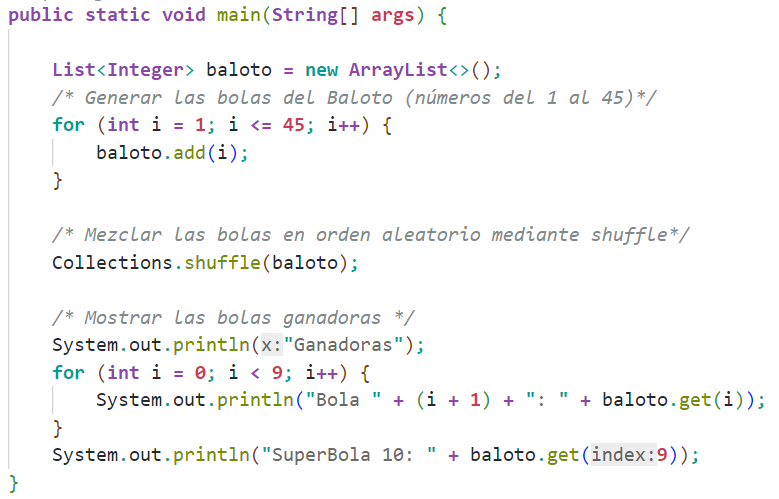
Versión de Java: 1.2

Tabla 7  
 *Metodos de la clase Collections*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| static <T> void sort​(List<T> list, Comparator<? super T> c) | Ordena la lista especificada en orden ascendente, según el orden natural de sus elementos. |
| static void reverse​(List<?> list) | Invierte el orden de los elementos en la lista especificada. |
| static void shuffle​(List<?> list) | Permuta aleatoriamente la lista especificada utilizando una fuente predeterminada de aleatoriedad. |
| static <T> int binarySearch​(List<? extends Comparable<? super T>> list, T key) | Busca en la lista especificada el objeto especificado utilizando el algoritmo de búsqueda binaria. |
| static <T> boolean addAll​(Collection<? super T> c, T... elements) | Agrega todos los elementos especificados a la colección especificada. |

*Nota.* En la Tabla 7, se muestran algunos de los métodos mas importantes, teniendo en cuenta que son más de 60 métodos (copy, max, min, fill... entre otros) que incluyen ordenar, buscar, mezclar, invertir, copiar y más.

**Figura 7**. Ejemplo en código de la clase Collections.



## Clase Arrays

El objetivo de la interfaz Enumeration es proporcionar un mecanismo simple y eficiente para iterar secuencialmente elementos en una colección como un Vector o un Hashtable, sin permitir modificaciones en la colección subyacente, cabe resaltar que la interfaz Enumeration es una versión anterior a la interfaz Iterator.

Paquete: java.util

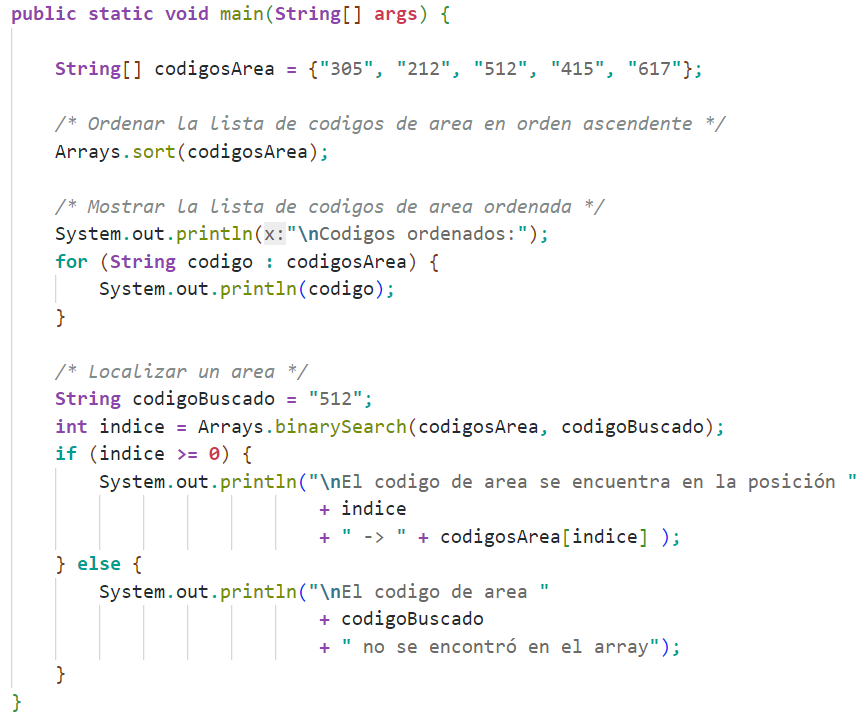
Versión de Java: 1.2

Tabla 8   
*Metodos de la clase Arrays*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| static <T> List<T> asList​(T... a) | Devuelve una lista de tamaño fijo respaldada por la matriz especificada. |
| static int binarySearch​(Object[] a, Object key) | Busca en la matriz especificada el objeto especificado utilizando el algoritmo de búsqueda binaria. |
| static <T> int compare​(T[] a, T[] b, Comparator<? super T> cmp) | Compara dos Object matrices lexicográficamente usando un comparador especificado. |
| static int compareUnsigned​(short[] a, short[] b) | Compara dos short matrices lexicográficamente, tratando numéricamente los elementos como sin signo. |
| static <T> T[] copyOf​(T[] original, int newLength) | Copia la matriz especificada, truncando o rellenando con nulos (si es necesario) para que la copia tenga la longitud especificada. |
| static <T> T[] copyOfRange​(T[] original, int from, int to) | Copia el rango especificado de la matriz especificada en una nueva matriz. |
| static boolean deepEquals​(Object[] a1, Object[] a2) | Devuelve true si las dos matrices especificadas son profundamente iguales entre sí. |
| static boolean equals​(Object[] a, Object[] a2) | Devuelve true si las dos matrices especificadas de Objetos son iguales entre sí. |
| static void fill​(Object[] a, Object val | Asigna la referencia de objeto especificada a cada elemento de la matriz de objetos especificada. |
| static int hashCode​(Object[] a) | Devuelve un código hash basado en el contenido de la matriz especificada. |
| static <T> int mismatch​(T[] a, T[] b, Comparator<? super T> cmp) | Busca y devuelve el índice de la primera discrepancia entre dos Object matrices; de lo contrario, devuelve -1 si no se encuentra ninguna discrepancia. |
| static <T> void parallelSetAll​(T[] array, IntFunction<? extends T> generator) | Establezca todos los elementos de la matriz especificada, en paralelo, utilizando la función de generador proporcionada para calcular cada elemento. |
| static <T> void parallelSort​(T[] a, Comparator<? super T> cmp) | Ordena la matriz de objetos especificada según el orden inducido por el comparador especificado. |
| static <T> void sort​(T[] a, Comparator<? super T> c) | Ordena la matriz de objetos especificada según el orden inducido por el comparador especificado. |
| static <T> Stream<T> stream​(T[] array) | Devuelve un secuencial Stream con la matriz especificada como fuente. |
| static String toString​(Object[] a) | Devuelve una representación de cadena del contenido de la matriz especificada. |

*Nota.* Existen todavía más métodos para esta clase, no obstante, la mayoría son sobrecargas de los presentados, teniendo así el mismo nombre, pero diferentes parámetros o retorno.

**Figura 8.** Ejemplo en código de la clase Arrays



## Interfaz List

El objetivo es proporcionar una colección ordenada y modificable de elementos en la que se puede acceder a los elementos mediante índices y realizar operaciones de inserción, eliminación y búsqueda de manera eficiente. Las implementaciones comunes de la interfaz incluyen ArrayList, LinkedList y Vector.

Paquete: java.util

Versión de Java: 1.2

Tabla 9   
*Métodos de la interfaz List*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| void add​(int index, E element) | Inserta el elemento especificado en la posición especificada en esta lista |
| boolean add​(E e) | Agrega el elemento especificado al final de esta lista |
| boolean addAll​(int index, Collection<? extends E> c) | Inserta todos los elementos de la colección especificada en esta lista en la posición especificada |
| boolean addAll​(Collection<? extends E> c) | Agrega todos los elementos de la colección especificada al final de esta lista, en el orden en que los devuelve el iterador de la colección especificada |
| void clear() | Elimina todos los elementos de esta lista |
| boolean contains​(Object o) | Devuelve true si esta lista contiene el elemento especificado. |
| boolean containsAll​(Collection<?> c) | Devuelve true si esta lista contiene todos los elementos de la colección especificada. |
| static <E> List<E> copyOf​(Collection<? extends E> coll) | Devuelve una Lista no modificable |
| boolean equals​(Object o) | Compara el objeto especificado con esta lista para la igualdad. |
| E get​(int index) | Devuelve el elemento en la posición especificada en esta lista. |
| int hashCode() | Devuelve el valor del código hash para esta lista. |
| int indexOf​(Object o) | Devuelve el índice de la primera aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento. |
| boolean isEmpty() | Devuelve true si esta lista no contiene elementos. |
| Iterator<E> iterator() | Devuelve un iterador sobre los elementos de esta lista en la secuencia adecuada. |
| int lastIndexOf​(Object o) | Devuelve el índice de la última aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento |
| ListIterator<E> listIterator() | Devuelve un iterador de lista sobre los elementos de esta lista (en la secuencia adecuada). |
| default void sort​(Comparator<? super E> c) | Ordena esta lista según el orden inducido por el especificado Comparator. |
| static <E> List<E> of() | Devuelve una lista no modificable que contiene cero elementos. |
| E remove​(int index) | Elimina el elemento en la posición especificada en esta lista |
| boolean remove​(Object o) | Elimina la primera aparición del elemento especificado de esta lista, si está presente |
| E set​(int index, E element) | Reemplaza el elemento en la posición especificada en esta lista con el elemento especificado (operación opcional). |
| int size() | Devuelve el número de elementos de esta lista. |
| Object[] toArray() | Devuelve una matriz que contiene todos los elementos de esta lista |
| List<E> subList​(int fromIndex, int toIndex) | Devuelve una vista de la parte de esta lista entre el especificado fromIndex, inclusive, y toIndex, exclusivo. |

*Nota.* Aún existen algunos métodos no mencionados, mayormente son sobrecargas de estos mismos o omitidos: *of(), toArray(), spliterator(), retainAll().*

**Figura 9.** Ejemplo en código de la interfaz List



## Clase ArrayList

La clase implementa la interfaz List y proporciona una implementación de una lista dinámica respaldada por un arreglo, donde resulta ser eficiente siendo que es redimensionable en la que los elementos se almacenan de manera contigua en memoria. Esto permite un acceso rápido a los elementos mediante índices y un rendimiento eficiente en operaciones como agregar, eliminar y buscar elementos.

Paquete: java.util

Versión de Java: 1.2

Tabla 10   
*Métodos de la clase ArrayList*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| void add​(int index, E element) | Inserta el elemento especificado en la posición especificada en esta lista. |
| boolean add​(E e) | Agrega el elemento especificado al final de esta lista. |
| boolean addAll​(int index, Collection<? extends E> c) | Inserta todos los elementos de la colección especificada en esta lista, comenzando en la posición especificada. |
| void clear() | Elimina todos los elementos de esta lista. |
| Object clone() | Devuelve una copia superficial de esta ArrayList instancia. |
| boolean contains​(Object o) | Devuelve true si esta lista contiene el elemento especificado. |
| void ensureCapacity​(int minCapacity) | Aumenta la capacidad de esta ArrayList instancia, si es necesario, para garantizar que pueda contener al menos la cantidad de elementos especificados por el argumento de capacidad mínima. |
| void forEach​(Consumer<? super E> action) | Realiza la acción dada para cada elemento de Iterable hasta que se hayan procesado todos los elementos o la acción genere una excepción. |
| E get​(int index) | Devuelve el elemento en la posición especificada en esta lista. |
| int indexOf​(Object o) | Devuelve el índice de la primera aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento. |
| boolean isEmpty() | Devuelve true si esta lista no contiene elementos. |
| Iterator<E> iterator() | Devuelve un iterador sobre los elementos de esta lista en la secuencia adecuada. |
| int lastIndexOf​(Object o) | Devuelve el índice de la última aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento. |
| E remove​(int index) | Elimina el elemento en la posición especificada en esta lista. |
| boolean remove​(Object o) | Elimina la primera aparición del elemento especificado de esta lista, si está presente. |
| E set​(int index, E element) | Reemplaza el elemento en la posición especificada en esta lista con el elemento especificado. |
| List<E> subList​(int fromIndex, int toIndex) | Devuelve una vista de la parte de esta lista entre el especificado fromIndex, inclusive, y toIndex, exclusivo. |
| void trimToSize() | Recorta la capacidad de esta ArrayList instancia para que tenga el tamaño actual de la lista. |
| Object[] toArray() | Devuelve una matriz que contiene todos los elementos de esta lista en la secuencia adecuada (desde el primero hasta el último elemento). |

*Nota.* Debido a que implementa la interfaz List, mayormente todos los métodos de esta simplemente están optimizados y sobrescritos por lo tanto comparten los mismos métodos y algunas implementaciones nuevas.

**Figura 10**. Ejemplo en código de la clase ArrayList



## Clase LinkedList

La clase LinkedList implementa la interfaz List y proporciona una implementación de una lista doblemente enlazada. A diferencia de ArrayList, LinkedList no utiliza un arreglo interno para almacenar los elementos, sino que cada elemento está vinculado a su predecesor y sucesor. Esto permite un acceso rápido a elementos cercanos, pero puede ser menos eficiente para acceder a elementos en posiciones aleatorias. es especialmente útil cuando se requieren operaciones frecuentes de inserción o eliminación en el medio de la lista.

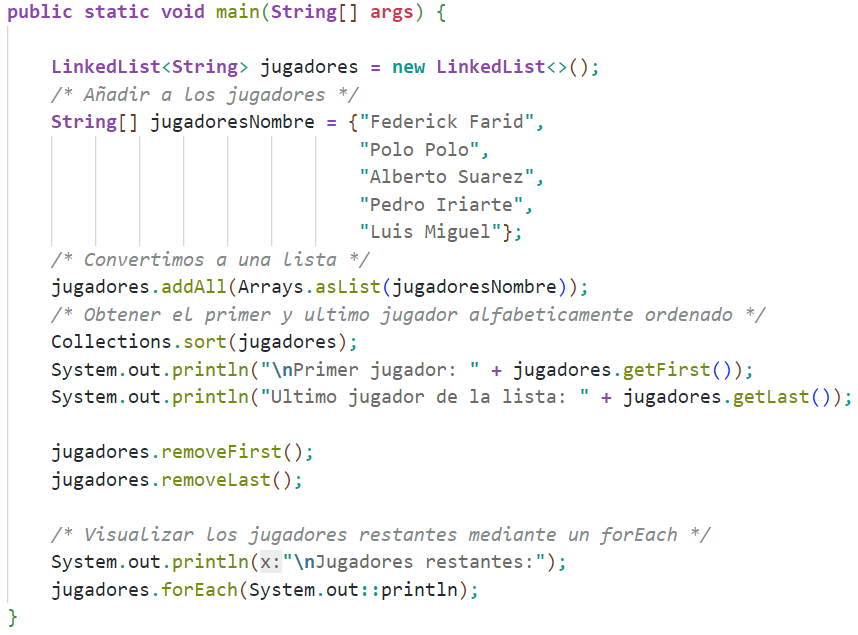
Paquete: java.util

Versión de Java: 1.2

Tabla 11   
*Métodos de la clase LinkedList*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| void add​(int index, E element) | Inserta el elemento especificado en la posición especificada en esta lista. |
| boolean add​(E e) | Agrega el elemento especificado al final de esta lista. |
| boolean addAll​(int index, Collection<? extends E> c) | Inserta todos los elementos de la colección especificada en esta lista, comenzando en la posición especificada. |
| void addFirst​(E e) | Inserta el elemento especificado al principio de esta lista. |
| void addLast​(E e) | Agrega el elemento especificado al final de esta lista. |
| void clear() | Elimina todos los elementos de esta lista. |
| Object clone() | Devuelve una copia superficial de este LinkedList. |
| boolean contains​(Object o) | Devuelve true si esta lista contiene el elemento especificado. |
| Iterator<E> descendingIterator() | Devuelve un iterador sobre los elementos de este de que en orden secuencial inverso. |
| E element() | Recupera, pero no elimina, el encabezado (primer elemento) de esta lista. |
| E get​(int index) | Devuelve el elemento en la posición especificada en esta lista. |
| 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 aparición 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 aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento. |
| boolean offer​(E e) | Agrega el elemento especificado como la cola (último elemento) de esta lista. |
| boolean offerFirst​(E e) | Inserta el elemento especificado al principio de esta lista. |
| boolean offerLast​(E e) | Inserta el elemento especificado al final de esta lista. |
| E peek() | Recupera, pero no elimina, el encabezado (primer elemento) de esta lista. |
| E peekFirst() | Recupera, pero no elimina, el primer elemento de esta lista, o devuelve null si esta lista está vacía. |
| E peekLast() | Recupera, pero no elimina, el último elemento de esta lista, o devuelve null si esta lista está vacía. |
| E poll() | Recupera y elimina el encabezado (primer elemento) de esta lista. |
| E pollFirst() | Recupera y elimina el primer elemento de esta lista, o regresa null si esta lista está vacía. |
| E pollLast() | Recupera y elimina el último elemento de esta lista, o regresa null si esta lista está vacía. |
| E pop() | Extrae un elemento de la pila representada por esta lista. |
| void push​(E e) | Empuja un elemento a la pila representada por esta lista. |
| E remove() | Recupera y elimina el encabezado (primer elemento) de esta lista. |
| boolean removeFirstOccurrence​(Object o) | Elimina la primera aparición del elemento especificado en esta lista (al recorrer la lista de principio a fin). |
| default Iterator<E> asIterator() | Retorna un Iterator que atraviesa los elementos restantes cubiertos por esta enumeración. |
| boolean hasMoreElements() | Comprueba si esta enumeración contiene más elementos. |
| E nextElement() | Retorna el siguiente elemento de esta enumeración si este objeto de enumeración tiene al menos un elemento más para proporcionar. |

*Nota.* Aun no están contenidos en su totalidad, todos los métodos de LinkedList, este contiene al igual que ArrayList, la implementación de List por ello comparten dichos métodos y los sobrescriben.



**Figura 11.** Ejemplo en código de la clase LinkedList

## Interfaz Set

El objetivo principal de la interfaz Set es proporcionar una estructura de datos que no permite la duplicación de elementos y permite realizar operaciones eficientes de búsqueda y eliminación, siendo que hereda de la interfaz Collection y agrega operaciones específicas para trabajar con conjuntos.

Paquete: java.util

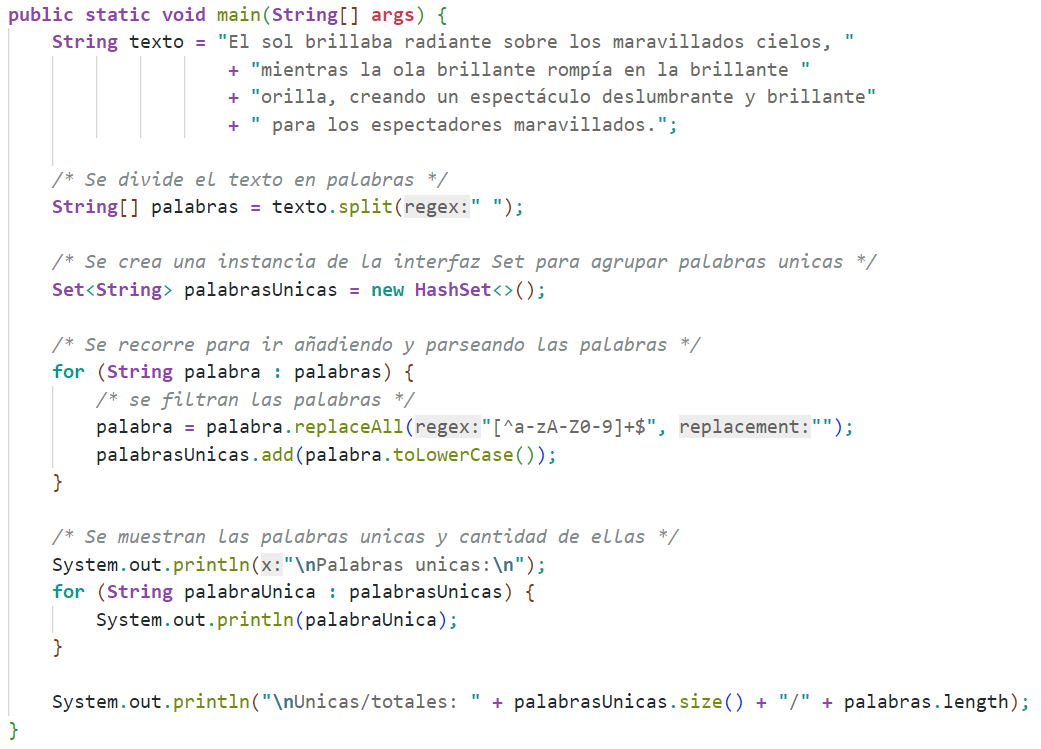
Versión de Java: 1.2

Tabla 12   
*Métodos de la interfaz Set*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| boolean add​(E e) | Agrega el elemento especificado a este conjunto si aún no está presente. |
| boolean addAll​(Collection<? extends E> c) | Agrega todos los elementos de la colección especificada a este conjunto si aún no están presentes |
| void clear() | Elimina todos los elementos de este conjunto |
| boolean contains​(Object o) | Devuelve truesi este conjunto contiene el elemento especificado. |
| boolean containsAll​(Collection<?> c) | Devuelve true si este conjunto contiene todos los elementos de la colección especificada. |
| boolean equals​(Object o) | Compara el objeto especificado con este conjunto para la igualdad. |
| int hashCode() | Devuelve el valor del código hash para este conjunto. |
| boolean isEmpty() | Devuelve truesi este conjunto no contiene elementos. |
| Iterator<E> iterator() | Devuelve un iterador sobre los elementos de este conjunto. |
| static <E> Set<E> of​(E... elements) | Devuelve un conjunto no modificable que contiene un número arbitrario de elementos. |
| boolean remove​(Object o) | Elimina el elemento especificado de este conjunto si está presente (operación opcional). |
| int size() | Devuelve el número de elementos de este conjunto (su cardinalidad). |
| Object[] toArray() | Devuelve una matriz que contiene todos los elementos de este conjunto. |

*Nota.* Debido a que hereda métodos de la interfaz Collection, se omite en esta Tabla 12, algunos de ellos y se incluyen los propios de la interfaz y sus sobreescrituras.

**Figura 12.** Ejemplo en código de la interfaz Set



## Interfaz SortedSet

Es una subinterfaz de la interfaz Set que mantiene los elementos ordenados según su orden natural o mediante un comparador personalizado. Garantiza que los elementos se almacenen en un orden específico y permite realizar operaciones de conjunto eficientes, como la búsqueda de elementos y la obtención de subconjuntos basados en rangos.

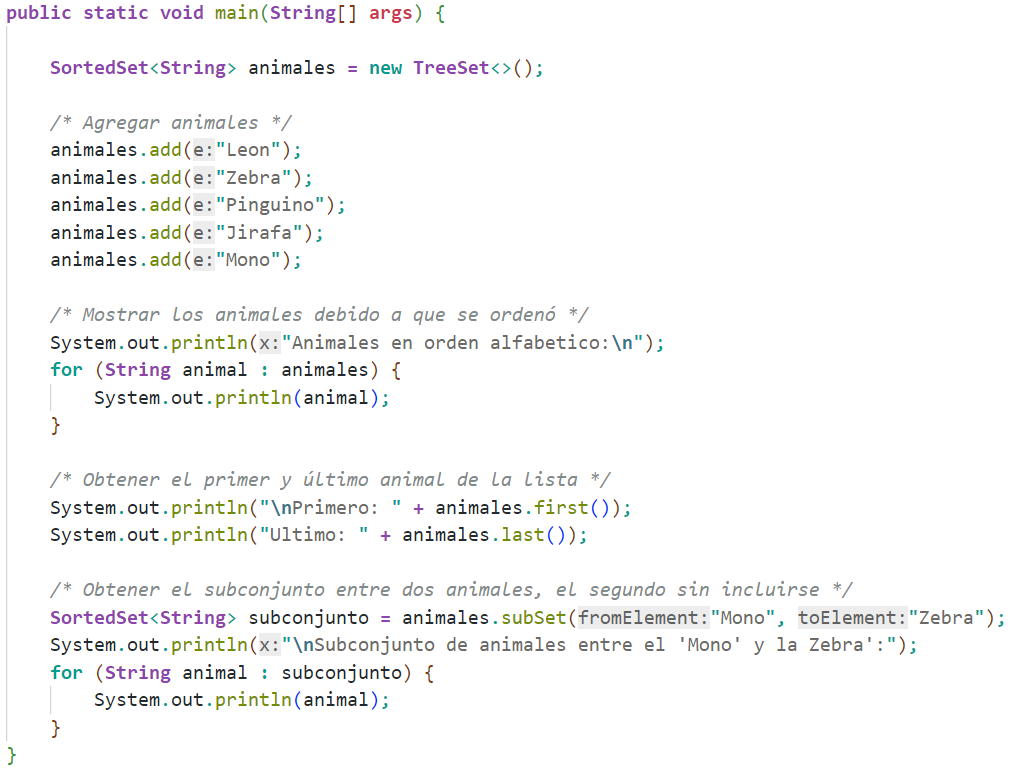
Paquete: java.util

Versión de Java: 1.2

Tabla 13  
*Métodos de la interfaz SortedSet*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| Comparator<? super E> comparator() | Devuelve el comparador utilizado para ordenar los elementos de este conjunto, o null si este conjunto utiliza la ordenación natural de sus elementos. |
| E first() | Devuelve el primer elemento (más bajo) actualmente en este conjunto. |
| SortedSet<E> headSet​(E toElement) | Devuelve una vista de la parte de este conjunto cuyos elementos son estrictamente menores que toElement. |
| E last() | Devuelve el último elemento (más alto) actualmente en este conjunto. |
| default Spliterator<E> spliterator() | Crea un Spliterator sobre los elementos de este conjunto ordenado. |
| SortedSet<E> subSet​(E fromElement, E toElement) | Devuelve una vista de la parte de este conjunto cuyos elementos van desde fromElement, inclusive, hasta toElement, exclusivo. |
| SortedSet<E> tailSet​(E fromElement) | Devuelve una vista de la parte de este conjunto cuyos elementos son mayores o iguales que fromElement. |

**Figura 13.** Ejemplo en código de la interfaz SortedSet



## Clase HashSet

La clase HashSet se basa en la estructura de datos de tabla hash, lo que permite un acceso rápido a los elementos mediante el uso de claves. Utiliza el concepto de "hashing" para asignar los elementos a "buckets" en la tabla hash, lo que facilita la búsqueda y verificación de duplicados de manera eficiente, es importante destacar que el orden de iteración de los elementos en un HashSet puede variar con el tiempo, ya que está basado en la función hash y no se mantiene un orden específico.

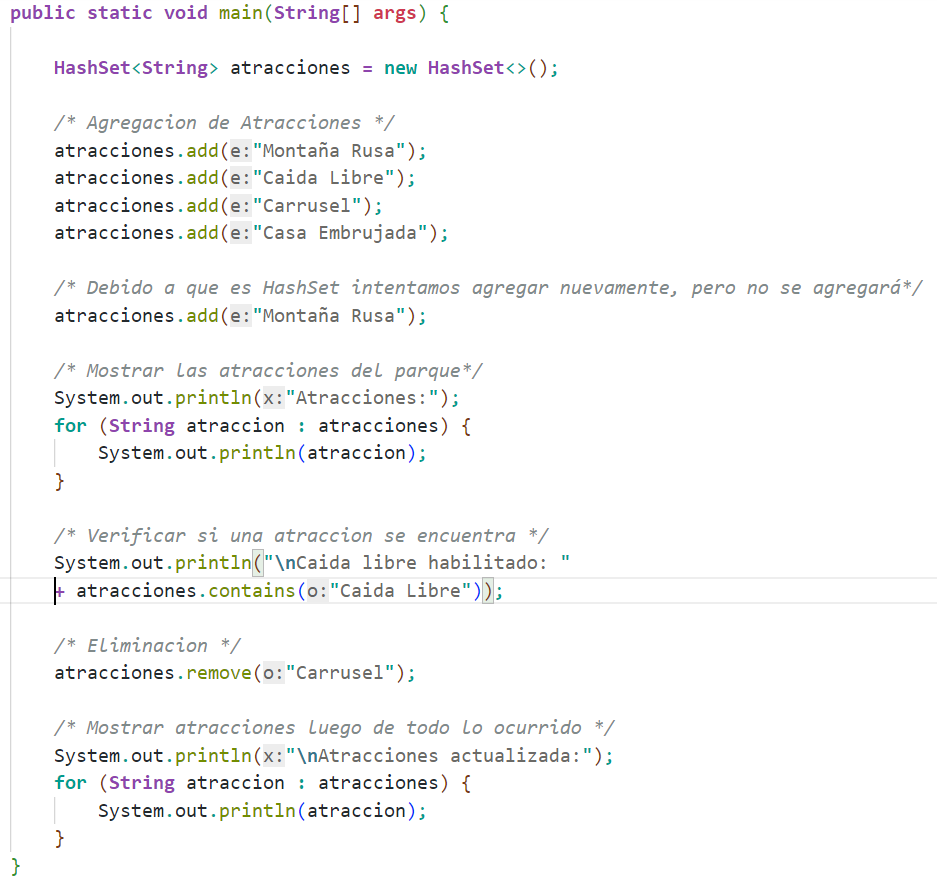
Paquete: java.util

Versión de Java: 1.2

Tabla 14   
*Métodos de la clase HashSet*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| boolean add​(E e) | Agrega el elemento especificado a este conjunto si aún no está presente. |
| void clear() | Elimina todos los elementos de este conjunto. |
| Object clone() | Devuelve una copia superficial de esta HashSet instancia: los elementos en sí no se clonan. |
| boolean contains​(Object o) | Devuelve true si este conjunto contiene el elemento especificado. |
| boolean isEmpty() | Devuelve true si este conjunto no contiene elementos. |
| Iterator<E> iterator() | Devuelve un iterador sobre los elementos de este conjunto. |
| boolean remove​(Object o) | Elimina el elemento especificado de este conjunto si está presente. |
| int size() | Devuelve el número de elementos de este conjunto (su cardinalidad). |
| Spliterator<E> spliterator() | Crea un enlace tardío y rápido Spliterator sobre los elementos de este conjunto. |

**Figura 14.** Ejemplo en código de la clase HashSet



## 

## Clase TreeSet

La clase TreeSet implementa la interfaz SortedSet y utiliza una estructura de árbol (*red-black tree*) para almacenar los elementos. Mantiene los elementos ordenados de acuerdo con su orden natural o utilizando un comparador personalizado. TreeSet ofrece un rendimiento eficiente para las operaciones de inserción, eliminación y búsqueda, y permite realizar operaciones de conjunto basadas en el orden, como obtener el primer elemento, el último elemento y subconjuntos basados en rangos.

Paquete: java.util

Versión de Java: 1.2

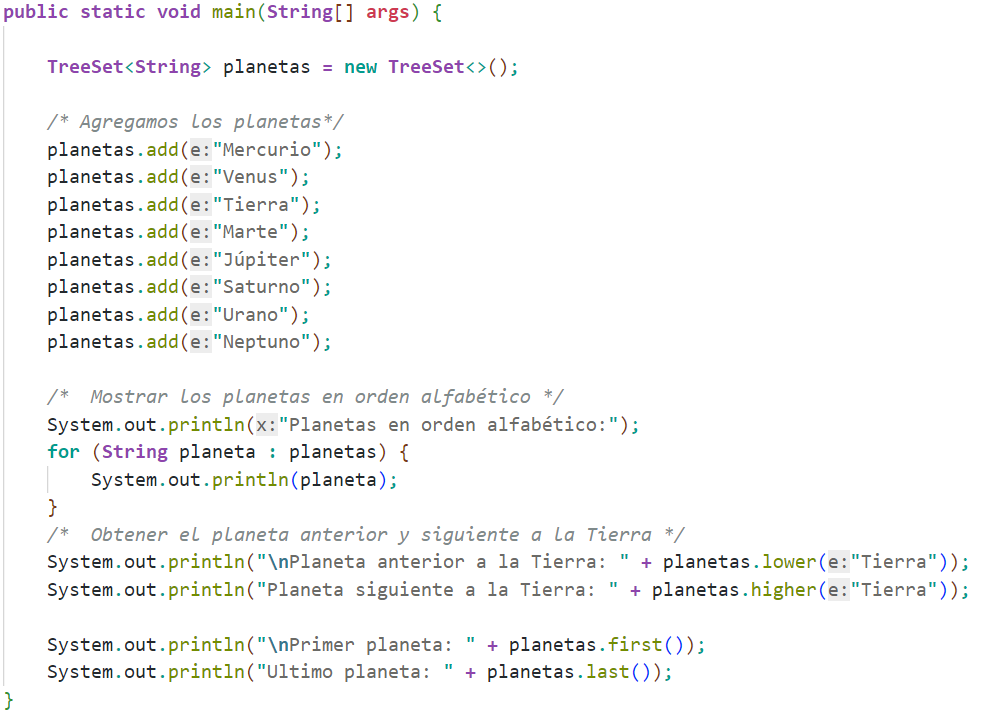
Tabla 15  
*Métodos de la clase TreeSet*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| boolean add​(E e) | Agrega el elemento especificado a este conjunto si aún no está presente. |
| boolean addAll​(Collection<? extends E> c) | Agrega todos los elementos de la colección especificada a este conjunto. |
| E ceiling​(E e) | Devuelve el menor elemento de este conjunto mayor o igual que el elemento dado, o null si no existe tal elemento. |
| void clear() | Elimina todos los elementos de este conjunto. |
| Object clone() | Devuelve una copia superficial de esta TreeSet instancia. |
| boolean contains​(Object o) | Devuelve true si este conjunto contiene el elemento especificado. |
| Iterator<E> descendingIterator() | Devuelve un iterador sobre los elementos de este conjunto en orden descendente. |
| NavigableSet<E> descendingSet() | Devuelve una vista en orden inverso de los elementos contenidos en este conjunto. |
| E first() | Devuelve el primer elemento (más bajo) actualmente en este conjunto. |
| E floor​(E e) | Devuelve el mayor elemento de este conjunto menor o igual que el elemento dado, o null si no existe tal elemento. |
| SortedSet<E> headSet​(E toElement) | Devuelve una vista de la parte de este conjunto cuyos elementos son estrictamente menores que toElement. |
| NavigableSet<E> headSet​(E toElement, boolean inclusive) | Devuelve una vista de la parte de este conjunto cuyos elementos son menores que (o iguales, si inclusive es verdadero) toElement. |
| E higher​(E e) | Devuelve el elemento mínimo de este conjunto estrictamente mayor que el elemento dado, o null si no existe tal elemento. |
| boolean isEmpty() | Devuelve true si este conjunto no contiene elementos. |
| Iterator<E> iterator() | Devuelve un iterador sobre los elementos de este conjunto en orden ascendente. |
| E last() | Devuelve el último elemento (más alto) actualmente en este conjunto. |
| E lower​(E e) | Devuelve el mayor elemento de este conjunto estrictamente menor que el elemento dado, o null si no existe tal elemento. |
| E pollFirst() | Recupera y elimina el primer elemento (más bajo), o regresa null si este conjunto está vacío. |
| E pollLast() | Recupera y elimina el último elemento (más alto), o regresa null si este conjunto está vacío. |
| boolean remove​(Object o) | Elimina el elemento especificado de este conjunto si está presente. |
| SortedSet<E> subSet​(E fromElement, E toElement) | Devuelve una vista de la parte de este conjunto cuyos elementos van desde fromElement, inclusive, hasta toElement, exclusivo. |
| NavigableSet<E> tailSet​(E fromElement, boolean inclusive) | Devuelve una vista de la parte de este conjunto cuyos elementos son mayores que (o iguales que, si inclusive es verdadero) fromElement. |
| int size() | Devuelve el número de elementos de este conjunto (su cardinalidad). |

*Nota.* Es de mencionar que se omitieron algunos métodos sobrecargados, pero con la misma funcionalidad.

## 

**Figura 15.** Ejemplo en código de la clase HashSet



## Interfaz Map

El objetivo de la interfaz Map es proporcionar una estructura de datos que permita asociar valores con claves y permita la recuperación rápida de valores utilizando las claves correspondientes. Es útil para tareas como almacenar configuraciones, realizar búsquedas basadas en claves y realizar operaciones de asignación clave-valor eficientes.

Paquete: java.util

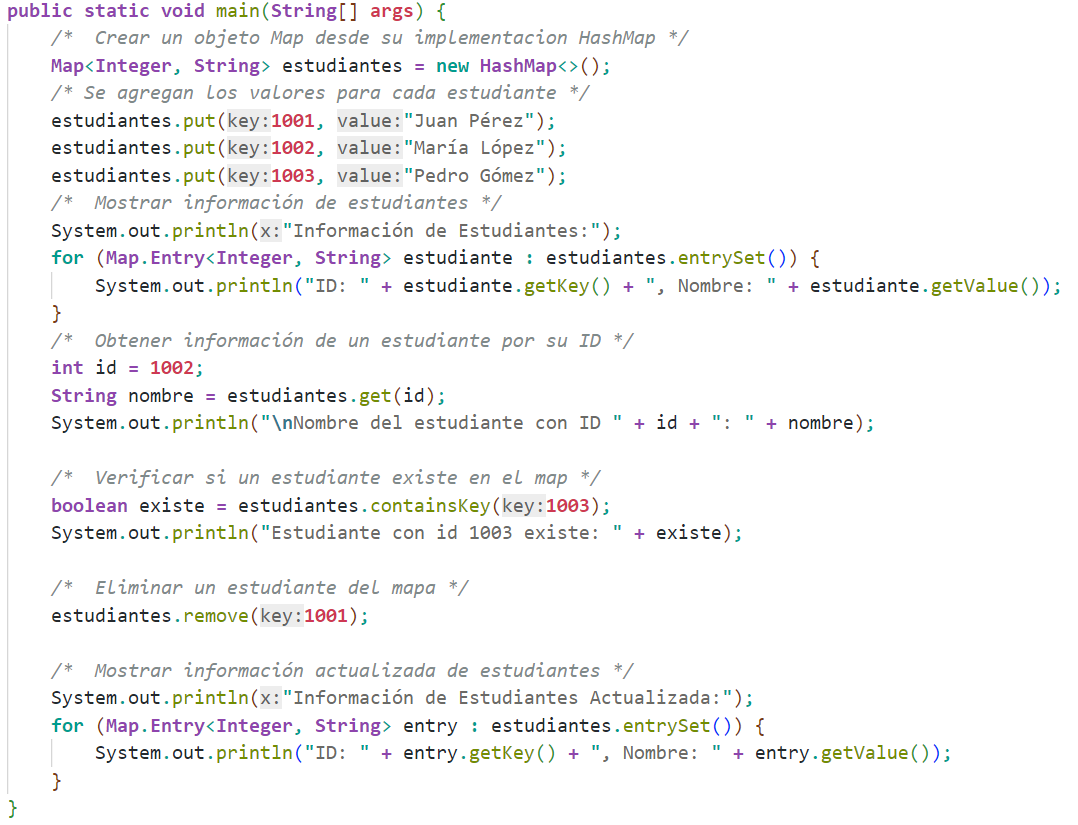
Versión de Java: 1.2

Tabla 16  
 *Métodos de la interfaz Map*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| void clear() | Elimina todas las asignaciones de este mapa |
| default V compute​(K key, BiFunction<? super K,​? super V,​? extends V> remappingFunction) | Intenta calcular una asignación para la clave especificada y su valor asignado actual (o nullsi no hay una asignación actual). |
| boolean containsKey​(Object key) | Devuelve truesi este mapa contiene un mapeo para la clave especificada. |
| boolean containsValue​(Object value) | Devuelve truesi este mapa asigna una o más claves al valor especificado. |
| static <K,​V> Map<K,​V> copyOf​(Map<? extends K,​? extends V> map) | Devuelve un Mapa no modificable que contiene las entradas del Mapa dado. |
| static <K,​V> Map.Entry<K,​V> entry​(K k, V v) | Devuelve un no modificable Map.Entryque contiene la clave y el valor dados. |
| Set<Map.Entry<K,​V>> entrySet() | Devuelve una Setvista de las asignaciones contenidas en este mapa. |
| boolean equals​(Object o) | Compara el objeto especificado con este mapa para la igualdad. |
| default void forEach​(BiConsumer<? super K,​? super V> action) | Realiza la acción dada para cada entrada en este mapa hasta que todas las entradas hayan sido procesadas o la acción arroje una excepción. |
| V get​(Object key) | Devuelve el valor al que se asigna la clave especificada, o nullsi este mapa no contiene ninguna asignación para la clave. |
| default V getOrDefault​(Object key, V defaultValue) | Devuelve el valor al que se asigna la clave especificada, o defaultValuesi este mapa no contiene ninguna asignación para la clave. |
| int hashCode() | Devuelve el valor del código hash para este mapa. |
| boolean isEmpty() | Devuelve truesi este mapa no contiene asignaciones de clave-valor. |
| Set<K> keySet() | Devuelve una Setvista de las claves contenidas en este mapa. |
| default V merge​(K key, V value, BiFunction<? super V,​? super V,​? extends V> remappingFunction) | Si la clave especificada aún no está asociada con un valor o está asociada con un valor nulo, la asocia con el valor no nulo proporcionado. |
| static <K,​V> Map<K,​V> ofEntries​(Map.Entry<? extends K,​? extends V>... entries) | Devuelve un mapa no modificable que contiene claves y valores extraídos de las entradas dadas. |
| V put​(K key, V value) | Asocia el valor especificado con la clave especificada en este mapa (operación opcional). |
| default V replace​(K key, V value) | Reemplaza la entrada de la clave especificada solo si actualmente está asignada a algún valor. |
| V remove​(Object key) | Elimina el mapeo de una clave de este mapa si está presente (operación opcional). |
| Collection<V> values() | Devuelve una Collectionvista de los valores contenidos en este mapa. |

*Nota.* Se omitieron algunos métodos, debido a que son métodos sobrecargados con el mismo retorno pero diferentes parámetros o de mínimo interés como: *of(), remove(), putIfAbsent(),computeIfAbsent(), computeIfPresent().*

**Figura 16.** Ejemplo en código de la interfaz Map



## Clase HashMap

La clase HashMap ofrece una implementación eficiente de un mapa no ordenado utilizando una estructura de tabla hash. Es ampliamente utilizada en aplicaciones Java para almacenar y recuperar datos de manera rápida y eficiente utilizando claves, es importante tener en cuenta que HashMap no garantiza un orden específico de iteración de los elementos. Si se necesita un mapa ordenado, se puede considerar utilizar TreeMap o LinkedHashMap, que proporcionan diferentes garantías de ordenamiento según las necesidades específicas.

Paquete: java.util

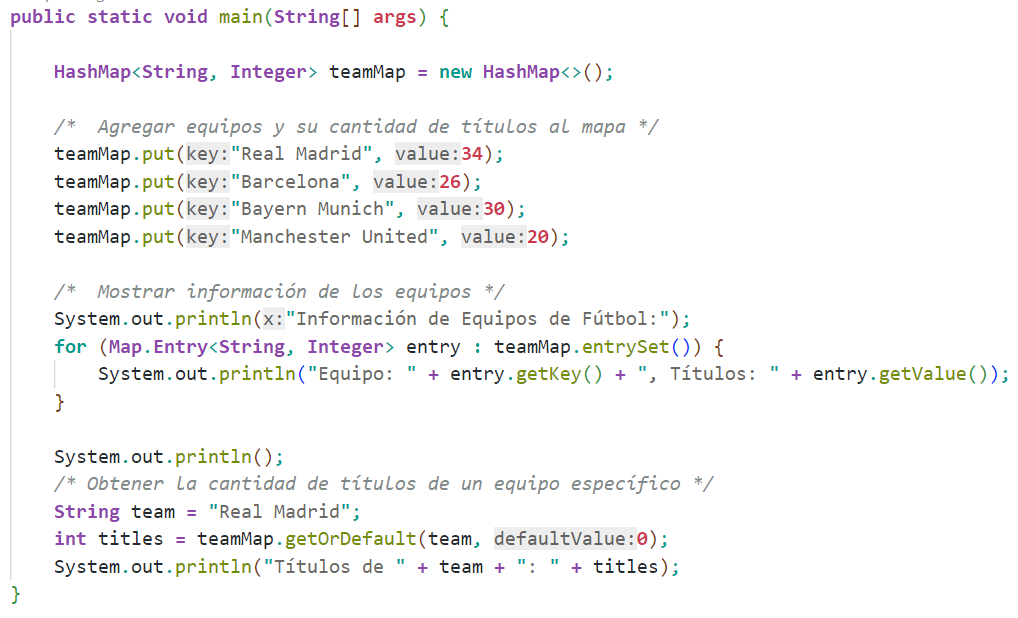
Versión de Java: 1.2

Tabla 17   
*Metodos de la clase HashMap*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| void clear() | Elimina todas las asignaciones de este mapa. |
| Object clone() | Devuelve una copia superficial de esta HashMap instancia: las claves y los valores en sí no se clonan. |
| V compute​(K key, BiFunction<? super K,​? super V,​? extends V> remappingFunction) | Intenta calcular una asignación para la clave especificada y su valor asignado actual (o null si no hay una asignación actual). |
| boolean containsKey​(Object key) | Devuelve true si este mapa contiene un mapeo para la clave especificada. |
| boolean containsValue​(Object value) | Devuelve true si este mapa asigna una o más claves al valor especificado. |
| Set<Map.Entry<K,​V>> entrySet() | Devuelve una Set vista de las asignaciones contenidas en este mapa. |
| V get​(Object key) | Devuelve el valor al que se asigna la clave especificada, o null si este mapa no contiene ninguna asignación para la clave. |
| boolean isEmpty() | Devuelve true si este mapa no contiene asignaciones de clave-valor. |
| Set<K> keySet() | Devuelve una Set vista de las claves contenidas en este mapa. |
| V merge​(K key, V value, BiFunction<? super V,​? super V,​? extends V> remappingFunction) | Si la clave especificada aún no está asociada con un valor o está asociada con un valor nulo, la asocia con el valor no nulo proporcionado. |
| V put​(K key, V value) | Asocia el valor especificado con la clave especificada en este mapa. |
| void putAll​(Map<? extends K,​? extends V> m) | Copia todas las asignaciones del mapa especificado a este mapa. |
| V remove​(Object key) | Elimina la asignación de la clave especificada de esta asignación, si está presente. |
| int size() | Devuelve el número de asignaciones de clave-valor en esta asignación. |
| Collection<V> values() | Devuelve una Collection vista de los valores contenidos en este mapa. |

*Nota.* Se omitieron algunos métodos sobrecargados o no tan relevantes, puesto que esta clase implementa la interfaz Map.

**Figura 17.** Ejemplo en codigo de la clase HashMap



## 

## Clase TreeMap

La clase TreeMap en la API de Colecciones de Java es una implementación de la interfaz SortedMap que utiliza un árbol binario de búsqueda balanceado para almacenar los pares clave-valor. Su objetivo principal es proporcionar una implementación eficiente de un mapa ordenado en el que los elementos se mantienen ordenados según el orden natural de las claves o un comparador personalizado, el uso de TreeMap es preferible cuando se necesita un orden específico en los elementos del mapa.

Paquete: java.util

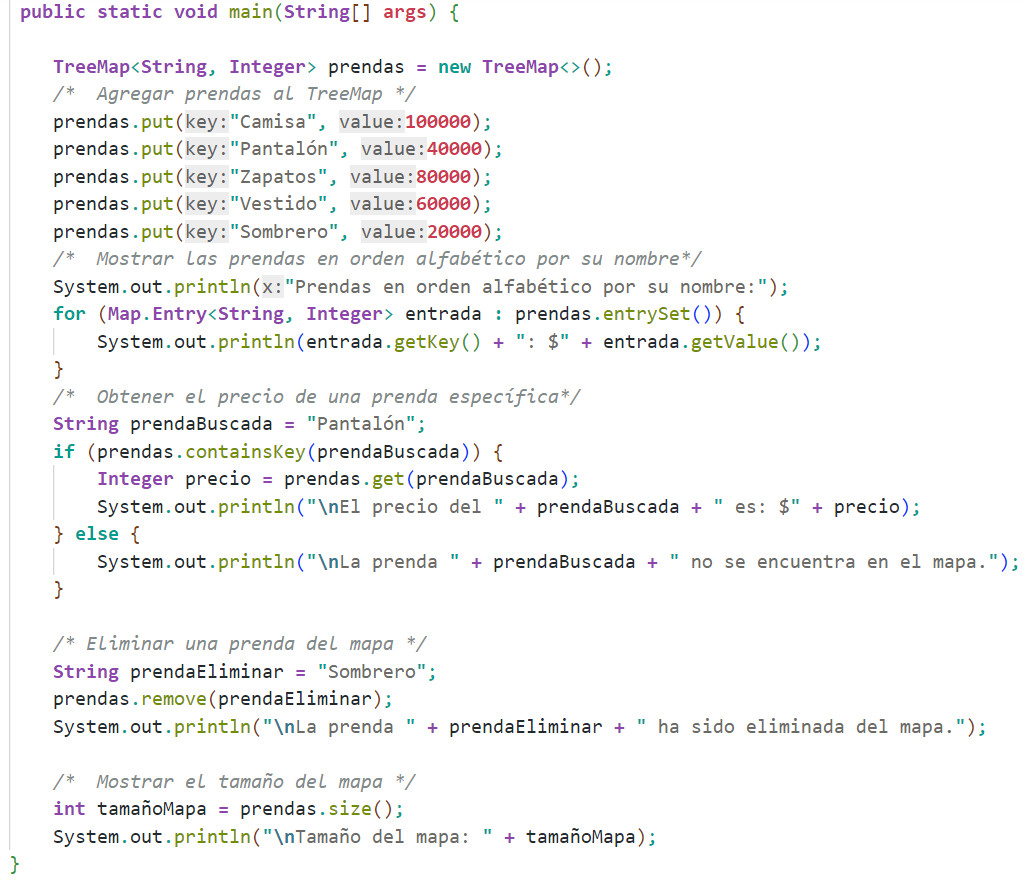
Versión de Java: 1.2

Tabla 18   
*Metodos de la clase TreeMap*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| Map.Entry<K,​V> ceilingEntry​(K key | Devuelve una asignación de clave-valor asociada con la clave mínima mayor o igual que la clave dada, o null si no existe tal clave. |
| K ceilingKey​(K key) | Devuelve la clave mínima mayor o igual que la clave dada, o null si no existe tal clave. |
| void clear() | Elimina todas las asignaciones de este mapa. |
| Object clone() | Devuelve una copia superficial de esta TreeMap instancia. |
| boolean containsKey​(Object key) | Devuelve true si este mapa contiene un mapeo para la clave especificada. |
| boolean containsValue​(Object value) | Devuelve true si este mapa asigna una o más claves al valor especificado. |
| NavigableSet<K> descendingKeySet() | Devuelve una NavigableSet vista en orden inverso de las claves contenidas en este mapa. |
| NavigableMap<K,​V> descendingMap() | Devuelve una vista en orden inverso de las asignaciones contenidas en este mapa. |
| Set<Map.Entry<K,​V>> entrySet() | Devuelve una Set vista de las asignaciones contenidas en este mapa. |
| Map.Entry<K,​V> firstEntry() | Devuelve una asignación de clave-valor asociada con la clave mínima en este mapa, o null si el mapa está vacío. |
| K firstKey() | Devuelve la primera clave (más baja) actualmente en este mapa. |
| Map.Entry<K,​V> floorEntry​(K key) | Devuelve una asignación de clave-valor asociada con la mayor clave menor o igual que la clave dada, o null si no existe tal clave. |
| K floorKey​(K key) | Devuelve la mayor clave menor o igual a la clave dada, o null si no existe tal clave. |
| V get​(Object key) | Devuelve el valor al que se asigna la clave especificada, o null si este mapa no contiene ninguna asignación para la clave. |
| SortedMap<K,​V> headMap​(K toKey) | Devuelve una vista de la parte de este mapa cuyas claves son estrictamente menores que toKey. |
| Map.Entry<K,​V> higherEntry​(K key) | Devuelve una asignación de clave-valor asociada con la clave mínima estrictamente mayor que la clave dada, o null si no existe tal clave. |
| K higherKey​(K key) | Devuelve la clave mínima estrictamente mayor que la clave dada, o null si no existe tal clave. |
| Set<K> keySet() | Devuelve una Set vista de las claves contenidas en este mapa. |
| Map.Entry<K,​V> lastEntry() | Devuelve una asignación de clave-valor asociada con la clave mayor en este mapa, o null si el mapa está vacío. |
| K lastKey() | Devuelve la última clave (más alta) actualmente en este mapa. |
| Map.Entry<K,​V> lowerEntry​(K key) | Devuelve una asignación de clave-valor asociada con la clave mayor estrictamente menor que la clave dada, o null si no existe tal clave. |
| K lowerKey​(K key) | Devuelve la clave mayor estrictamente menor que la clave dada, o null si no existe tal clave. |
| NavigableSet<K> navigableKeySet() | Devuelve una NavigableSet vista de las claves contenidas en este mapa. |
| Map.Entry<K,​V> pollFirstEntry() | Elimina y devuelve una asignación de clave-valor asociada con la clave mínima en este mapa, o null si el mapa está vacío. |
| Map.Entry<K,​V> pollLastEntry() | Elimina y devuelve una asignación de clave-valor asociada con la clave más grande en este mapa, o null si el mapa está vacío. |
| V put​(K key, V value) | Asocia el valor especificado con la clave especificada en este mapa. |
| void putAll​(Map<? extends K,​? extends V> map) | Copia todas las asignaciones del mapa especificado a este mapa. |
| NavigableMap<K,​V> subMap​(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive) | Devuelve una vista de la parte de este mapa cuyas claves van de fromKey a toKey. |
| SortedMap<K,​V> tailMap​(K fromKey) | Devuelve una vista de la parte de este mapa cuyas claves son mayores o iguales que fromKey. |

*Nota*. Se hicieron omisiones de métodos, debido a que implementa varios sobrescritos de manteniendo la misma funcionalidad y sobrecargados con distintos parámetros, no obstante, se mencionaron los relevantes.

**Figura 18.** Ejemplo en codigo de la clase TreeMap



## Clase LinkedHashMap

La clase LinkedHashMap es una implementación de la clase Map y extiende de la clase HashMap, siendo una opción adecuada cuando se necesita mantener el orden de inserción de los elementos en un mapa, además de las ventajas de acceso rápido proporcionadas por HashMap. Es útil en situaciones donde se requiere mantener un seguimiento del orden de inserción de los elementos o donde el orden de los elementos es relevante, es importante tener en cuenta que el mantenimiento del orden de inserción en LinkedHashMap conlleva una ligera sobrecarga en términos de rendimiento y uso de memoria en comparación con HashMap.

Paquete: java.util

Versión de Java: 1.4

Tabla 19   
*Metodos de la clase LinkedHashMap*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| boolean containsValue​(Object value) | Devuelve true si este mapa asigna una o más claves al valor especificado. |
| Set<Map.Entry<K,​V>> entrySet() | Devuelve una Set vista de las asignaciones contenidas en este mapa. |
| V get​(Object key) | Devuelve el valor al que se asigna la clave especificada, o null si este mapa no contiene ninguna asignación para la clave. |
| Set<K> keySet() | Devuelve una Set vista de las claves contenidas en este mapa. |
| protected boolean removeEldestEntry​(Map.Entry<K,​V> eldest) | Devuelve true si este mapa debe eliminar su entrada más antigua. |
| Collection<V> values() | Devuelve una Collection vista de los valores contenidos en este mapa. |

*Nota*. Los métodos mencionados en la Tabla 19, son independientes a las que se heredan y implementan (sobrescritura).

## Interfaz Stream

**Figura 19.** Ejemplo en codigo de la clase LinkedHashMap

La interfaz Stream es una secuencia de elementos que permite realizar operaciones en forma de flujo de datos de manera declarativa y paralela. Su objetivo principal es proporcionar una forma eficiente y expresiva de trabajar con colecciones de elementos, permitiendo realizar operaciones como filtrado, mapeo, ordenamiento y reducción de manera más concisa y funcional; Es importante destacar que los flujos (Stream) son inmutables, lo que significa que no modifican los datos subyacentes. En cambio, generan nuevos flujos con las transformaciones y operaciones aplicadas.

Paquete: java.util.stream

Versión de Java: 1.8

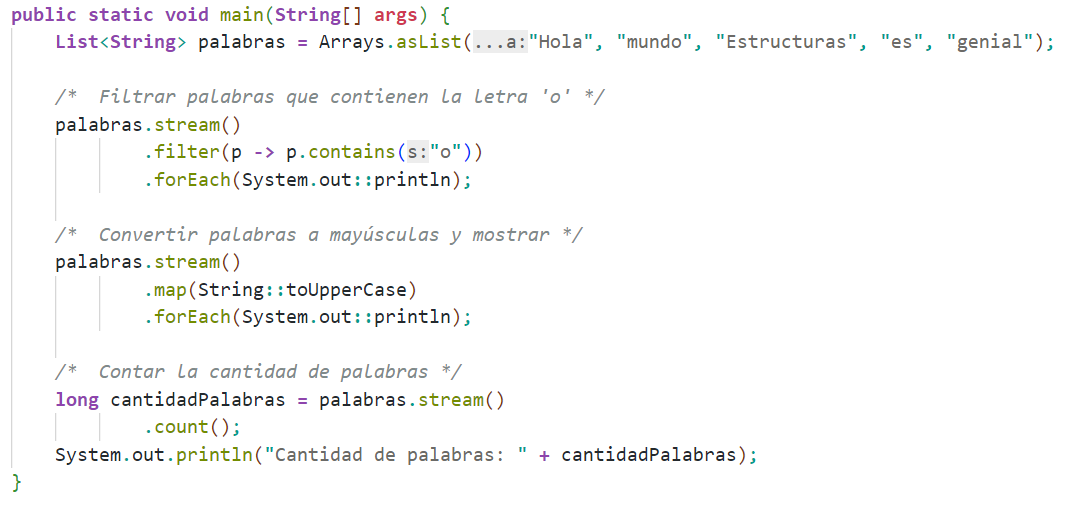
Tabla 20   
*Metodos de la interfaz Stream*

|  |  |
| --- | --- |
| ***Métodos*** | ***Descripción*** |
| filter(Predicate<? super T> predicate) | Filtra los elementos del Stream según el predicado especificado. |
| map(Function<? super T, ? extends R> mapper) | Transforma los elementos del Stream aplicando la función especificada. |
| forEach(Consumer<? super T> action) | Realiza una acción en cada elemento del Stream. |
| collect(Collector<? super T, A, R> collector) | Recolecta los elementos del Stream en una colección o en otro tipo de objeto. |
| reduce(T identity, BinaryOperator<T> accumulator) | Reduce los elementos del Stream a un solo valor aplicando una operación de reducción y un valor inicial. |
| sorted() | Ordena los elementos del Stream en orden natural. |
| distinct() | Elimina los elementos duplicados del Stream |
| limit(long maxSize) | Limita el tamaño del Stream a un número máximo de elementos. |
| skip(long n) | Salta los primeros n elementos del Stream. |
| anyMatch(Predicate<? super T> predicate) | Verifica si al menos un elemento del Stream cumple con el predicado especificado. |

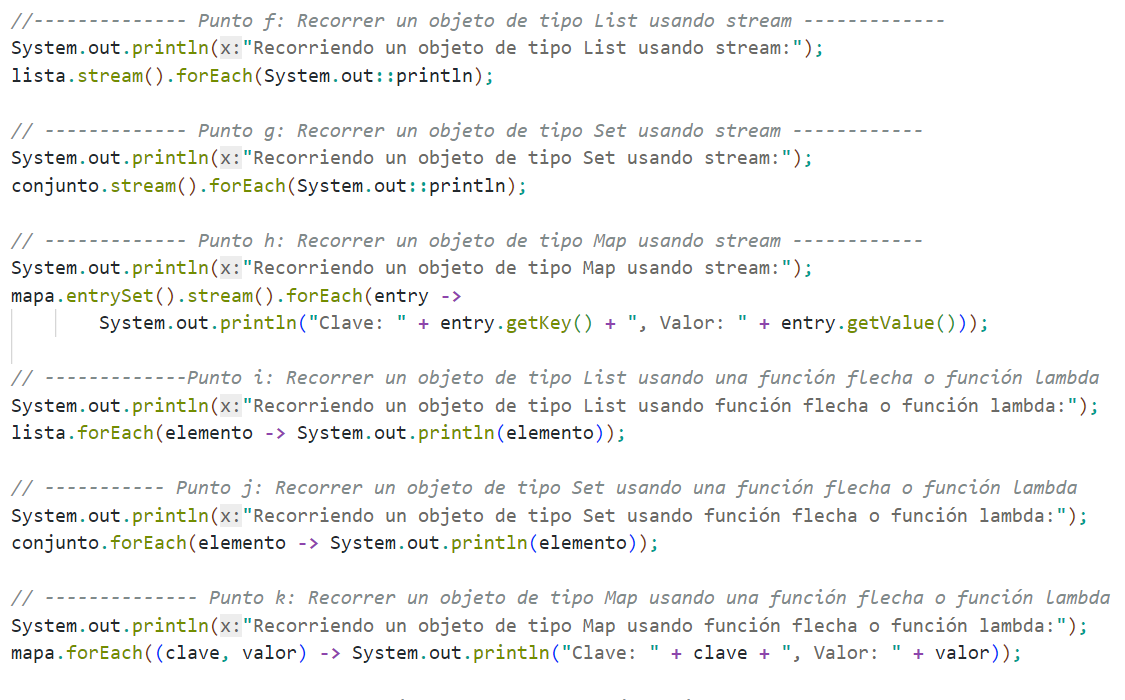
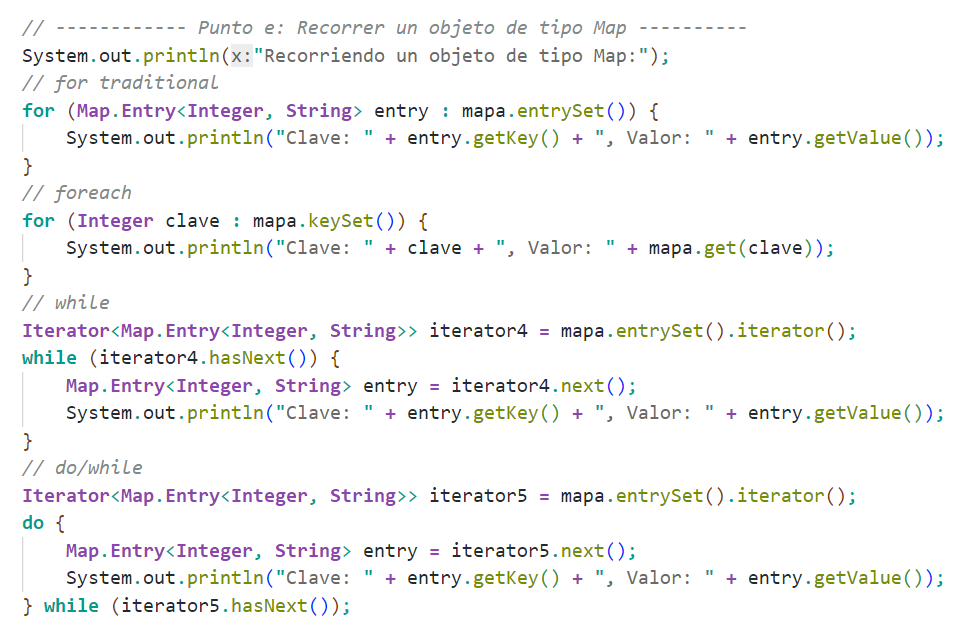
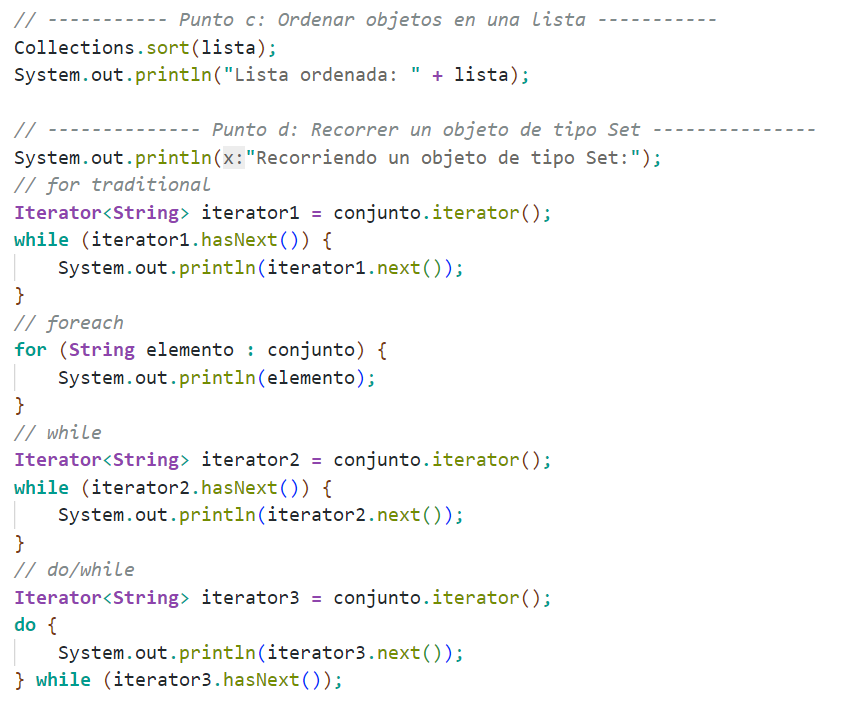
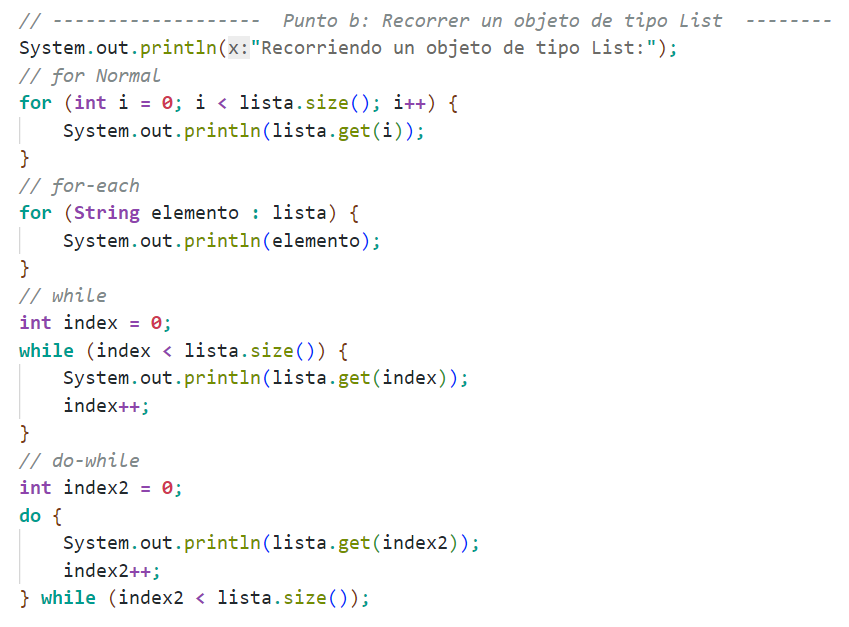
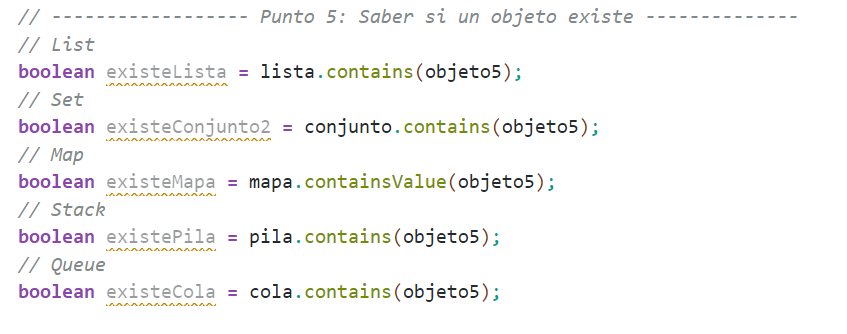
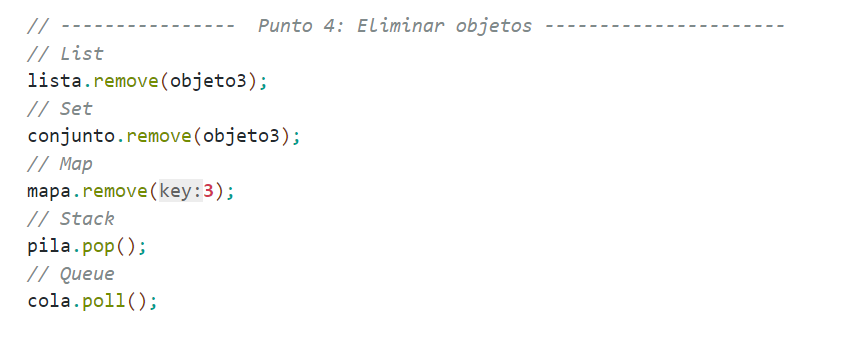
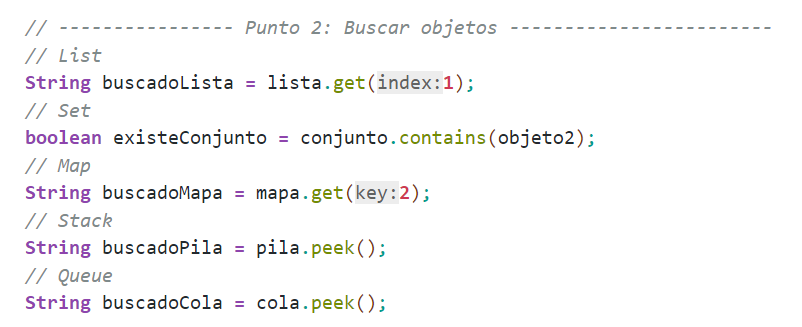
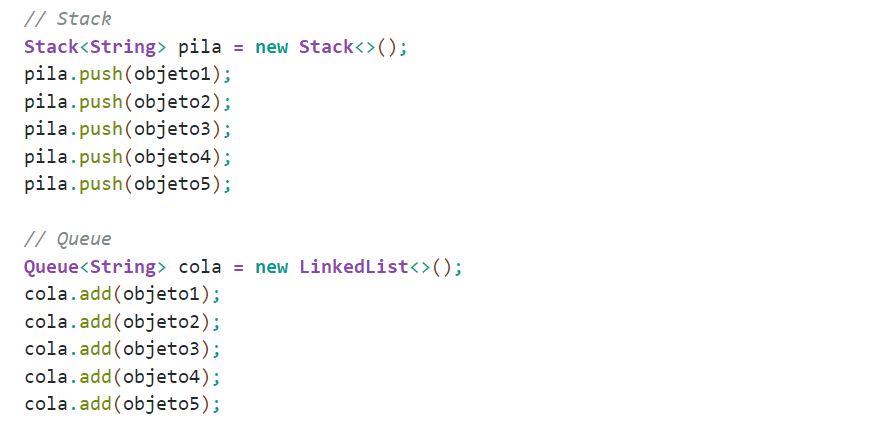
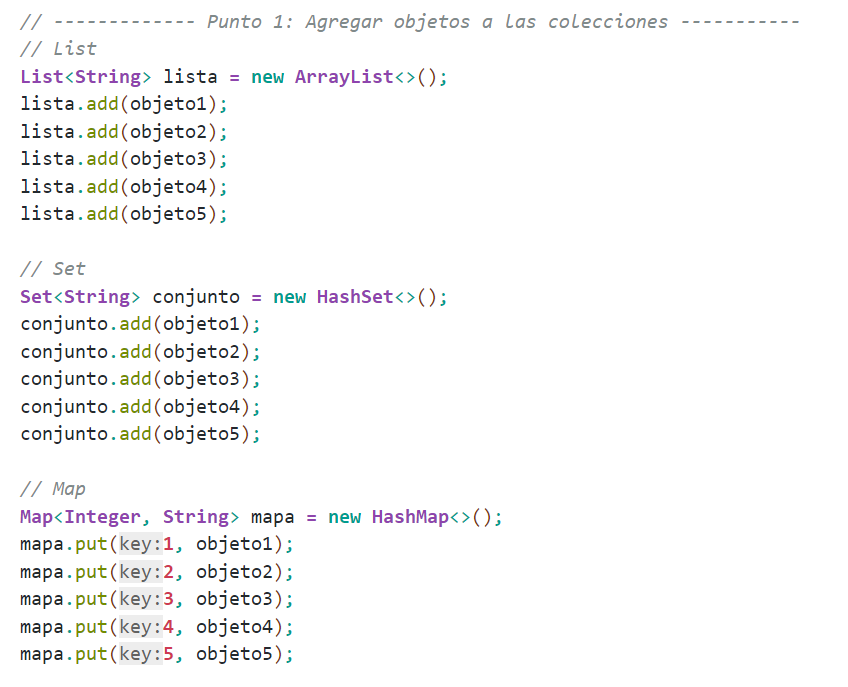
*Nota*. La interfaz Stream es bastante compleja, por ello en la tabla se representan los métodos mas importantes o mayormente implementados, el resto de métodos véase en la bibliografía.

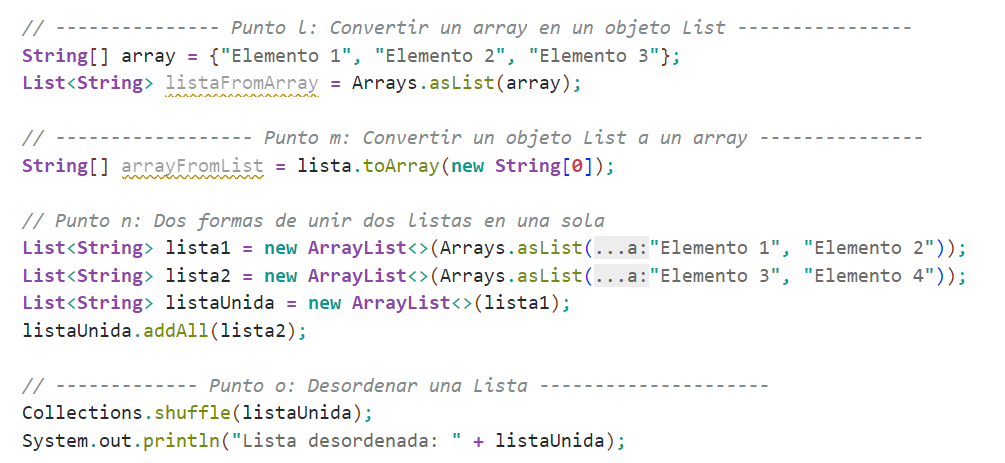
# Desarrollo Creación de un Programa

**Figura 20.** Ejemplo en codigo de la interfaz Stream



***código fuente***: ColeccionesPrograma.java





Se buscó resumir todo en esta sección del código, que contiene los funcionamientos básicos de las clases List, Set, Map, Queue y Stack, junto con otras funcionalidades.

# Lista de referencias

*ArrayList (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/ArrayList.html

*Arrays (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Arrays.html

*Collection (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Collection.html

*Collections (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Collections.html

*Dictionary (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Dictionary.html

*Enumeration (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Enumeration.html

*HashMap (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/HashMap.html

*HashSet (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/HashSet.html

*Hashtable (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Hashtable.html

*LinkedHashMap (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/LinkedHashMap.html

*LinkedList (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/LinkedList.html

*List (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/List.html

*Map (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Map.html

*Properties (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Properties.html

*Set (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Set.html

*SortedSet (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/SortedSet.html

*Stream (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/stream/Stream.html

*TreeMap (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/TreeMap.html

*TreeSet (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/TreeSet.html

*Vector (java SE 14 & JDK 14)*. (s/f). Oracle.com. , de https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Vector.html