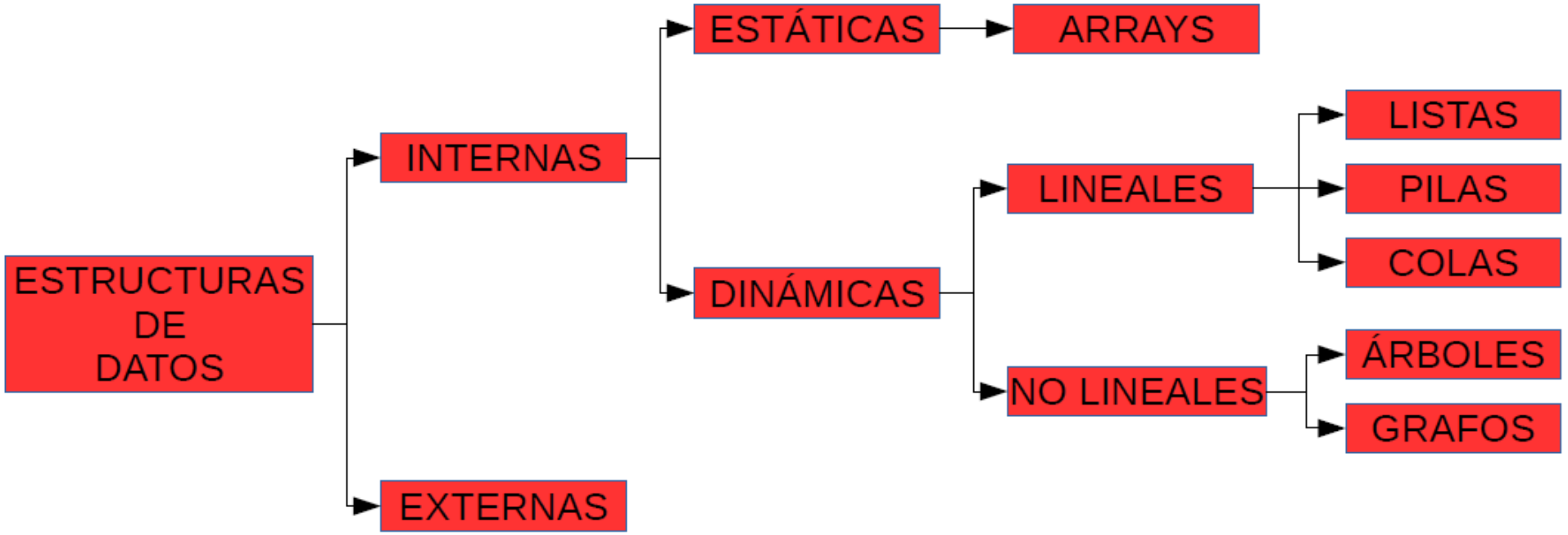
UD5: Almacenamiento de la información en estructuas de datos.

IES LOS SAUCES – BENAVENTE
CFGS DESARROLLO DE APLICACIONES WEB
PROGRAMACIÓN

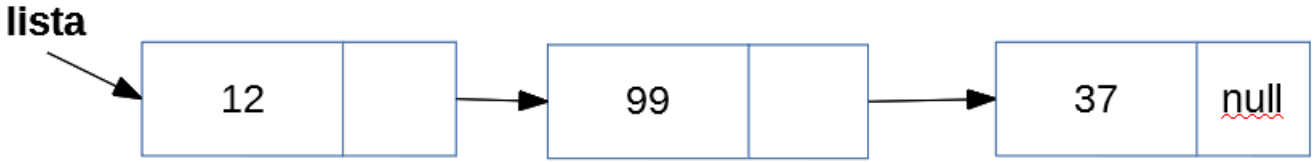
Estructuras de datos

→En programación, una estructura de datos es una forma de organizar un conjunto de datos con el objetivo de facilitar su manipulación.



Listas enlazadas

- **Una lista enlazada es una estructura de datos formada por una secuencia de elementos (nodos) dispuestos uno detrás de otro, en la que cada elemento se conecta al siguiente elemento mediante una referencia.**



- **Los nodos se componen de dos partes (campos): datos y referencia (apunta al siguiente elemento de la lista).**

Listas enlazadas

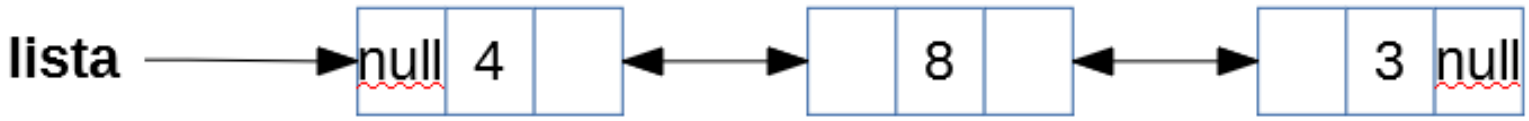
Tipos de listas enlazadas

- **Lista simplemente enlazada: cada nodo contiene una única referencia al nodo siguiente, o a null si se trata del último nodo.**

Listas enlazadas

Tipos de listas enlazadas

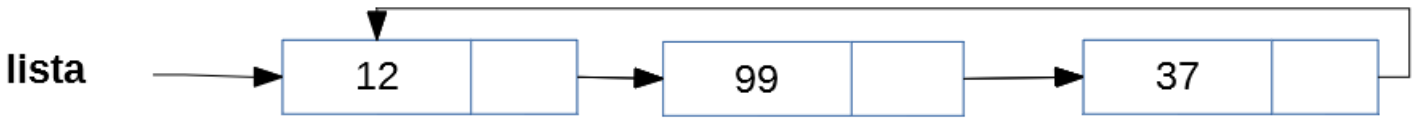
- **Lista doblemente enlazada: cada nodo contiene dos referencias, una al nodo anterior, o a null si se trata del primer nodo, y otra al nodo siguiente, o a null si se trata del último nodo.**



Listas enlazadas

Tipos de listas enlazadas

- **Lista circular simplemente enlazada: cada nodo contiene una única referencia al nodo siguiente, o al primer nodo si se trata del último nodo.**



Listas enlazadas

Tipos de listas enlazadas

- **Lista circular doblemente enlazada: Cada nodo contiene dos referencias, una al nodo anterior, o al último nodo si se trata del primer nodo, y otra al nodo siguiente, o al primer nodo si se trata del último nodo.**

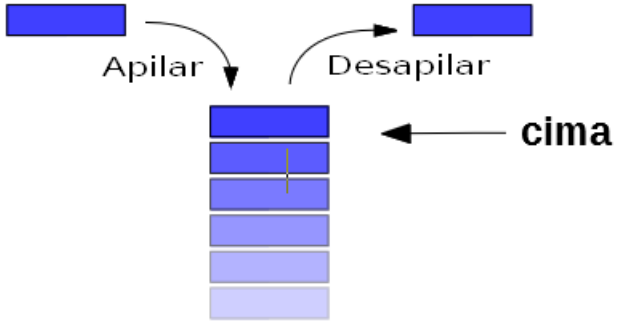
Listas enlazadas

Operaciones sobre listas enlazadas (https://visualgo.net/es/list)

- ➔ **Recorrido de los elementos de la lista**
- ➔ **Insertar un elemento en la lista**
  - **En la cabeza de la lista**
  - **En el final de la lista**
  - **Entre dos nodos de la lista**
- ➔ **Borrar un elemento de la lista**
  - **En la cabeza de la lista**
  - **En otra posición de la lista**
- ➔ **Buscar un elemento en la lista**

Pilas

- **Una pila (stack) es una estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (Last In First Out) que permite almacenar y recuperar datos.**



- **Los elementos se añaden o se eliminan de la pila sólo por su parte superior (cima).**

Pilas

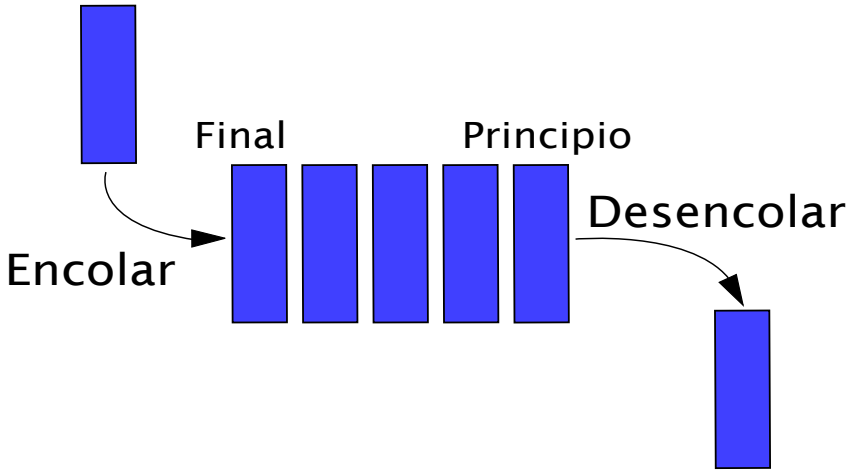
Operaciones sobre pilas (https://visualgo.net/es/list)

- ➔ **Apilar (push)**
- ➔ **Desapilar (pop)**
- ➔ **Cima (peek)**
- ➔ **Limpiar pila**
- ➔ **Pila vacía**

Una pila se puede implementar mediante un array o mediante una lista enlazada.

Colas

- **Una cola (queue) es una estructura de datos en la que el modo de acceso a sus elementos es de tipo FIFO (First In First Out) que permite almacenar y recuperar datos.**



- **Los elementos se añaden por el final de la cola y se eliminan por el principio de la cola.**

Colas

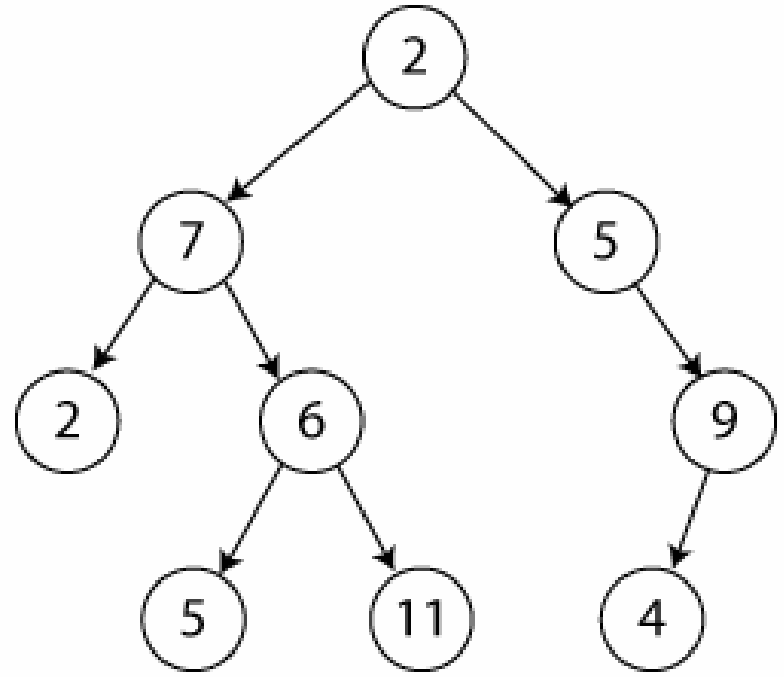
Operaciones sobre colas (https://visualgo.net/es/list)

- ➔ **Encolar (enqueue)**
- ➔ **Desencolar (dequeue)**
- ➔ **Frente (peek)**
- ➔ **Borrar cola**
- ➔ **Cola vacía**

Una cola se puede implementar mediante un array o mediante una lista enlazada.

Árboles

- **Un árbol es una estructura de datos no lineal que puede representarse como un conjunto de nodos enlazados entre sí por medio de ramas.**



Árboles

Tipos de árboles

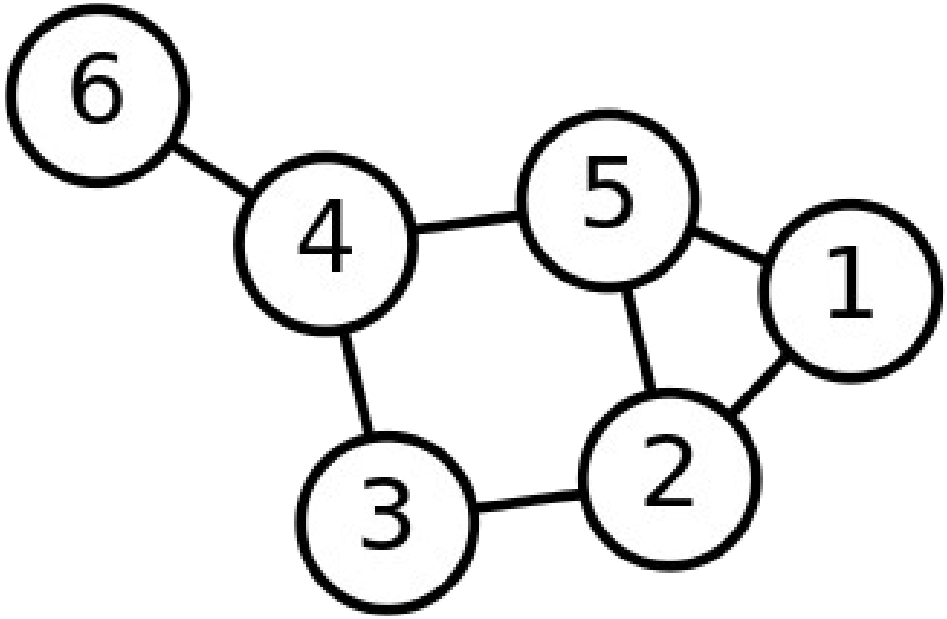
- **Árbol binario: https://es.wikipedia.org/wiki/Árbol_binario**
  - **de búsqueda: https://es.wikipedia.org/wiki/Árbol_binario_de_búsqueda**
  - **de búsqueda auto-balanceable: https://es.wikipedia.org/wiki/Árbol_binario_de_búsqueda_auto-balanceable**
- **Árbol AVL: https://es.wikipedia.org/wiki/Árbol_AVL**
- **Árbol multicamino: https://es.wikipedia.org/wiki/Árbol_multicamino**
- **Árbol B: https://es.wikipedia.org/wiki/Árbol-B**

Operaciones sobre árboles

- Insertar elementos
- Buscar elementos
- Borrar elementos
- Recorrer el árbol (https://es.wikipedia.org/wiki/Recorrido_de_árboles)
  - Recorrido en árboles binarios
    - **Preorden:** nodo raíz, sub-árbol izquierdo, sub-árbol derecho
    - **Inorden:** sub-árbol izquierdo, nodo raíz, sub-árbol derecho
    - **Postorden:** sub-árbol izquierdo, sub-árbol derecho, nodo raíz

Grafos

- **Un grafo es una estructura de datos no lineal que consiste en un conjunto de nodos (vértices) y un conjunto de arcos (aristas) que establecen relaciones entre los nodos.**

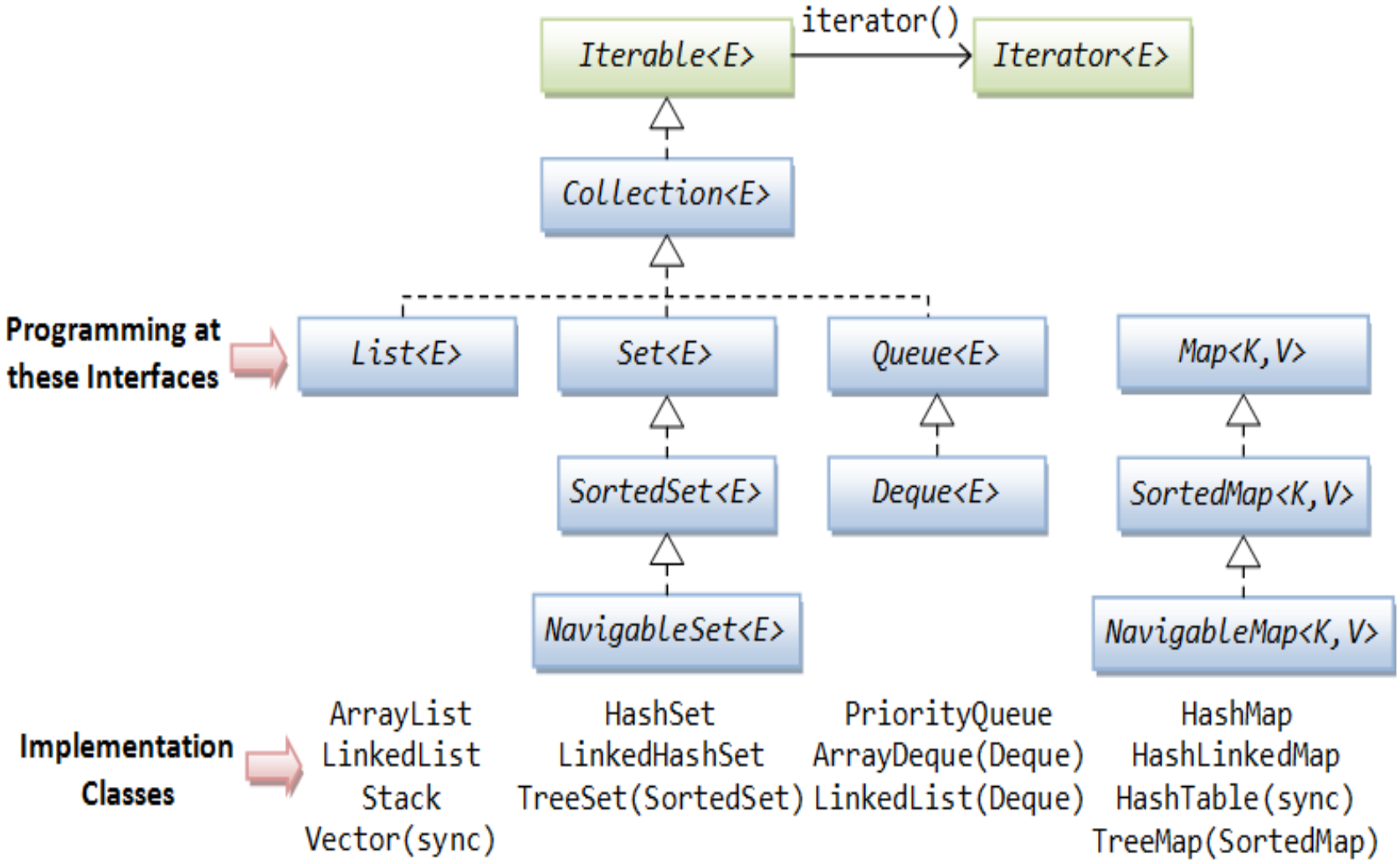


Colecciones

La API de Java provee varias estructuras de datos predefinidas, conocidas como colecciones, que se utilizan para almacenar grupos de objetos relacionados.

Para utilizarlas usaremos el Java Collections Framework (JCF), el cual contiene un conjunto de clases e interfaces del paquete java.util para gestionar colecciones de objetos.

Colecciones



Colecciones

interfaces	
Collection<E>	Define métodos para tratar una colección genérica de elementos
List<E>	Admite elementos repetidos y mantiene un orden inicial
Set<E>	Colección que no admite elementos repetidos
SortedSet<E>	Set cuyos elementos se mantienen ordenados según el criterio establecido
Queue<E>	Colección ordenada de elementos por el orden en el que van a ser procesados. Admite elementos repetidos
Map<K,V>	Conjunto de pares clave/valor, sin repetición de claves
SortedMap<K,V>	Map cuyos elementos se mantienen ordenados según el criterio establecido

Colecciones

clases	
ArrayList<E>	Agrupa elementos como un array
LinkedList<E>	Organiza los elementos a la manera de una lista doblemente enlazada
Vector<E>	Igual que ArrayList pero sincronizado
Stack<E>	Define las operaciones del tipo Pila
HashSet<E>	Guarda los elementos de un conjunto sin mantener un orden
LinkedHashSet<E>	Igual que HashSet pero ordenado por orden de inserción
TreeSet<E>	Guarda los elementos de un conjunto manteniendo el orden de los elementos. Los elementos del conjunto se organizan en un árbol
PriorityQueue<E>	Cola con elementos ordenados
HashMap<K,V>	Organiza los elementos en una tabla hash
LinkedHashMap<K,V>	Utiliza una lista doblemente enlazada
HashTable<K,V>	Igual que HashMap pero sincronizado
TreeMap<K,V>	Mantiene el orden de sus elementos atendiendo al campo clave

Collection<E>

interface Collection<E>

- **Describe el comportamiento común de las colecciones en Java.**
- **Declara métodos que serán implementados por las distintas clases.**
- **Métodos para añadir elementos a la colección.**
- **Métodos para eliminar elementos de la colección.**
- **Métodos para buscar elementos en la colección.**
- **Métodos de colección.**

Collection<E>

Métodos para añadir elementos a la colección
`boolean add(E e)`
`boolean addAll(Collection <? extends E> c)`
Métodos para eliminar elementos de la colección
`boolean remove(Object o)`
`boolean removeAll(Collection<?> c)`
`void clear()`
`boolean retainAll(Collection<?> c)`
Métodos para buscar elementos en la colección
`boolean contains(Object o)`
`boolean containsAll(Collection<?> c)`
`boolean equals(Object o)`

Collection<E>

Métodos de colección
`Iterator<E> iterator()`
`Object[] toArray()`
`boolean isEmpty()`
`int size()`

List<E>

Métodos para añadir elementos a la lista
void add(int index, E element)
boolean addAll(int index, Collection<? extends E> c)
E set(int index, E element)
Métodos para eliminar elementos de la lista
E remove(int index)
Métodos para buscar elementos en la lista
E get(int index)
int indexOf(Object o)
int lastIndexOf(Object o)
Métodos de lista
List<E> subList(int fromIndex, int toIndex)
ListIterator<E> listIterator()
ListIterator<E> listIterator(int index)

Iterator<E> y ListIterator<E>

Métodos de la interface Iterator
boolean hasNext()
E next()
void remove()
Métodos de la interface ListIterator
boolean hasNext()
boolean hasPrevious()
E next()
E previous()
void remove()
int nextIndex()
int previousIndex()
void add(E e)
void set(E e)

Vector<E>

clase Vector<E>

- **Implementa una lista de elementos mediante un array de tamaño variable.**
- **Conforme se añaden elementos el tamaño del array irá creciendo si es necesario.**
- **El array tendrá una capacidad inicial, y en el momento en el que se rebase dicha capacidad, se aumentará el tamaño del array.**
- **Los métodos están sincronizados.**
- **No se recomienda usar.**

ArrayList<E>

clase ArrayList<E>

- **Es equivalente a la clase Vector.**
- **Es preferible usar la clase ArrayList a la clase Vector.**
- **Constructores:**

```
public ArrayList()

public ArrayList(int initialCapacity)

public ArrayList(Collection<? extends E> c)
```

LinkedList<E>

clase LinkedList<E>

- **Organiza los elementos de la colección a la manera de una lista doblemente enlazada.**

- **Más lento para iterar, más rápido para insertar y eliminar.**

- **Se puede usar para crear una estructura de datos Cola o Pila.**

- **Constructores:**

```
public LinkedList()

public LinkedList(Collection<? extends E> c)
```

LinkedList<E>

clase LinkedList<E>

- **Métodos:**

```
public E getFirst()

public E getLast()

public E removeFirst()

public E removeLast()

public void addFirst(E e)

public void addLast(E e)
```

Stack<E>

clase Stack<E>

- **Constructor**

```
public Stack()
```

- **Métodos:**

```
public E push(E item)

public E pop()

public E peek()

public boolean empty()
```

Set<E>

interface Set<E>

- **Se utiliza para trabajar con conjuntos.**
- **Un conjunto es una colección de elementos no duplicados.**
- **Declara los mismos métodos que Collection, pero con las restricciones que evitan tener elementos duplicados.**
- **Clases implementadoras: HashSet, LinkedHashSet y TreeSet.**

Map<K,V>

interface Map<K,V>

- **Colecciones que almacenan pares clave/valor sin repeticiones en las claves.**
- **No deriva de Collection.**
- **Clases implementadoras: HashMap, HashTable, TreeMap y LinkedHashMap.**

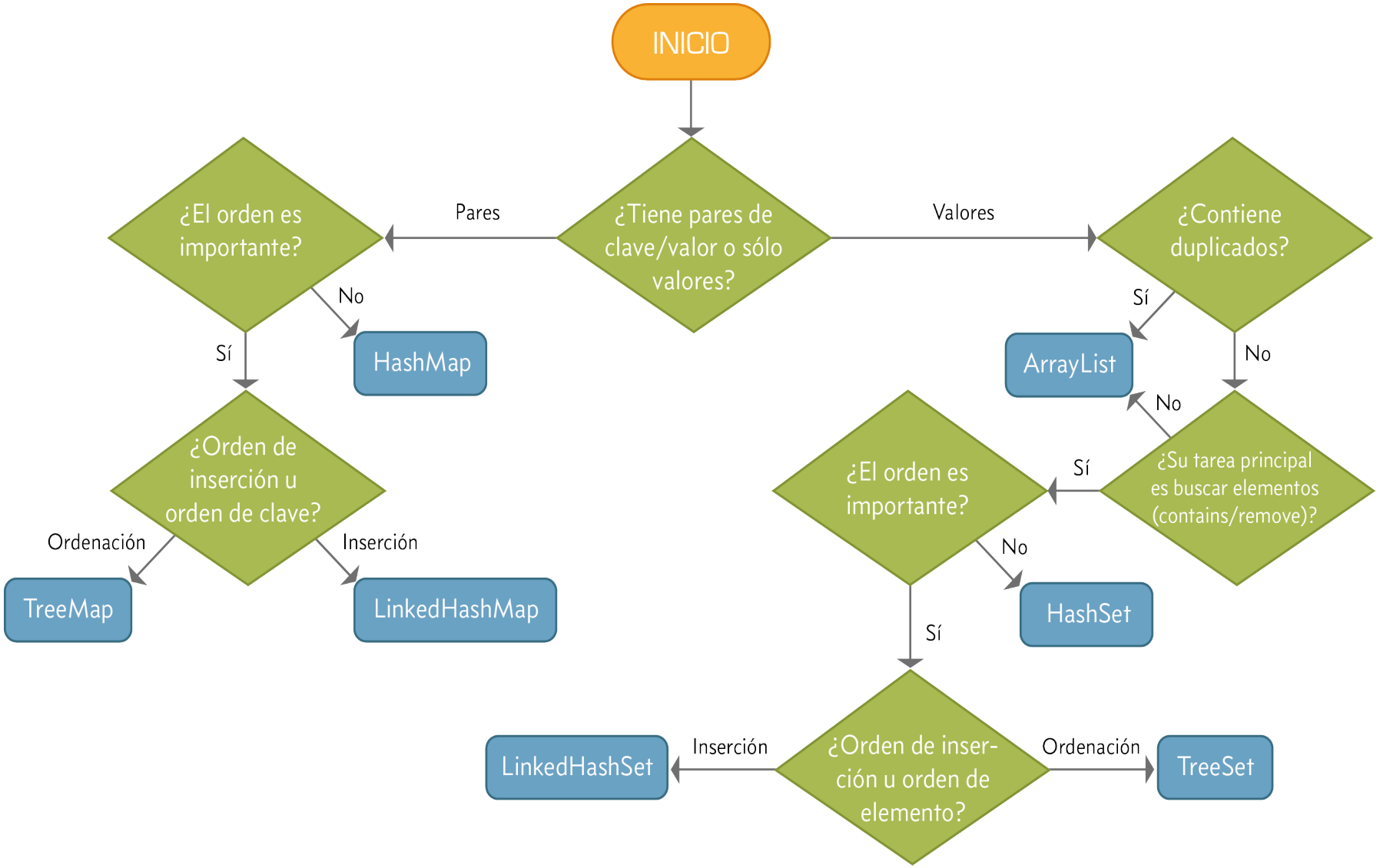
Map<K,V>

interface Map<K,V>

- **Métodos:**

```
V put(K key, V value)

void putAll(Map<? extends K,? extends V> m)

V remove(Object key)

void clear()

V get(Object key)

boolean isEmpty()
```

Diagrama de decisión para uso de colecciones Java



Comparación de objetos

Numerosas operaciones con colecciones exigen que sus elementos sean comparables. Esta propiedad se establece a nivel de clase, implementando la interface Comparable (java.lang) o la interface Comparator (java.util).

Comparable<T>

La interface Comparable<T> se utiliza para establecer un orden natural entre los objetos de una misma clase.

Toda clase que implemente la interface Comparable<T> debe implementar el método:

```
public int compareTo(T o)
```

- Devuelve un valor negativo si el objeto que llama al método es menor que el pasado en el argumento.
- Devuelve un cero si son iguales.
- Devuelve un valor positivo si el objeto que llama al método es mayor que el pasado en el argumento.

Comparator<T>

Hay métodos de ordenación y búsqueda de objetos que utilizan la interface Comparator<T> para determinar el orden natural entre dos elementos.

Toda clase que implemente la interface Comparator<T> debe implementar los métodos:

```
int compare(T o1, T o2)
```

- Devuelve un valor negativo si o1 es menor que o2.
- Devuelve un cero si son iguales.
- Devuelve un valor positivo si o1 es mayor que o2.

```
boolean equals(Object obj)
```

Collections

La clase Collections, perteneciente al paquete java.util, contiene métodos estáticos para trabajar con colecciones.

Los métodos implementan algoritmos genéricos de ordenación, búsqueda, máximo y mínimo.

Además, dispone de métodos para dotar de las características de sincronización a una colección, y para convertir a una colección de sólo lectura.

No se pueden crear objetos de la clase Collections.