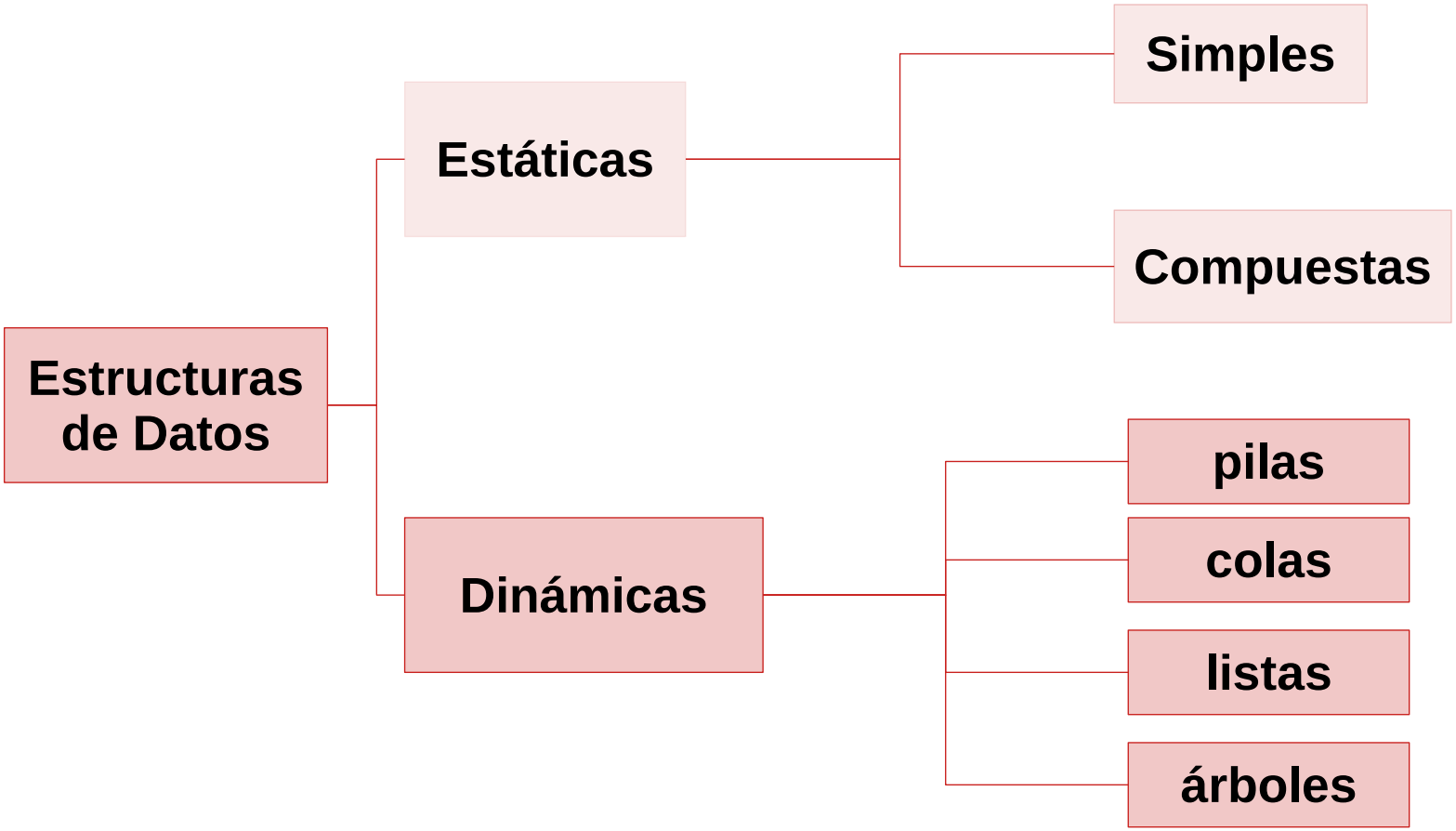
La Sénia
Institut d'Educació Secundària
Paiporta

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Estructuras dinámicas

Curso 2023 - 2024

1. Estructuras de datos



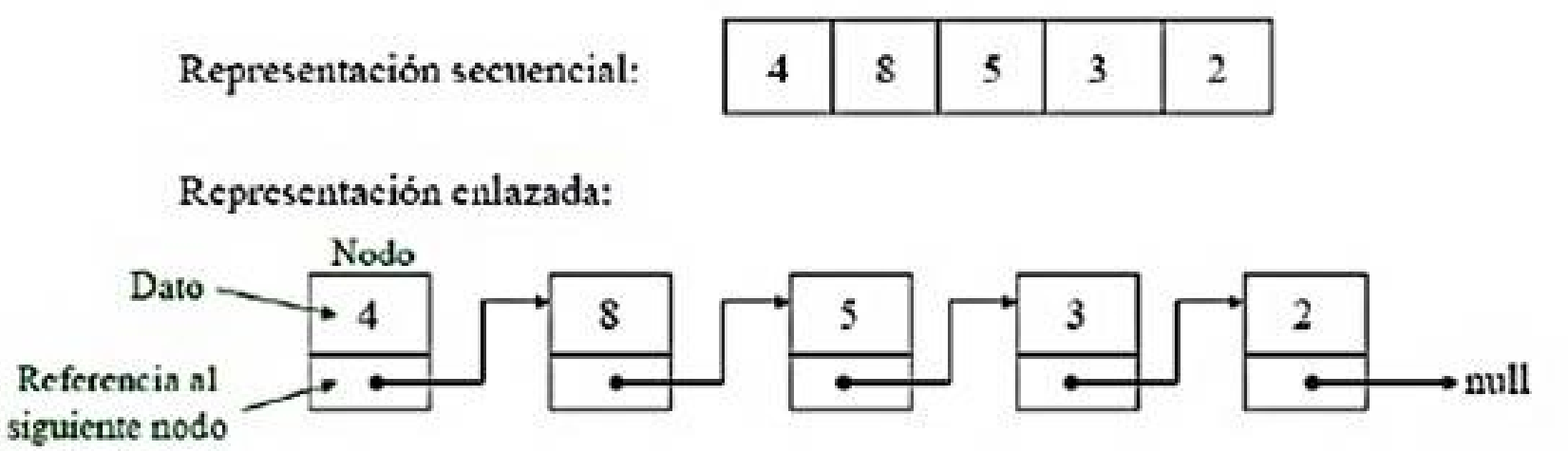
1. Estructuras de datos

- **Estáticas**: el tamaño ocupado en memoria se define con anterioridad a la ejecución del programa. Su dimensión no se modifica.
- **Dinámicas**: reservan un espacio en la memoria que puede crecer o decrecer durante la ejecución del programa.

En Java, las estructuras de datos dinámicas se implementan mediante las **Colecciones:** lista, pila, cola y árbol.

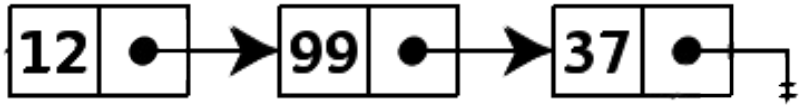
2. Listas

Una lista es un conjunto de elementos llamados **nodos**. Estos nodos se componen de un **dato** y de una **referencia**, enlace o dirección al siguiente elemento o nodo de la lista:

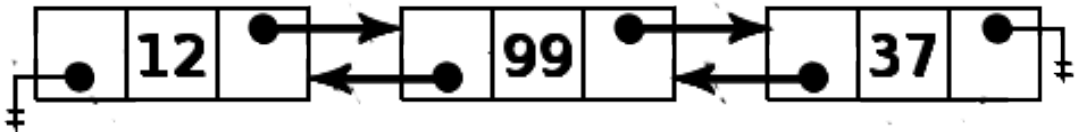


2. Listas

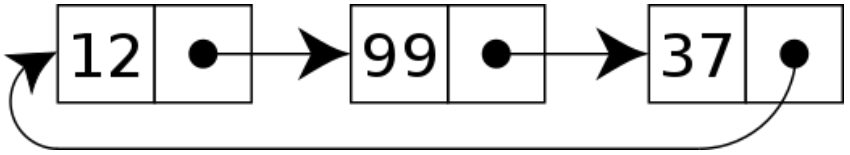
Podemos tener **listas simples**:



Y **listas doblemente enlazadas**, donde además de mantener una referencia al nodo siguiente, se mantiene otra referencia al nodo anterior:

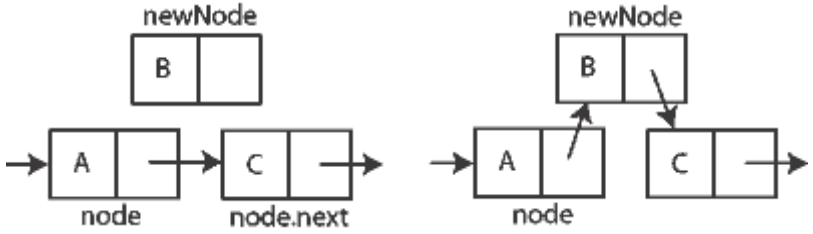


O **listas circulares** donde el último nodo contiene la referencia al primero:

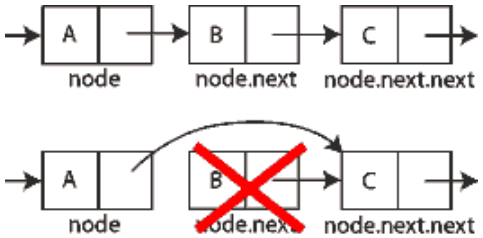


2. Listas

- **Búsqueda**: Se recorre la lista desde el primer elemento hasta llegar al elemento deseado.
- **Inserción** de elemento: Al principio y al final es elemental. Entre nodos se procede como en la imagen:

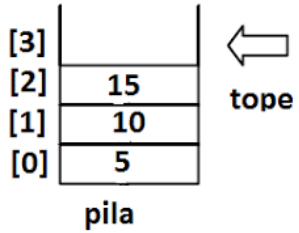


- **Borrado** de elemento: Al principio y al final es elemental. Entre nodos se procede como en la imagen:

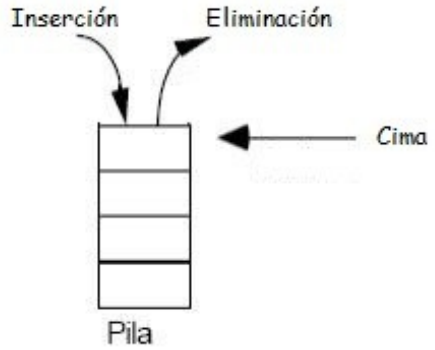


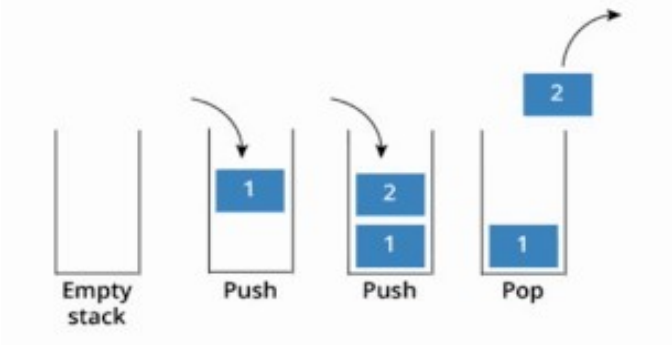
3. Pila

La pila es una estructura de datos LIFO (Last In, First Out) donde los elementos se almacenan de forma que sólo se puede acceder al elemento en el tope de la misma:



Las operaciones posibles son apilar (**push**), desapilar (**pop**) y obtener el elemento cima (**top**):

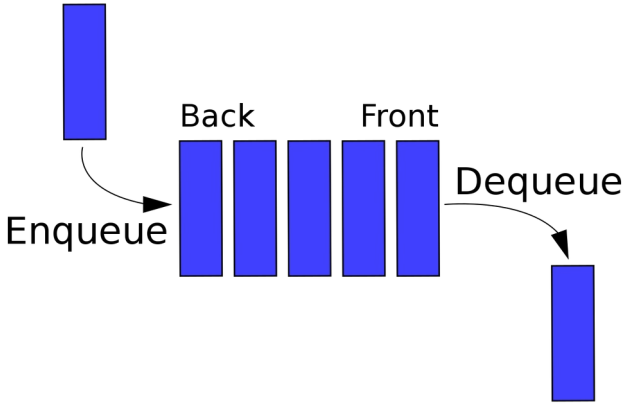




4. Cola

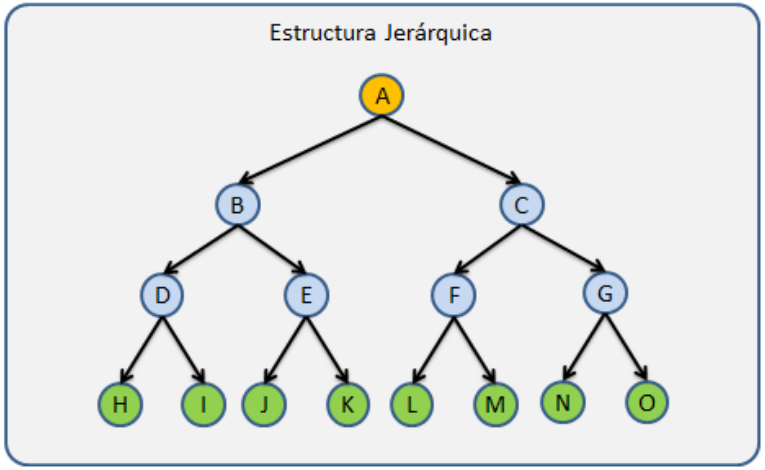
La cola es una estructura de datos FIFO (First In, First Out) donde el primer elemento que se introduce en ella es el primer elemento en salir.

Las operaciones con la cola es encolar (**add**) por el principio o front y desencolar (**remove**) por el final o rear

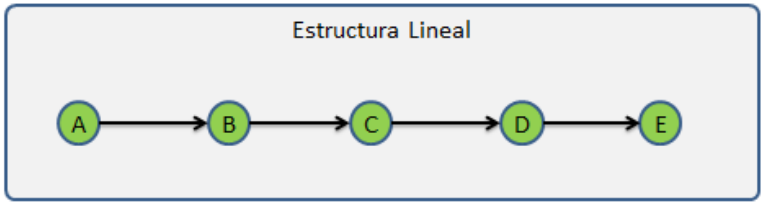


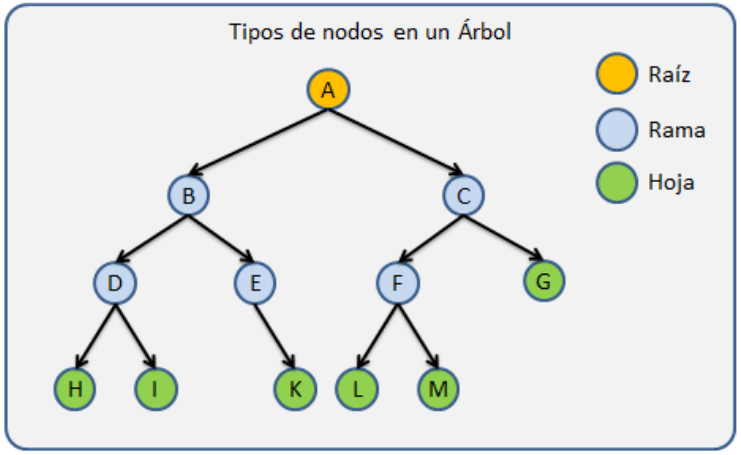
5. Árbol

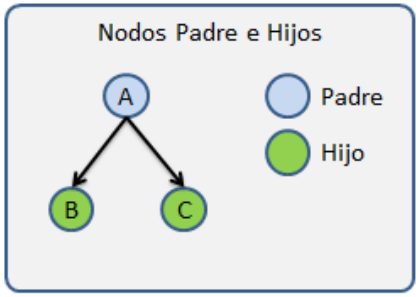
El Árbol es una estructura de datos donde sus nodos se almacenan de forma jerárquica y no lineal. No se va a profundizar en esta estructura en el presente módulo.

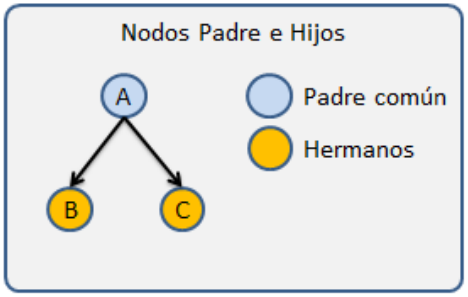


V.S.







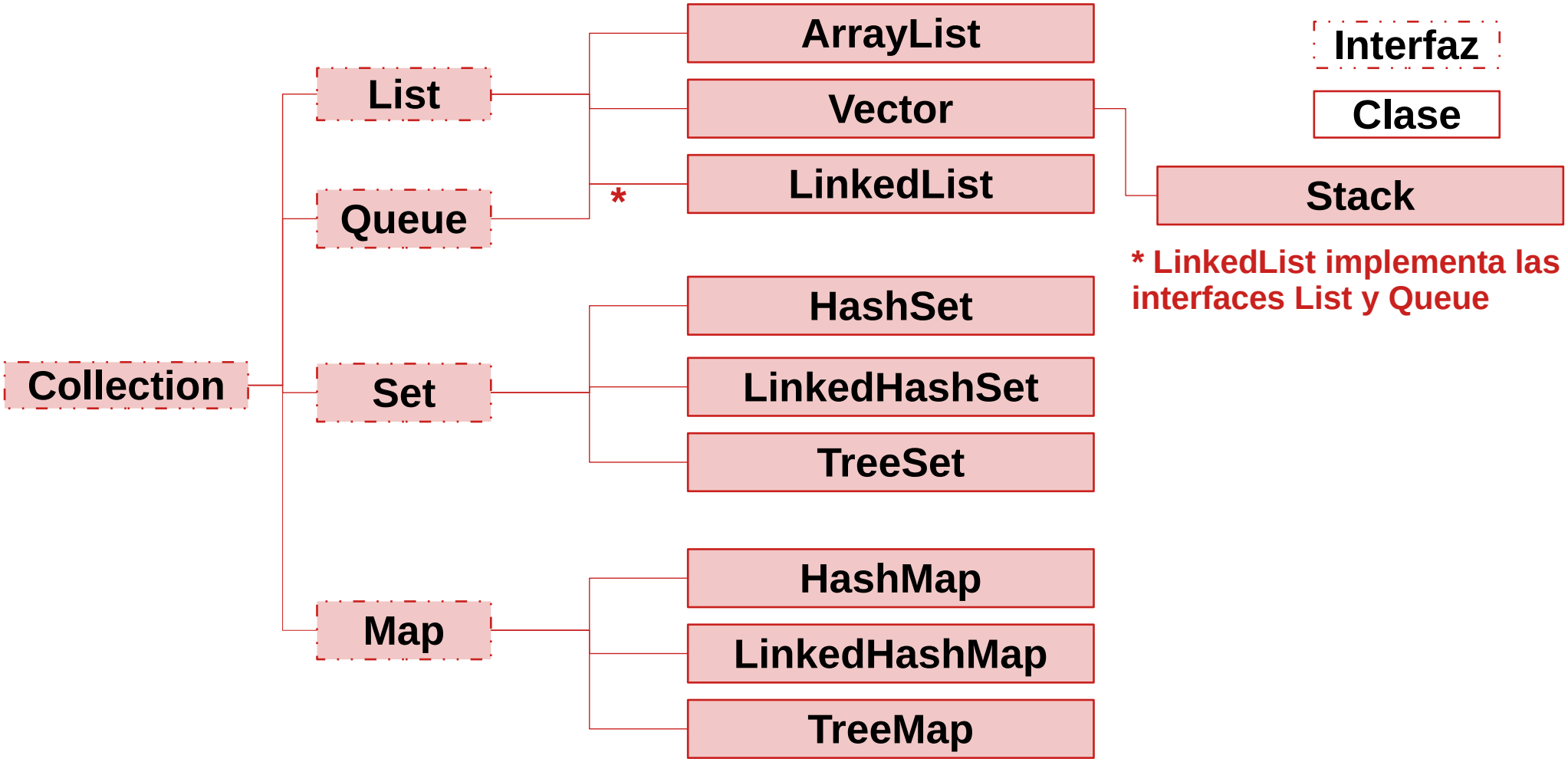


6. Colecciones

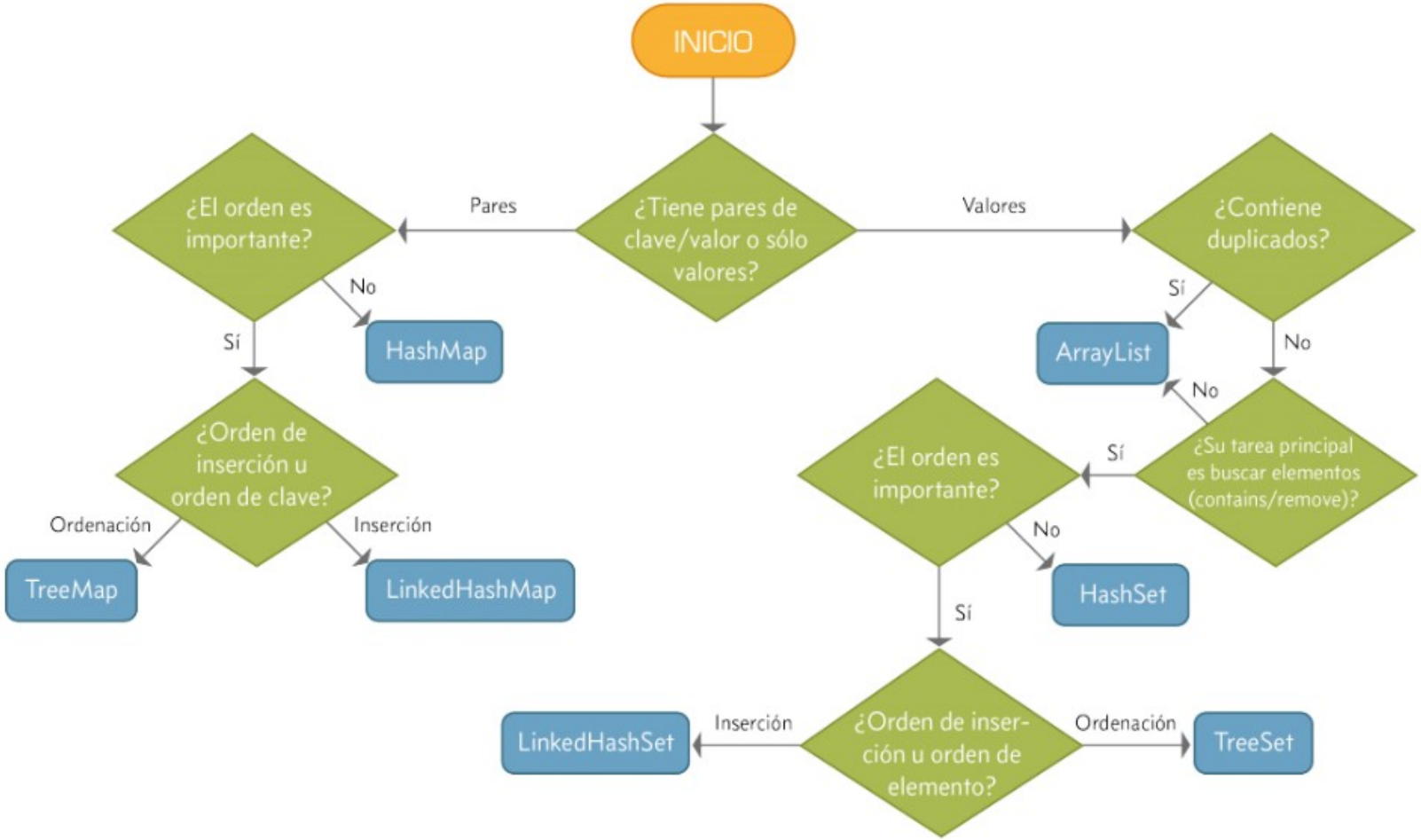
Java usa la interfaz genérica **Collection** para gestionar estructuras dinámicas. Así, se puede almacenar en ellas cualquier tipo de **objeto** (**no puede almacenar tipos simples**) y usar métodos muy utilizados como añadir, eliminar, obtener el tamaño, etc.

Collection<Tipo> nombre = new Collection<tipo>();

6. Colecciones



6. Colecciones.



6. Colecciones. ArrayList

Características:

- Es un Vector dinámico que crece o decrece.
- Es una colección ordenada tal y como se insertan los valores.
- Puede contener valores repetidos o duplicados y permite nulo (null).

Operaciones básicas:

ArrayList.**add**: Añade un elemento a la colección
ArrayList.**isEmpty**(): Comprueba si está vacía o no
ArrayList.**size**(): Devuelve el número de elementos
ArrayList.**contains**(Elemento): Indica si el elemento existe
ArrayList.**remove**(Elemento): Devuelve falso si el elemento no existe y lo elimina si existe
ArrayList.**clear**(): Elimina todos los elementos
ArrayList.**iterator**(): Permite la iteración de todos los elementos
Collections.**sort**(ArrayList): Ordenación ascendente de toda la lista
Collections.**sort**(ArrayList,Collections.**reverseOrder**()): Ordenación descendente de toda la lista

6. Colecciones. ArrayList

Ejemplo:

Crea un ArrayList para almacenar los días de la semana.

Crea otro ArrayList para almacenar los números del 1 al 10.

Itera sobre ellos para mostrarlos (usa Iterator).

Ordena de forma ascendente los días de la semana.

Ordena descendentemente los números (usa sort).

Muestra el resultado final iterando la estructura con foreach

6. Colecciones. ArrayList

Ejemplo almacenando los días de la semana y números del 1 al 10:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
public class EDArrayList{
    public static void main (String[] args){
        //Creamos ArrayList
        ArrayList<String> diasSemana = new ArrayList<String>();
        ArrayList<Integer> numeros = new ArrayList<Integer>();
        //Añadimos elementos al ArrayList
        diasSemana.add("Lunes");
        diasSemana.add("Martes");
        diasSemana.add("Miércoles");
        diasSemana.add("Jueves");
        diasSemana.add("Viernes");
        diasSemana.add("Sabado");
        diasSemana.add("Domingo");
        for (int i=1; i<=10;i++){
            numeros.add(i);
        }
        //Continúa en la siguiente hoja
```

6. Colecciones. ArrayList

```
        //Iteramos la estructura con Iterator
        System.out.println("Iteramos con Iterator");
        Iterator<String> ital = diasSemana.iterator();
        while (ital.hasNext()){
            System.out.println(ital.next()); }
        Iterator<Integer> ital1 = numeros.iterator();
        while (ital1.hasNext()){
            System.out.println(ital1.next());  }

        //Ordenamos las listas
        Collections.sort(diasSemana);
        Collections.sort(numeros,Collections.reverseOrder());

        //Iteramos la estructura con foreach
        System.out.println("Iteramos con foreach");
        for (String dia : diasSemana) {
            System.out.println(dia);  }

        for (Integer numero : numeros) {
            System.out.println(numero);  }
    }
}
```

6. Colecciones. LinkedList

Características:

- Es una colección ordenada por la inserción de elementos.
- Puede contener valores repetidos o duplicados.
- Permite ser usada como una lista, una pila o una cola

Operaciones básicas:

LinkedList.**add**: Añade un elemento a la colección (permite indicar posición).
También **addFirst** (al principio) y **addLast** (al final)
LinkedList.**isEmpty**(): Comprueba si está vacía o no
LinkedList.**size**(): Devuelve el número de elementos
LinkedList.**contains**(Elemento): Indica si el elemento existe
LinkedList.**remove**(Elemento): Devuelve el elemento y lo elimina. También **removeFirst** y **removeLast**
LinkedList.**get**(Elemento): Devuelve el elemento en la posición indicada. También **getFirst** (primer elemento) y **getLast** (último elemento)
LinkedList.**clear**(): Elimina todos los elementos
LinkedList.**iterator**(): Permite la iteración de todos los elementos
Collections.**sort**(LinkedList): Ordenación ascendente de toda la lista
Collections.**sort**(LinkedList.Collections,**reverseOrder**()): Ordenación descendente

6. Colecciones. LinkedList

Ejemplo:

Crea una LinkedList para almacenar los días de la semana.

Crea otra LinkedList para almacenar los números del 1 al 10.

Itera para mostrarlos (usa Iterator).

Muestra el primer día de la semana, el día de en medio y el último día de la semana (usa get).

Ordena de forma ascendente los días de la semana.

Ordena descendentemente los números (usa sort).

Muestra el resultado final iterando la estructura con foreach

6. Colecciones. LinkedList

```
import java.util.LinkedList;
import java.util.Collections;
import java.util.Iterator;
public class EDLinkedList{
    public static void main (String[] args){
        //Creamos LinkedList
        LinkedList<String> diasSemana = new LinkedList<String>();
        LinkedList<Integer> numeros = new LinkedList<Integer>();
        //Añadimos elementos al ArrayList
        diasSemana.add("Lunes"); diasSemana.add("Martes");
        diasSemana.add("Miércoles"); diasSemana.add("Jueves");
        diasSemana.add("Viernes"); diasSemana.add("Sabado");
        diasSemana.add("Domingo");
        for (int i=1; i<=10;i++){ numeros.add(i); }
        //Iteramos la estructura con Iterator
        System.out.println("Iteramos con Iterator");
        Iterator<String> ital = diasSemana.iterator();
        while (ital.hasNext()){
            System.out.println(ital.next()); }
        Iterator<Integer> ital1 = numeros.iterator();
        while (ital1.hasNext()){
            System.out.println(ital1.next());  }
        //Continúa en la siguiente hoja
```

6. Colecciones. LinkedList

```
        //Obtenemos el primer día, el último y el de en medio de la semana
        System.out.printf("El primer día de la semana es el %s\n",diasSemana.getFirst());
        System.out.printf("El día en la mitad de la semana es el %s\n",
diasSemana.get(diasSemana.size()/2));
        System.out.printf("El primer día de la semana es el %s\n",diasSemana.getLast());

        //Ordenamos las listas
        Collections.sort(diasSemana);
        Collections.sort(numeros,Collections.reverseOrder());

        //Iteramos la estructura con foreach
        System.out.println("Iteramos con foreach");
        for (String dia : diasSemana) {
            System.out.println(dia);  }

        for (Integer numero : numeros) {
            System.out.println(numero);  }
    }
}
```

6. Colecciones. Queue

Características:

- Las colas se usan en Java mediante una lista enlazada LinkedList (tiene las propiedades de ésta) usando la interfaz Queue.

Operaciones:

Queue.**add**: Añade un elemento al final de la cola

Queue.**offer**: Añade el elemento al final de la cola

Queue.**remove**(): Devuelve y elimina la cabeza o primer elemento de la cola

Queue.**poll**(Elemento): Devuelve y elimina la cabeza o primer elemento de la cola. Si la cola está vacía devuelve null

Queue.**element**(): Devuelve la cabeza o primer elemento de la cola

Queue.**peek**(): Devuelve la cabeza o frente de la cola. Si la cola está vacía devuelve null

6. Colecciones. Queue

Ejemplo:
Crea una Queue para almacenar los días de la semana.
Crea otra para almacenar los números del 1 al 10.
Itera sobre ellas para mostrarlos (usa Iterator con los días de la semana y foreach con los números).
Obtén e imprime el primer día de la semana y el primer número de la cola.
Desencola uno a uno los elementos de cada cola mostrando el elemento que se va a eliminar y el estado de la cola después de eliminarlo.

6. Colecciones. Queue

```
import java.util.Queue;
import java.util.LinkedList;
import java.util.Iterator;

public class EDQueue {
    public static void main(String[] args) {
        // Creamos Queue
        Queue<String> diasSemana = new LinkedList<String>();
        Queue<Integer> numeros = new LinkedList<Integer>();
        String dia;
        Integer numero;

        // Añadimos elementos a Queue
        diasSemana.add("Lunes");
        diasSemana.add("Martes");
        diasSemana.add("Miércoles");
        diasSemana.add("Jueves");
        diasSemana.add("Viernes");
        diasSemana.add("Sabado");
        diasSemana.add("Domingo");
        for (int i = 1; i <= 10; i++) {
            numeros.add(i);
        }

        //Continúa en la siguiente hoja
```

6. Colecciones. Queue

```
// Iteramos la estructura con Iterator
System.out.println("Iteramos con Iterator");
Iterator<String> itq = diasSemana.iterator();
while (itq.hasNext()) {
    System.out.println(itq.next());
}
System.out.println("Iteramos con Foreach");
//Iteramos con foreach
for (Integer num : numeros) {
    System.out.println(num);
}

// Obtenemos el primer elemento
dia = diasSemana.peek();
numero = numeros.peek();
System.out.printf("El primer día de la semana es el %s\n", dia);
System.out.printf("El primer número de la cola es el %s\n", numero);

 //Continúa en la siguiente hoja
```

6. Colecciones. Queue

```
        // Desencolamos los elementos uno a uno hasta dejar la cola vacía
        System.out.println("Desencolamos los elementos");
        while (dia!=null) {
             System.out.printf("Elimina el elemento %s de la cola\n", dia);
             diasSemana.poll();
             System.out.printf("La cola es: %s\n", diasSemana);
             dia = diasSemana.peek();
        }

        while (numero!=null) {
             System.out.printf("Se elimina el elemento %d\n", numero);
             numeros.poll();
             System.out.printf("La cola es: %s\n", numeros.toString());
             numero = numeros.peek();
        }
    }
}
```

6. Colecciones. Stack

Características:

- Es una colección que extiende la clase Vector para implementar la pila
- Puede usar funciones como add de Vector pero entonces no se comporta como una pila
- No permite acceso aleatorio a los elementos

Operaciones:

Stack.**isEmpty**(): Comprueba si está vacía o no

Stack.**push**(Elemento): Añade un elemento en la cima o tope de la pila

Stack.**pop**(): Devuelve y elimina el elemento en la cima o tope de la pila

Stack.**peek**(): Devuelve el elemento en la cima o tope de la pila sin eliminarlo. Si la pila es vacía devuelve una **excepción**

Stack.**search**(Elemento): Busca el elemento en la pila y devuelve la posición del mismo

6. Colecciones. Stack

Ejemplo:

Crea una Stack para almacenar los días de la semana.

Crea otra para almacenar los números del 1 al 10

Itera sobre ellas para mostrarlos (usa Iterator con los días de la semana y foreach con los números).

Busca el “Jueves” y el 5 en la pila y muestra su posición.

Obtén e imprime el elemento tope de cada pila.

Desapila uno a uno los elementos de cada pila mostrando el elemento que se va a eliminar y el estado de la pila.

6. Colecciones. Stack

```
import java.util.Stack;
import java.util.Iterator;

public class EDStack {
    public static void main(String[] args) {
        // Creamos Stack
        Stack<String> diasSemana = new Stack<String>();
        Stack<Integer> numeros = new Stack<Integer>();
        String dia;
        Integer numero;

        // Añadimos elementos a Stack
        diasSemana.push("Lunes");
        diasSemana.push("Martes");
        diasSemana.push("Miércoles");
        diasSemana.push("Jueves");
        diasSemana.push("Viernes");
        diasSemana.push("Sabado");
        diasSemana.push("Domingo");
        for (int i = 1; i <= 10; i++) {
            numeros.push(i);
        }

        //Continúa en la siguiente hoja
```

6. Colecciones. Stack

```
        // Iteramos
        System.out.println("Iteramos con Iterator");
        Iterator<String> its = diasSemana.iterator();
        while (its.hasNext()) {
            System.out.println(its.next());
        }
        System.out.println("Iteramos con Foreach");
        for (Integer num : numeros) {
            System.out.println(num);
        }

         // Buscamos un elemento
         dia = "Jueves";
         numero = 5;
         System.out.printf("El día de la semana %s está en la posición %d de la pila %s\n",
dia, diasSemana.search(dia),diasSemana);
         System.out.printf("El número %d está en la posición %d de la pila %s\n", numero,
numeros.search(numero),numeros.toString());

        // Obtenemos el primer elemento
        System.out.printf("El día de la semana en el tope de la pila es el %s\n",
diasSemana.peek());
        System.out.printf("El número en el tope de la pila es el %s\n", numeros.peek());
```

6. Colecciones. Stack

```
        // Desapilamos elementos hasta dejar la pila vacía
        System.out.println("Desapilamos los elementos hasta vaciar la pila");

        while (!diasSemana.isEmpty()) {
            System.out.printf("Se va a eliminar el elemento %s\n", dia);
            diasSemana.pop();
            System.out.printf("La pila es: %s\n", diasSemana);
            if (!diasSemana.isEmpty()){
                dia = diasSemana.peek();
            }
        }

        while (!numeros.isEmpty()) {
            System.out.printf("Se va a eliminar el elemento %d\n", numero);
            numeros.pop();
            System.out.printf("La pila es: %s\n", numeros.toString());
            if (!numeros.isEmpty()){
                numero = numeros.peek();
            }
        }
    }
}
```

6. Colecciones. HashSet

Características:

- Conjunto no ordenado.
- No mantiene la posición de los elementos insertados
- **No puede contener valores repetidos**
- Permite el valor nulo (null)
- No permite acceso aleatorio a los elementos

Operaciones básicas:

HashSet.**add**: Añade un elemento a la colección

HashSet.**isEmpty**(): Comprueba si está vacía o no

HashSet.**size**(): Devuelve el número de elementos de HashSet

HashSet.**contains**(Elemento): Si el elemento existe en el HashSet

HashSet.**remove**(Elemento): Devuelve falso si el elemento no existe y lo elimina

HashSet.**clear**(): Elimina todos los elementos de HashSet

6. Colecciones. HashSet

Ejemplo:

Crea un HashSet para almacenar los días de la semana.

Muestra su contenido.

Añade un elemento.

Comprueba que existe el elemento insertado.

Elimina el elemento.

Limpia el Set

6. Colecciones. HashSet

```
import java.util.HashSet;

public class EDHashSet {
    public static void main (String[] args){
        //Creamos HashSet para almacenar días de la semana
        HashSet<String> diasSemana = new HashSet<String>();
        String dia;

        //Añadimos elementos al HashSet
        diasSemana.add("Lunes");
        diasSemana.add("Martes");
        diasSemana.add("Miércoles");
        diasSemana.add("Jueves");
        diasSemana.add("Viernes");
        diasSemana.add("Sabado");
        diasSemana.add("Domingo");

        //Imprimimos el contenido de la colección si no es vacia
        if (!diasSemana.isEmpty()){
            System.out.println(diasSemana);
        }

         //Continúa en la siguiente hoja
```

6. Colecciones. HashSet

```
        //Añadimos un día inventado
        diasSemana.add("Juernes");

        //Comprobamos si existe el elemento
        dia="Juernes";
        if (diasSemana.contains(dia)){
            System.out.println("El día "+dia+" está en la colección "+diasSemana);
        } else {
            System.out.println("El día "+dia+" no está en la colección "+diasSemana);
        }

        //Eliminamos el elemento inventado
        if (diasSemana.remove(dia)){
            System.out.println("Tras eliminar el día "+dia+" la colección es "+diasSemana);
         }

        //Vaciamos la colección
        diasSemana.clear();
        System.out.println("Tras ejecutar clear la colección es "+diasSemana);
    }

}
```

6. Colecciones. LinkedHashSet

Características:

- Es una HashSet **ordenada**, mantiene la posición de los elementos que son insertados y permite iterar la colección.

Operaciones básicas:

LinkedHashSet.**add**: Añade un elemento a la colección

LinkedHashSet.**isEmpty**(): Comprueba si está vacía o no

LinkedHashSet.**size**(): Devuelve el número de elementos

LinkedHashSet.**contains**(Elemento): Indica si el elemento existe

LinkedHashSet.**remove**(Elemento): Devuelve falso si el elemento no existe y lo elimina si existe

LinkedHashSet.**clear**(): Elimina todos los elementos

LinkedHashSet.**iterator**(): Permite la iteración de todos los elementos

6. Colecciones. LinkedHashSet

Ejemplo:

Crea un LinkedHashSet para almacenar los días de la semana.

Comprueba que se mantienen en el orden creado.

Recorre la estructura con un iterador para mostrar los elementos.

Recorre la estructura con foreach.

6. Colecciones. LinkedHashSet

```
import java.util.LinkedHashSet;
import java.util.Iterator;

public class EDLinkedHashSet {
    public static void main (String[] args){
        //Creamos HashSet para almacenar meses del año
        LinkedHashSet<String> diasSemana = new LinkedHashSet<String>();

        //Añadimos elementos al LinkedHashSet
        diasSemana.add("Lunes");
        diasSemana.add("Martes");
        diasSemana.add("Miércoles");
        diasSemana.add("Jueves");
        diasSemana.add("Viernes");
        diasSemana.add("Sabado");
        diasSemana.add("Domingo");

        //Imprimimos el contenido de la colección si no es vacia
        if (!diasSemana.isEmpty()){
            System.out.println(diasSemana);
        }

        //Continúa en la siguiente hoja
```

6. Colecciones. LinkedHashSet

```
        //Iteramos la estructura con Iterator
        System.out.println("Iteramos con Iterator");
        Iterator<String> itlhs = diasSemana.iterator();
        while (itlhs.hasNext()){
            System.out.println(itlhs.next());
        }

        //Iteramos la estructura con foreach o for mejorado
        System.out.println("Iteramos con foreach");
        for (String dia : diasSemana) {
            System.out.println(dia);

        }

    }
}
```

6. Colecciones. TreeSet

Características:

- Es una colección ordenada en **orden ascendente**
- No puede contener valores repetidos.
- No permite el valor nulo (null)

Operaciones básicas:

TreeSet.**add**: Añade un elemento a la colección

TreeSet.**isEmpty**(): Comprueba si está vacía o no

TreeSet.**size**(): Devuelve el número de elementos

TreeSet.**contains**(Elemento): Indica si el elemento existe

TreeSet.**remove**(Elemento): Devuelve falso si el elemento no existe y lo elimina si existe

TreeSet.**clear**(): Elimina todos los elementos

TreeSet.**iterator**(): Permite la iteración de todos los elementos

6. Colecciones. TreeSet

Ejemplo:

Crea un TreeSet para almacenar los días de la semana.

Crea otro para almacenar números del 1 al 10. Los insertamos de manera inversa.

6. Colecciones. TreeSet

```
import java.util.TreeSet;
import java.util.Iterator;

public class EDTreeSet {
    public static void main (String[] args){
        //Creamos TreeSet para almacenar meses del año
        TreeSet<String> diasSemana = new TreeSet<String>();
        TreeSet<Integer> numeros = new TreeSet<Integer>();

        //Añadimos elementos al TreeSet
        diasSemana.add("Lunes");
        diasSemana.add("Martes");
        diasSemana.add("Miércoles");
        diasSemana.add("Jueves");
        diasSemana.add("Viernes");
        diasSemana.add("Sabado");
        diasSemana.add("Domingo");
        for (int i=10; i>=1;i--){
            numeros.add(i);
        }

        //Continúa en la siguiente hoja
```

6. Colecciones. TreeSet

```
        //Iteramos con Iterator
        System.out.println("Iteramos con Iterator");
        Iterator<String> itts = diasSemana.iterator();
        while (itts.hasNext()){
            System.out.println(itts.next());
        }
        Iterator<Integer> itt1s = numeros.iterator();
        while (itt1s.hasNext()){
            System.out.println(itt1s.next());
        }

        //Iteramos con foreach
        System.out.println("Iteramos con foreach");
        for (String dia : diasSemana) {
            System.out.println(dia);
        }
        for (Integer numero : numeros) {
            System.out.println(numero);
        }
    }
}
```

6. Colecciones. HashMap

Características:

- Es una colección que permite almacenar pares clave(K):valor(V)
- Se indican como HashMap<K,V>
- Las claves deben ser únicas
- Permite el valor nulo (null) en los valores y la clave null
- No es ordenada y no es iterable. Usar entrySet para recorrerla.

Operaciones básicas:

HashMap.**put**: Añade o modifica un elemento a la colección.
HashMap.**get**(clave):Devuelve el valor asociado a esa clave
HashMap.**isEmpty**(): Comprueba si está vacía o no
HashMap.**size**(): Devuelve el número de elementos de HashMap
HashMap.**containsKey**(clave): Si la clave existe en el HashMap
HashMap.**containsValue**(valor): Si el valor existe en el HashMap
HashMap.**remove**(clave): Lo elimina. También **remove**(clave,valor)
HashMap.**clear**(): Elimina todos los elementos de HashMap
HashMap.**keySet**(): Devuelve el conjunto de claves del HashMap

6. Colecciones. HashMap

Ejemplo:
Crea un HashMap para almacenar los días de la semana (clave Integer y valor String).
Crea otro para almacenar los números del 1 al 10 (clave String y valor Integer).
Muestra el tamaño de ambos HashMap.
Comprobamos algunas claves y algunos valores
Buscamos elementos.
Reemplaza un valor.
Elimina un elemento por clave y por valor.
Recorre las claves de los HashMap con foreach imprimiendo clave-valor de todos sus elementos.

6. Colecciones. HashMap

```
import java.util.HashMap;

public class EDHashMap {
    public static void main(String[] args) {
        // Creamos HashMap
        HashMap<Integer, String> diasSemana = new HashMap<Integer, String>();
        HashMap<String, Integer> numeros = new HashMap<String, Integer>();
        String ord, valor;
        int index;
        // Añadimos elementos a HashMap
        diasSemana.put(1, "Lunes");
        diasSemana.put(2, "Martes");
        diasSemana.put(3, "Miércoles");
        diasSemana.put(4, "Jueves");
        diasSemana.put(5, "Viernes");
        diasSemana.put(6, "Sabado");
        diasSemana.put(7, "Domingo");
        numeros.put("primero", 1);
        numeros.put("segundo", 2);
        numeros.put("tercero", 3);
        numeros.put("cuarto", 4);
        numeros.put("quinto", 5);
        numeros.put("sexto", 6);
        numeros.put("séptimo", 7);
        numeros.put("octavo", 8);
        numeros.put("noveno", 9);
        numeros.put("décimo", 10);
```

6. Colecciones. HashMap

```
// Mostramos el tamaño de los HashMap
System.out.printf("El HashMap diasSemana contiene %d elementos\n", diasSemana.size());
System.out.printf("El HashMap numeros contiene %d elementos\n", numeros.size());
 // Comprobamos existen o no elementos
index = 7;
if (diasSemana.containsKey(index)) {
    System.out.printf("El índice %d es %s\n", index, diasSemana.get(index));
}
ord = "octavo";
if (numeros.containsKey(ord)) {
    System.out.printf("El índice %s tiene el valor %d\n", ord, numeros.get(ord));
}
 // Buscamos un elemento
index = 4;
ord = "quinto";
System.out.printf("El %s tiene la clave %d \n", diasSemana.get(index), index);
System.out.printf("La clave %s contiene el valor %d\n", ord, numeros.get(ord));
 // Reemplazamos un elemento (Java 8)
valor = "Juernes";
index = 4; // corresponde al Jueves
System.out.printf("El %s con clave %d se va a reemplazar por %s\n",
                    diasSemana.get(index), index, valor);
diasSemana.replace(index, diasSemana.get(index), valor);
System.out.printf("El día %d ahora es %s\n", index, diasSemana.get(index));

 //Continúa en la siguiente hoja
```

6. Colecciones. HashMap

```
        // Eliminamos elementos de ambos HashMap
        index = 4; // corresponde al Jueves
        ord = "cuarto";
        System.out.printf("El %s con clave %d se elimina\n", diasSemana.get(index), index);
        diasSemana.remove(index);
        System.out.printf("El %d con clave %s se elimina\n", numeros.get(ord), ord);
        numeros.remove(ord);

        // Tratamos de recorrerla

        System.out.println("Mostramos los elementos de dias de la semana: ");
        for (Integer ds : diasSemana.keySet()) {
            System.out.printf("La clave %d contiene el día %s\n", ds, diasSemana.get(ds));
        }

        System.out.println("Mostramos los elementos de números: ");
        for (String num : numeros.keySet()) {
            System.out.printf("La clave %s contiene el número %d\n", num, numeros.get(num));
        }
    }
}
```

6. Colecciones. LinkedHashMap

Características:

- Es una colección que permite almacenar pares clave(K):valor(V)
- La diferencia con HashMap es que utiliza una lista enlazada

Operaciones básicas:

LinkedHashMap.**put**: Añade o modifica un elemento a la colección.

LinkedHashMap.**get**(clave):Devuelve el valor asociado a esa clave

LinkedHashMap.**isEmpty**(): Comprueba si está vacía o no

LinkedHashMap.**size**(): Devuelve el número de elementos de HashMap

LinkedHashMap.**containsKey**(clave): Si la clave existe en el HashMap

LinkedHashMap.**containsValue**(valor): Si el valor existe en el HashMap

LinkedHashMap.**remove**(clave): Lo elimina. También **remove**(clave,valor)

LinkedHashMap.**clear**(): Elimina todos los elementos de HashMap

LinkedHashMap.**keySet**(): Devuelve el conjunto de claves del HashMap

6. Colecciones. TreeMap

Características:

- Es una colección pares clave(K):valor(V) ordenada por clave
- Las claves deben ser únicas
- Permite el valor nulo (null) en los valores y la clave null.

Operaciones básicas:

TreeMap.**put**: Añade o modifica un elemento a la colección.
TreeMap.**get**(clave):Devuelve el valor asociado a esa clave
TreeMap.**isEmpty**(): Comprueba si está vacía o no
TreeMap.**size**(): Devuelve el número de elementos de TreeMap
TreeMap.**containsKey**(clave): Si la clave existe en el TreeMap
TreeMap.**containsValue**(valor): Si el valor existe en el TreeMap
TreeMap.**remove**(clave): Lo elimina. También **remove**(clave,valor)
TreeMap.**clear**(): Elimina todos los elementos de TreeMap
TreeMap.**FirstKey**(): Devuelve la primera clave.
TreeMap.**LastKey**(): Devuelve la última clave.

6. Colecciones. TreeMap

Ejemplo:
Crea un TreeMap para almacenar los días de la semana (clave Integer y valor String).
Muestra el tamaño del TreeMap.
Comprobamos algunas claves y algunos valores
Buscamos elementos.
Reemplaza un valor.
Elimina un elemento por clave y por valor.
Recorre las claves de los HashMap con foreach imprimiendo clave-valor de todos sus elementos.

6. Colecciones. TreeMap

```
import java.util.TreeMap;
import java.util.NavigableMap;

public class EDTreeMap {
    public static void main(String[] args) {
        // Creamos TreeMap
        TreeMap<Integer, String> diasSemana = new TreeMap<Integer, String>();
        NavigableMap<Integer, String> diasSemanaSDesc = new TreeMap<Integer, String>();
        String dia;
        int index;
        // Añadimos elementos a TreeMap
        diasSemana.put(1, "Lunes");
        diasSemana.put(2, "Martes");
        diasSemana.put(3, "Miércoles");
        diasSemana.put(4, "Jueves");
        diasSemana.put(5, "Viernes");
        diasSemana.put(6, "Sabado");
        diasSemana.put(7, "Domingo");
        // Mostramos el tamaño de los TreeMap
        System.out.printf("El TreeMap contiene %d elementos\n", diasSemana.size());
        System.out.println(diasSemana);
        //Mostramos la primera clave y la última del TreeMap
        System.out.printf("La clave mínima es %d\n",diasSemana.firstKey());
        System.out.printf("La clave máxima es %d\n",diasSemana.lastKey());

        //Continúa en la siguiente hoja
```

6. Colecciones. TreeMap

```
 //Mostramos subMapa de TreeMap
System.out.println("El submapa entre 3 y 5 es "+diasSemana.subMap(3,true, 5,true));

//ordenamos el TreeMap de forma descendente
System.out.println("antes de ordenar "+diasSemana);
System.out.println("después de ordenar "+diasSemana.descendingMap());

//Comprobamos existen o no elementos
index = 7;
if (diasSemana.containsKey(index)) {
    System.out.printf("El índice %d existe y tiene el valor %s\n", index,
            diasSemana.get(index));
}

 // Buscamos un elemento
dia = "Jueves";
if (diasSemana.containsValue(dia)){
    for (Integer ds : diasSemana.keySet()) {
        if (diasSemana.get(ds)==dia){
            System.out.printf("El %s es el valor de la clave %d \n", dia, ds);
        }
    }
}
 //Continúa en la siguiente hoja
```

6. Colecciones. TreeMap

```
        // Reemplazamos un elemento (Java 8)
        dia = "Juernes";
        index = 4; // corresponde al Jueves
        System.out.printf("El %s con clave %d se va a reemplazar por %s\n",
                          diasSemana.get(index), index, dia);
        diasSemana.replace(index, diasSemana.get(index), dia);
        System.out.printf("El día %d ahora es %s\n", index, diasSemana.get(index));

        // Eliminamos un elemento del TreeMap
        index = 4;
        System.out.printf("El %s con clave %d se elimina\n", diasSemana.get(index), index);
        diasSemana.remove(index);

        // Tratamos de recorrerla
        System.out.println("Mostramos los elementos de dias de la semana: ");
        for (Integer ds : diasSemana.keySet()) {
            System.out.printf("La clave %d contiene el día %s\n", ds, diasSemana.get(ds));
        }
    }
}
```