UD13- Estructuras dinámicas de datos

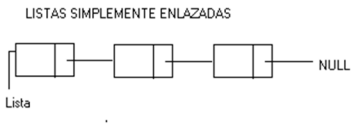
1. Introducción a las estructuras lineales.
2. Colecciones en Java.
   A. Listas.
   B. Colas.
   C. Pilas.
   D. Conjuntos.
3. Clase Collections.
4. Diccionarios.

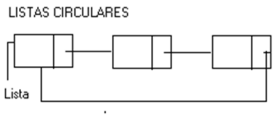
PROG. Nuria Fuentes Pérez

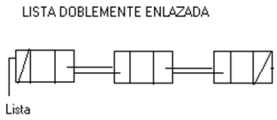
CENTRO FP SUPERIOR
CAMPUS CÁMARA COMERCIO SEVILLA

1. ESTRUCTURAS DINÁMICAS LINEALES.

- **Listas enlazadas.**

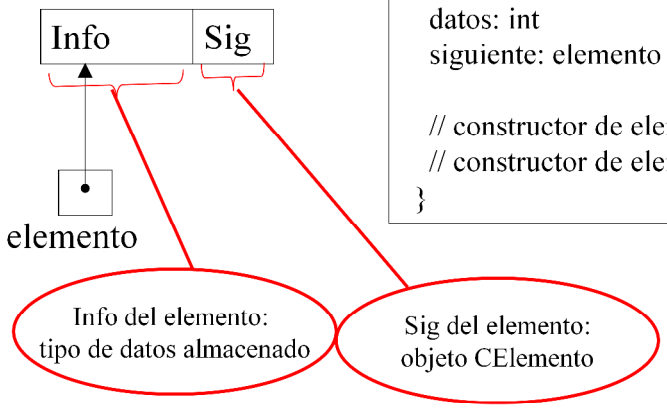






1. ESTRUCTURAS DINÁMICAS. Listas. Diseño

CElemento elemento;

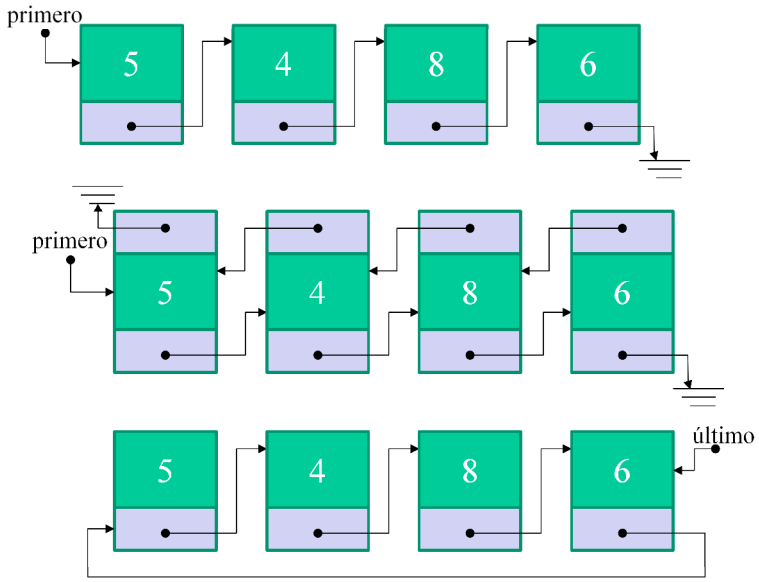



```
private class CElemento{

   // Atributos
   datos: int
   siguiente: elemento siguiente

   // constructor de elemento por defecto
   // constructor de elemento(datos, siguiente)
 }
```

1. ESTRUCTURAS DINÁMICAS. Listas. Diseño



1. ESTRUCTURAS DINÁMICAS. Listas. Diseño

CListaLinealSE

```
public class CListaLinealSE{
 // primero: referencia al primer elemento de la lista. Es el elemento de cabecera.
  //Constructor : de la lista por defecto
 // Métodos
 longitud() →   Devuelve el número de elementos de la lista
 aniadir(i, int) devuelve verdadero o falso →  Añadir un elemento en la posición i
 aniadirAlPrincipio(int) devuelve verdadero o falso →  Añadir un elemento al principio
 aniadirAlFinal(obj) devuelve verdadero o falso → Añadir un elemento al final
 borrar(i) devuelve los datos del elemento eliminado → Borrar el elemento de la posición i
 borrarPrimero() devuelve los datos del elemento eliminado → Borrar el primer elemento
 borrarUltimo() devuelve los datos del elemento eliminado → Borrar el último elemento
```

1. ESTRUCTURAS DINÁMICAS. Listas. Diseño

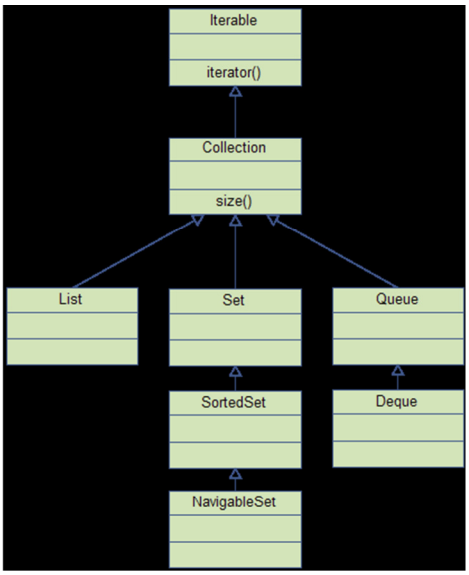
CListaLinealSE

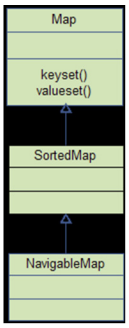
get(i) devuelve los datos del elemento → Obtener el elemento de la posición i

getPrimero() devuelve los datos del elemento → Retornar el primer elemento

getUltimo() devuelve los datos del elemento → Retornar el último elemento

2. COLECCIONES EN JAVA.





java.util

Existen una serie de **convenciones para nombrar a los genéricos**:

E – Element (usado bastante por Java Collections Framework)

K – Key (Llave, usado en mapas)

N – Number (para números)

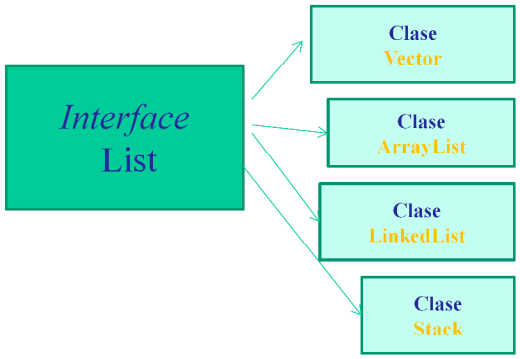
T – Type (Representa un tipo, es decir, una clase)

V – Value (representa el valor, también se usa en mapas)

S,U,V etc. – usado para representar otros tipos.

2.A. COLECCIONES EN JAVA. Listas (List)

Representan una lista ordenada de objetos, es decir, que los elementos son accesibles en un orden específico o mediante un índice (*index).* En una lista se puede agregar el mismo elemento más de una vez.




```
List<T> listaA = new ArrayList();
List<T> listaB = new LinkedList();
List<T> listaC = new Vector();
List<T> listaD = new Stack();
```

¿Diferencias entre Vector, ArrayList, LinkedList?

2.A. COLECCIONES EN JAVA. Listas (List)

Ej. Recorrer una lista

OJO

```
//Entorno:
    List<MiObjeto> lista;
//Algoritmo:
    lista = new ArrayList<>();
```

ITERADOR

```
//Entorno:
   MiObjeto obj;
   Iterator iterador;
//Algoritmo:
   //Iterador
   iterador = lista.iterator();
   while (iterador.hasNext()) {
      obj = iterador.next();
      //usar obj
   }//Fin Mientras
```

FOREACH

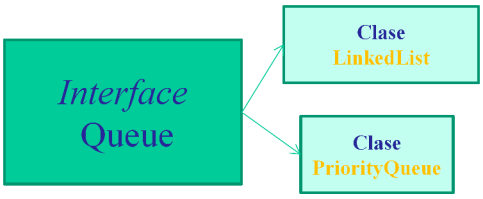
```
for(MiObjeto obj: lista){
   // usar obj
}//Fin Para
```

FOR

```
int i;
for(i=0;i<lista.size();i++){
   // usar lista.get(i)
}//Fin Para
```

2.B. COLECCIONES EN JAVA. Colas (Queue)

Representa una lista ordenada de objetos, como List, pero una cola está diseñada para tener sus elementos insertados al final de la cola y eliminados del principio. FIFO ("First in, First out")



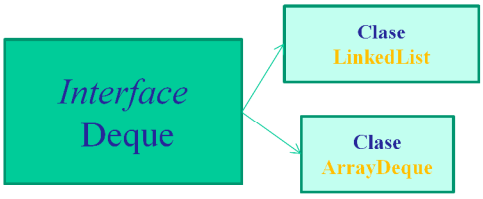

```
Queue colaA = new LinkedList();
Queue colaB = new PriorityQueue();
```

```
Queue<MiObjecto> cola = new LinkedList<MiObjecto>();
```

método	descripción
offer()	**Encolar**. Añade el objeto al final de la cola.
peek()	devuelve el objeto de la cabecera de la cola sin sacarlo
poll()	**Desencolar**. Devuelve el objeto de la cabecera sacándolo de la cola.

2.B. COLECCIONES EN JAVA. Colas doble entrada (Deque)

Representa un tipo de cola en la cual se pueden insertar y eliminar elementos de ambos lados.




```
Deque dequeA = new LinkedList();
Deque dequeB = new ArrayDeque();
```

```
Deque<MiObjecto> deque = new LinkedList<MiObjecto>();
```

2.C. COLECCIONES EN JAVA. Pilas (Stack)

Es una estructura de datos en la que se agregan elementos al "tope" del stack (cima de la pila), y también se quitan elementos del "tope". LIFO *(Last in, First out)*.

Clase
Stack

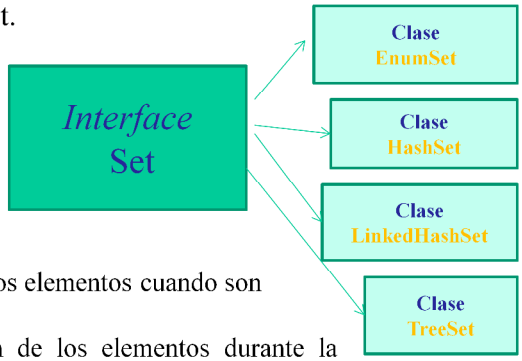
```
Stack pila = new Stack();
```

método	descripción
pop()	devuelve el objeto de la cima de la pila, eliminándolo
peek()	devuelve el objeto de la cima de la pila pero sin sacarlo
push()	coloca un objeto en la cima de la pila.

2.D. COLECCIONES EN JAVA. Conjuntos (Set)

Representa un conjunto de objetos, lo que significa que cada elemento puede existir solamente una vez en el Set.

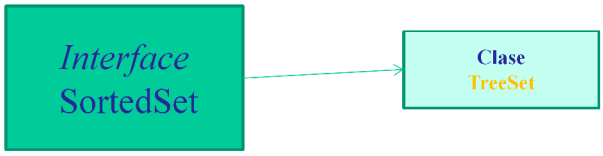
```
Set setA = new EnumSet();
Set setB = new HashSet();
Set setC = new LinkedHashSet();
Set setD = new TreeSet();
```



HashSet. No garantiza la secuencia de los elementos cuando son iterados.
LinkedHashSet. Sí garantiza el orden de los elementos durante la iteración, orden de inserción. Reinsertar un elemento que ya está no cambia su orden.
TreeSet. Garantiza el orden de los elementos al iterarlos, pero el orden en el que se almacenarían si se utilizara el método Collections.sort() en una lista o arreglo que contenga dichos elementos.

2.D. COLECCIONES EN JAVA. Conjuntos (Set)

Representa un conjunto de objetos, lo que significa que cada elemento puede existir solamente una vez en el Set.



3. CLASE COLLECTIONS

Contiene métodos estáticos que operan sobre colecciones.

- void copy(List, List)
- boolean disjoint(Collection, Collection)
- Object max(Collection)
- void reverse(List)
- void sort(List)

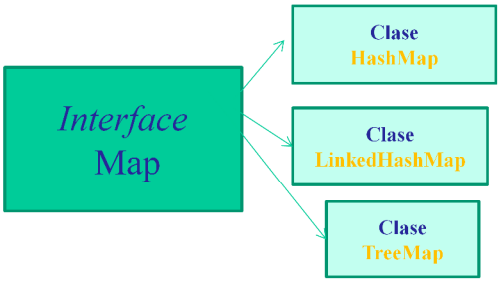
Interface Comparable<T>

Interface Comparator<T>

4. DICCIONARIOS. (Map)

Representa un conjunto de pares <clave,valor> <K,V> de tal manera que para una clave solamente tenemos un valor.

```
Map<Integer,String>  mapa1
   = new HashMap<Integer,String>();
Map<Integer,String>  mapa2
   = new TreeMap<Integer,String>();
Map<Integer,String>  mapa3
  = new LinkedHashMap<Integer,String>();
```



HashMap. No garantiza la secuencia de los elementos cuando son iterados. No acepta claves duplicadas ni nulas.
LinkedHashMap. Mantienen el orden de inserción de los elementos. (más lenta)
TreeMap. Ordena los elementos de forma "natural". Por ejemplo, si la clave son valores enteros (como luego veremos), los ordena de menor a mayor.