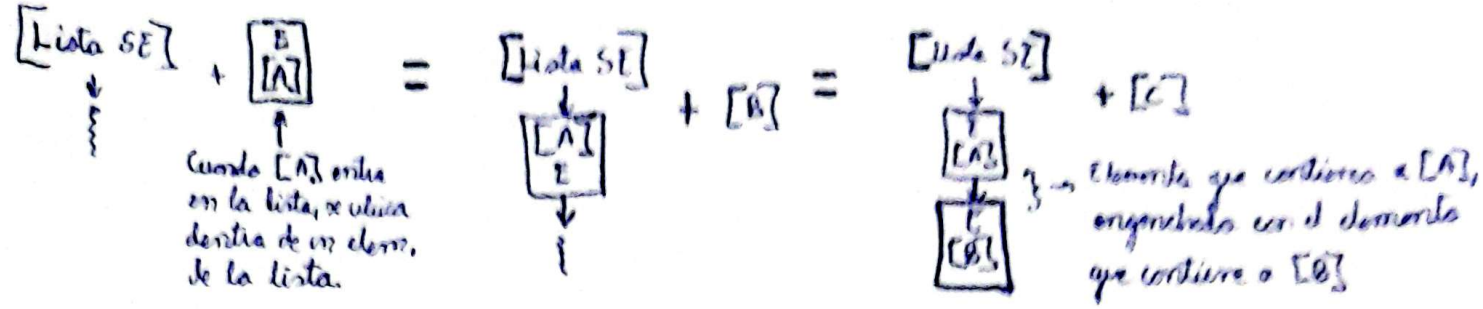
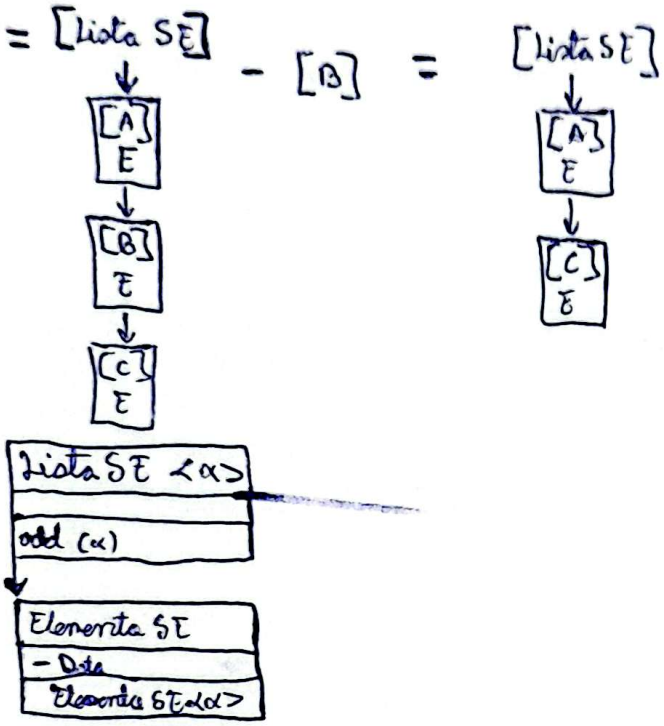
Tipos de Listas

→ Listas Dinámicos { - SE;
- DE;



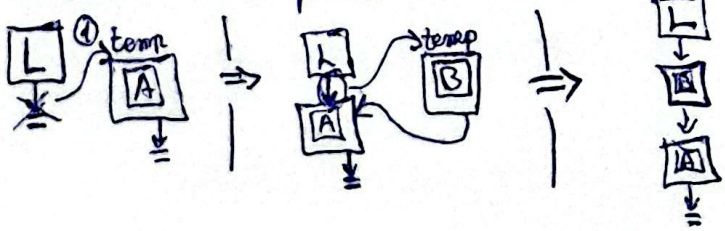


Programa de Ejemplo

```
Lista SE <integer> mi lista = new Lista SE <integer>
.
.
.
void mi Prueba (){
    integer a = new Integer (5);
    mi_lista. add(a);
}
```

(Primera Manera)

```
int add (α dato){
    Elemento SE <α> temporal = new Elemento SE <α> (dato)
① temporal. set Siguiente ( this.primerelemento)
    set Primer Elemento (temporal)
}
```

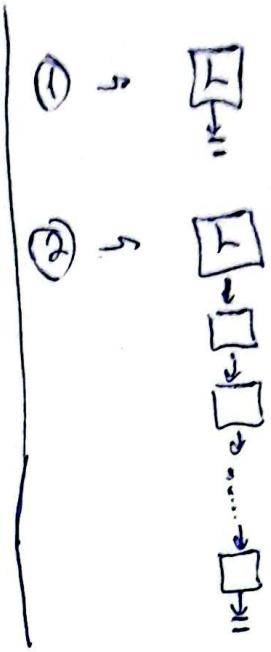


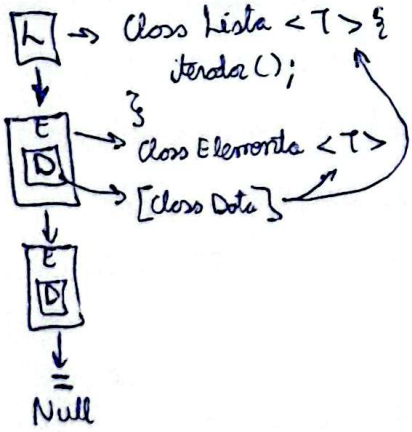
(Codigo de Elemento SE)

```
Class Elemento SE <α>{
  α data
  Elemento SE <α> siguiente;
  Elemento SE (α data) {
    this data = midato
  }
} private Elemento SE (){}.
```

(Segunda Manera)

```
int add (α data) {
    Elemento SE <α> temporal = nuevo Elemento SE <α> (data);
①   if ( getPrimerElemento () == null)
        setPrimerElemento (temporal);
②   else
        Elemento SE <α> flotante = getPrimerElemento ()
        while ( flotante.getSiguiente () ≠ null ) {
            flotante = flotante.getSiguiente ();
        }
        flotante.setSiguiente (temporal);
    }
```

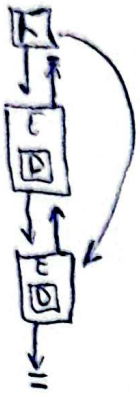




[Class Iterador] → sirve para guardar el estado de una operación de recorrido de una lista

```
Class Iterador <T> implements IIterador <T> {
    Lista <T> miLista;
    Elemento <T> actual;
    Iterador ( lista <T> l) {
        this.miLista = l
        this.actual = l.primerElemento
    }
    boolean haySiguiente () {
        return this.actual != null;
    }
    T getData () {
        T temporal = this.actual.getData ();
        this.actual = this.actual.getSiguiente ();
        return temporal;
    }
```

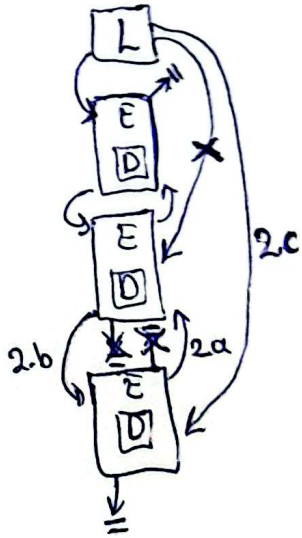
Lista Doblemente Enlazada




```
Class Lista DE {
    elemento DE primero
    elemento DE ultimo
}
Class ElementoDE {
    T.data;
    ElementoDE anterior;
    ElementoDE siguiente;
}
```

ADD (Protegido)
Insert ≠
Delete ≠

```
Void add (TD data){
    Elemento <TD> elemento = new ElementoDE(data);
    if (this.primero = null) {          → ① Lista Vacía
        this.primero = elemento;
        this.ultimo = elemento;
    }
    else {
    2.a elemento.anterior = this.ultimo;
    2.b this.ultimo.siguiente = elemento;   → ② Lista con Elementos
    2.c this.ultimo = elemento;
    }
}
```



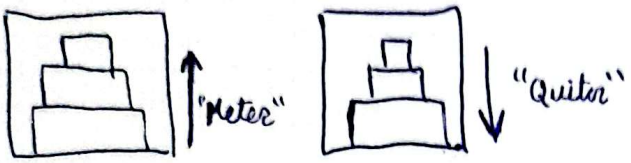

```
Void Insert (ElementoDE elemento, TD data){
① → Vacía   if (elemento == null)
                this.add(data);
            else
                ElementoDE <TD> nuevo Elemento = new ElementoDE(data);
                if (elemento = this.primero){
                    elemento.anterior = nuevo Elemento;
                    nuevo Elemento.siguiente = elemento;
                    this.primero = nuevo Elemento;
                } ⊛
```

```
        else{
            nuevo Elemento Siguiente = elemento;
            nuevo Elemento Anterior = elemento.anterior;
            nuevo Elemento Anterior Siguiente = nuevo Elemento;
            nuevo Elemento Siguiente Anterior = nuevo Elemento;
        }
    }

Void delete (Elemento DE elemento){
    if (elemento != null){
    -> boolean primero = false, ultimo = false;
    if (elemento.anterior == null){
        this.primero = elemento.siguiente;
        primero = true;
    }
    if (elemento.siguiente == null){
        this.ultimo = elemento.anterior;
        ultimo = true;
    }
    if (!primero)
        elemento.anterior.siguiente = elemento.siguiente;
    if (!ultimo)
        elemento.siguiente.anterior = elemento.anterior;
```

Pilas y Colas

[Pila] (LIFO)



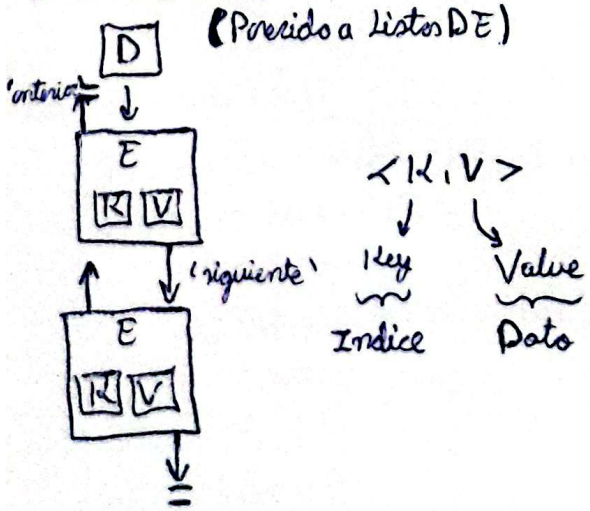
[Cola] (FIFO)

- Empuevue (Agrega al final)
- Depuevue (Devuelve el primer dem y la elim.)

```
Class PilaLIFO<T>{
    ListaDE<T> pila = new ListaDE();
    void push(T dato){ lista.add(dato)}
    T pop(){
        T temporal = pila.ultimo.getDato()
        pila.delete(pila ultima);
    }
}
```

Diccionarios

(Parecido a ListasDE)



Ej

Diccionario
Elemento D <K,V> Primero Elemento D<K,V> Ultimo
boolean add(K,V) boolean insert(K,V) boolean delete(K) Lista <K> getKeys() Lista <V> getValues() boolean exists(K) V get Value(K)

Elemento D
Elemento <K,V> Anterior Elemento <K,V> Siguiente K indice V dato
boolean delete() K get Key() V get Value() boolean Update Value(V)

Iterador D
Diccionario <K,V> mi Diccionario Elemento D<K,V> actual
boolean hasNext() V next() K getKey() ← No acepta !!! V getValue() ← No acepta !!! #get Actual()

Iterador get Iterador()

Iterador D find(K)

Iterador D find (IteradorD, K)

```
Lista <K> getKeys(){
    Iterador <K,V> it.this.get Iterador()
    Lista<K> claves = new Lista <K>();
    while (it. hasNext()){
        it. next();
        claves.add(it.getKey());
    }
    return claves
}
```

```
                (clave)
boolean exists (K){
    Iterador <K,V> it = this.get Iterador();
    while (it. hasNext()){
        it.next();
        if (it.getKey() == clave)
            return true;
    }
    return false
}
```

```
V get Value (Kclave){
    Iterador <K,V> it = new IteradorD()
    while (it. hasNext()){
        it.Next()
        if(it.getKey() == clave)
            return it.getValue();
    }
    return null
}
```

```
# IteradorD<K,V> find (IteradorD<K,V> it, Kclave
    while (it.hasNext()){
        it.Next();
        if(it.getKey() == clave)
            return it
    }
    return it (devolver un iterador)
}
```

(Otra forma, pero no repetir código)

```
# IIterador D<K,V> find (K clave) {
    IIterador D<K,V> it = nuevo . IIterador D();
    return (this. find (it, clave));
}
boolean existe (K clave){
    return this find (clave) hasNext();
}
V get Value (K clave) {
    return this. find (clave) get Value();
}
boolean insert (K clave, V data){
    IIterador D<K,V> it = this find (clave)
    if(it getActual () != null)
        it. getActual() set Value (data);
    else
        this. add (clave, data);
```