Listas Ordenadas

Programación 1

TDAs - Listas Ordenadas

La Lista Ordenada debe *aprender* a ordenar su contenido comparando las *claves* de sus elementos

Para eso utilizará la implementación de los métodos *compare*(...) y *compareByKey*(...) a implementar en cada lista.

- `public int compare(T elemento1, T elemento2)`

  Se utiliza para comparar dos elementos entre sí. Es utilizado internamente durante la inserción de nuevos elementos.

- `public int compareByKey(K clave, T elemento)`

  Se utiliza para comparar el valor de una clave con un valor determinado. Es utilizado internamente para la búsqueda por clave de un elemento en el método *seach(,,,)*.

En ambos casos el valor numérico devuelto determina el "peso" de la clave y el posterior ordenamiento de los elementos:

- **Si el valor retornado es 0 (cero) se asume que los elementos son iguales o equivalentes.**
- **Si el valor devuelto es mayor a cero, significa que la clave del primer elemento es mayor que la clave del segundo (en el *compare*), o que el valor recibido como clave de búsqueda es mayor que la clave del elemento (en el *compareByKey*).**
- **Si el valor devuelto es menor que cero, significa que la clave del primero es menor a la del segundo (*compare*), o que la clave de búsqueda recibida es mayor a la clave del elemento (*compareByKey*).**

El siguiente caso muestra la implementación de estos métodos con claves *comparables* (por ejemplo Strings):

```
public class OrdenadaPorString extends ListaOrdenada<String, Elemento> {

    @Override
    public int compare(Elemento elemento1, Elemento elemento2) {
        return elemento1.getClave().compareTo(elemento2.getClave());
    }

    @Override
    public int compareByKey(String clave, Elemento elemento) {
        return clave.compareTo(elemento.getClave());
    }

}
```

El siguiente caso muestra la implementación de estos métodos con claves *numéricas*:

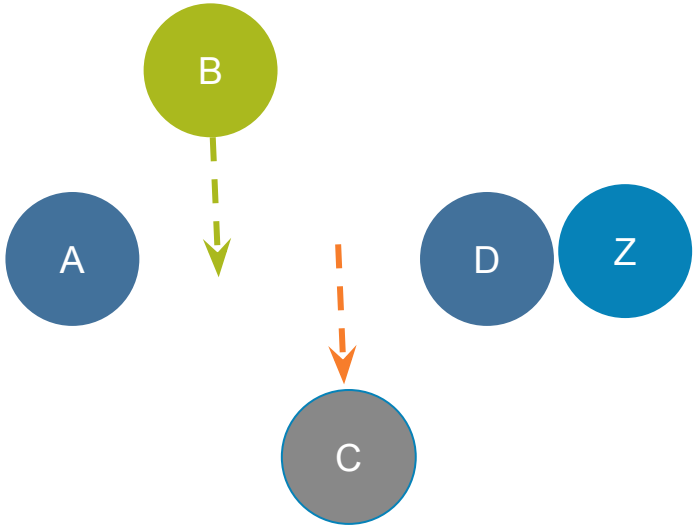
```
public class OrdenadaPorEntero extends ListaOrdenada<Integer, Elemento{

     @Override
     public int compare(Elemento elemento1, Elemento elemento2) {
          return elemento1.getNroClave() - elemento2.getNroClave();
     }

     @Override
     public int compareByKey(Integer clave, Elemento elemento) {
          return clave - elemento.getNroClave();
     }

}
```

Lista Ordenada (acceso random)



- Agrega en una posición dada según un orden definido para la lista (ascendente por nombre, descendente por edad, etc.).
- Se puede acceder o extraer cualquier elemento contenido.
- Un nodo se puede ubicar por posición o por *clave*.

get(int pos)

Permite obtener el elemento contenido en la posición requerida.

```
elem = lista.get(2)
```

search(K clave)

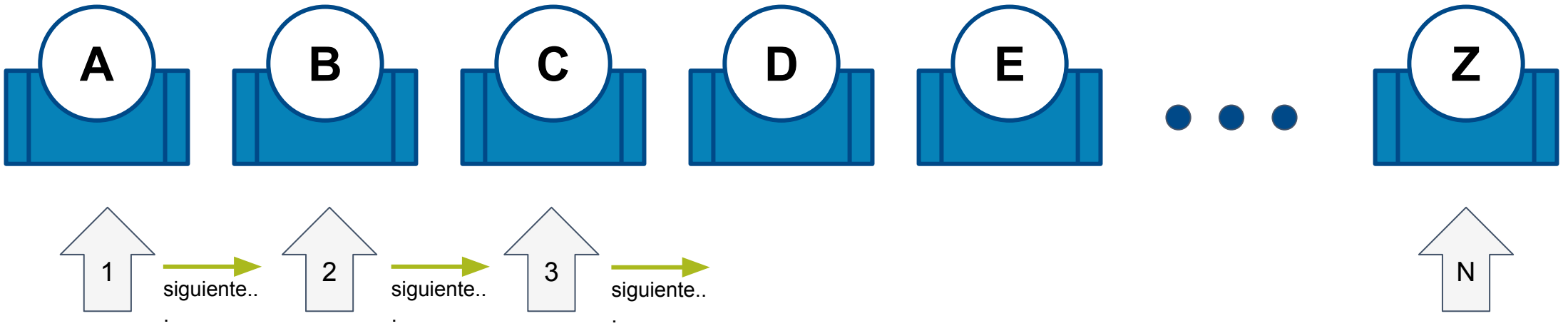
Permite obtener el elemento contenido que coincida con la clave requerida.

```
elem = lista.search("D")
```

iterator()

Permite recorrer la lista completa con for-each, pero también se puede utilizar para recorrer una lista en forma parcial con while.

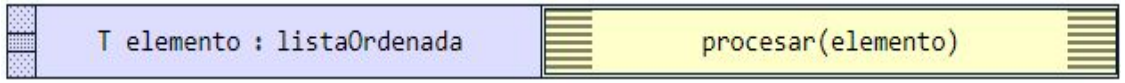
Procesar los elementos de la lista: se aprovecha que ésta es *Iterable*



el *Iterador* se desplaza elemento tras elemento...

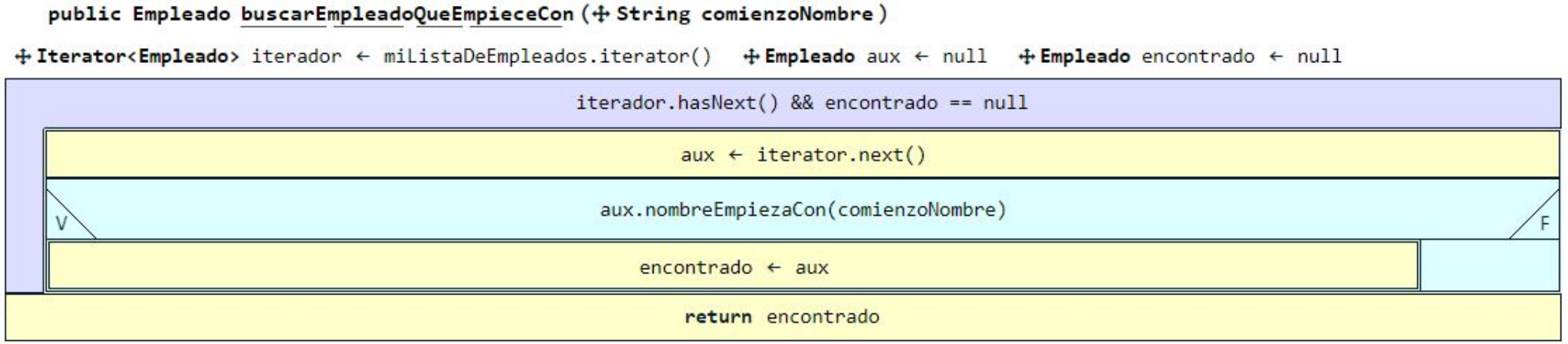
Iteradores

Recorrido completo



El uso del iterador es *implícito* pues es utilizado directamente por el ciclo, que lo toma de la estructura iterable.

Recorrido incompleto con iterador en vez de get



Necesitamos hacer *explícito* el uso del iterador y sus métodos *hasNext()* y *next()*.