

## 9.D. Listas.

### 3. Listas (III).

¿Y en qué se diferencia un `LinkedList` de un `ArrayList`? Los `LinkedList` utilizan listas doblemente enlazadas, que son listas enlazadas (como se vio en un apartado anterior), pero que permiten ir hacia atrás en la lista de elementos. Los elementos de la lista se encapsulan en los llamados **nodos**. Los nodos van enlazados unos a otros para no perder el orden y no limitar el tamaño de almacenamiento. Tener un doble enlace significa que en cada nodo se almacena la información de cuál es el siguiente nodo y además, de cuál es el nodo anterior. Si un nodo no tiene nodo siguiente o nodo anterior, se almacena `null` o nulo para ambos casos.

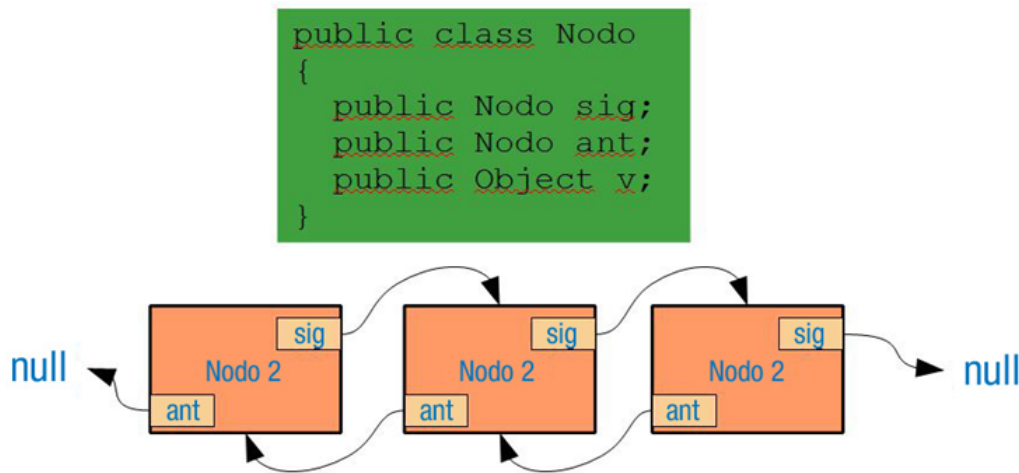


Imagen procedente de curso de Programación del MECD.

No es el caso de los `ArrayList`. Estos se implementan utilizando arrays que se van redimensionando conforme se necesita más espacio o menos. La redimensión es transparente a nosotros, no nos enteramos cuando se produce, pero eso redundará en una diferencia de rendimiento notable dependiendo del uso. Los `ArrayList` son más rápidos en cuanto a acceso a los elementos, acceder a un elemento según su posición es más rápido en un array que en una lista doblemente enlazada (hay que recorrer la lista). En cambio, eliminar un elemento implica muchas más operaciones en un array que en una lista enlazada de cualquier tipo.

¿Y esto que quiere decir? Que si se van a realizar muchas operaciones de eliminación de elementos sobre la lista, conviene usar una lista enlazada (`LinkedList`), pero si no se van a realizar muchas eliminaciones, sino que solamente se van a insertar y consultar elementos por posición, conviene usar una lista basada en arrays redimensionados (`ArrayList`).

`LinkedList` tiene otras ventajas que nos puede llevar a su uso. Implementa las interfaces `java.util.Queue` y `java.util.Deque`. Dichas interfaces permiten hacer uso de las listas como si fueran una cola de prioridad o una pila, respectivamente.

Las colas, también conocidas como colas de prioridad, son una lista pero que aportan métodos para trabajar de forma diferente. ¿Tú sabes lo que es hacer cola para que te atiendan en una ventanilla? Pues igual. Se trata de que el que primero llega es el primero en ser atendido (FIFO). Simplemente se aportan tres métodos nuevos: meter en el final de la lista (`add` y `offer`), sacar y eliminar el elemento más antiguo (`poll`), y examinar el elemento al principio de la lista sin eliminarlo (`peek`). Dichos métodos están disponibles en las listas enlazadas `LinkedList`:

- `boolean add(E e)` y `boolean offer(E e)`, retornarán `true` si se ha podido insertar el elemento al final de la `LinkedList`.
- `E poll()` retornará el primer elemento de la `LinkedList` y lo eliminará de la misma. Al insertar al final, los elementos más antiguos siempre están al principio. Retornará `null` si la lista está vacía.
- `E peek()` retornará el primer elemento de la `LinkedList` pero no lo eliminará, permite examinarlo. Retornará `null` si la lista está vacía.

Las pilas, mucho menos usadas, son todo lo contrario a las listas. Una pila es igual que una montaña de hojas en blanco, para añadir hojas nuevas se ponen encima del resto, y para retirar una se coge la primera que hay, encima de todas. En las pilas el último en llegar es el primero en ser atendido. Para ello se proveen de tres métodos: meter al principio de la pila (`push`), sacar y eliminar del principio de la pila (`pop`), y examinar el primer elemento de la pila (`peek`, igual que si usara la lista como una cola).

Las pilas se usan menos y haremos menos hincapié en ellas. Simplemente ten en mente que, tanto las colas como las pilas, son una lista enlazada sobre la que se hacen operaciones especiales.

### Autoevaluación

Dada la siguiente lista, usada como si fuera una cola de prioridad, ¿cuál es la letra que se mostraría por la pantalla tras su ejecución?

```
LinkedList<String> tt=new LinkedList<String>();
```

```
tt.offer("A"); tt.offer("B"); tt.offer("C");
```

```
System.out.println(tt.poll());
```

- ☐ A.
- ☐ C.
- ☐ D.