Estructuras de Datos
Universidad Nacional Autónoma de México
Laboratorio
Miriam Torres Bucio
Jueves 06 de Febrero de 2025

¿Qué es una lista?

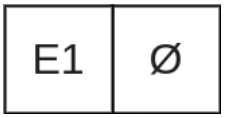
List en Java proporciona la posibilidad de mantener una colección de elementos de manera de que estos estén ordenados. Contiene los métodos basados en índices para insertar, actualizar, eliminar y buscar los elementos. Puede tener los elementos duplicados también. También podemos almacenar los elementos nulos en la lista.

Otra definición de lo que es una lista es la siguiente:

$$Lista = \begin{cases} \emptyset \\ \textit{Un elemento distinto del vacío, seguido de una lista} \end{cases} \tag{1}$$

Es decir, que la lista sin elementos se trata de la lista vacía la cuál de manera gráfica se vería de la siguiente manera:

Ahora bien, también existe la lista con un elemento que es precisamente la lista con dicho elemento seguido de la lista vacía.



Observemos que esta definición de listas es equivalente a la definición de los números naturales utilizando conjuntos donde se asocia al número cero con el conjunto vacío y al número uno como el conjunto que contiene al conjunto vacío, etc.

¿Cómo se crea una lista?

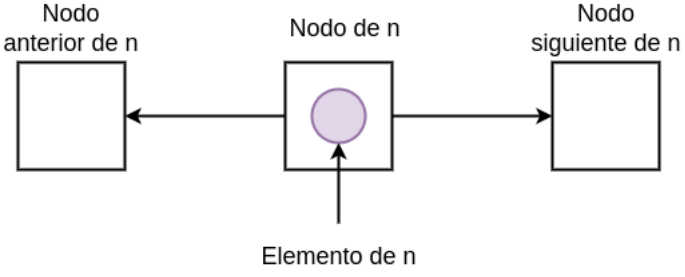
Una lista es de cierta forma es algo complicado de implementar en un lenguaje orientado a objetos como Java, pues recordemos que la idea de los objetos es que tienen identidad, propiedades y comportamiento por lo que a cualquier objeto instancia de lista le debemos poder pedir que ejecute cualquiera de los comportamientos que se hayan definido.

Sin embargo, aunque la definición de lista es correcta por lo anterior podemos ver que es de cierta manera difícil poder implementarlo, para poder cambiar esto haremos uso de nodos.

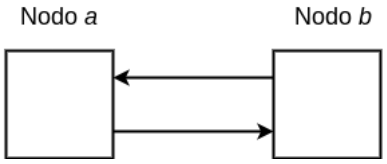
De esta manera, la definición de un nodo es la siguiente:

$$Nodo = \begin{cases} \emptyset \\ \textit{Un elemento con un nodo anterior y un nodo siguiente} \end{cases} \tag{2}$$

Entonces, un nodo es el nodo vacío o un elemento con un nodo anterior y un nodo siguiente.



Una restricción que vamos a tener es que si el nodo a es el nodo anterior del nodo b, esto ocurre si y solo si el nodo siguiente del nodo a es b.



Ahora, un ejemplo de usar dichos nodos en Java sería con una lista de compras de super mercado:

```
public class Nodo{
    Compra elemento;
    Nodo anterior;
    Nodo siguiente;
}
```

El elemento del nodo son artículos del super mercado.
Recordemos que debemos poner las propiedades privadas además de implementar los propios get y set.

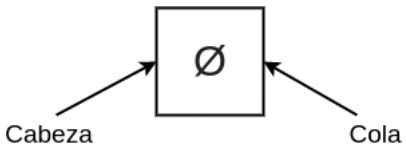
Nota: Los set dan la posibilidad de que usuarios externos a la clase Nodo no le den la suficiente consistencia a los Nodos.

Definición de lista con Nodos.

Tengamos en cuenta que hasta ahora solo hemos definido a los nodos, entonces para poder usarlos de una mejor manera vamos a definir a las listas de la siguiente forma:

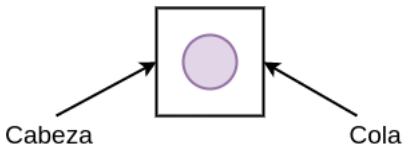
Una lista es un nodo al cual llamaremos cabeza y un nodo al cual llamaremos rabo los cuales representarán al nodo que contiene al primer elemento de la lista y al nodo que contiene al último elemento de la lista respectivamente.

Entonces, la **lista vacía** se ve de la siguiente manera:



Es decir, el nodo cabeza y el nodo rabo son vacíos.

Una **lista con un solo elemento**:



La cabeza y el rabo son el mismo nodo y este a su vez es distinto del vacío.
Recordemos que Java utiliza **referencias**, por lo que podemos tener dos referencias distintas apuntando al mismo objeto.

Una lista con varios elementos:

Si empezamos por la cabeza, nos movemos al nodo siguiente hasta llegar al rabo o bien, si empezamos desde el rabo nos movemos al nodo anterior hasta llegar a la cabeza.

Así como el nodo anterior a la cabeza es el vacío y el nodo siguiente al rabo es el vacío.

Ahora bien, regresando un poco a nuestra implementación de Listas, nuestra lista de compras del super mercado se vería de la siguiente manera:

```
public class ListaCompras{
    private class Nodo{
        private Compra elemento;
        private Nodo anterior;
        private Nodo siguiente;
        /** Constructor del Nodo. */
        private Nodo(Compra elemento){
            this.elemento = elemento;
        }
    }
    /** Propiedades de la lista. */
    private Nodo cabeza;
    private Nodo rabo;
    private int longitud;
}
```

Las listas de las compras pueden ver las propiedades de los nodos aunque sean privados, lo cual nos garantiza el encapsulamiento de datos y nos da cierta libertad de trabajar con los nodos sin la necesidad de preocuparnos por los getters y setters.

Veamos un ejemplo de un algoritmo de las listas, en este caso agregaremos elementos al final de la lista. Nuestro método se va a llamar **agregaFinal()** y sería de la siguiente manera:

```
public void agregaFinal(Compra elemento){
    if(elemento == null) return;
    Nodo n = new Nodo(elemento);
    if(esVacia()){
        cabeza = rabo = nodo;
    } else {
        n.anterior = rabo;
        rabo.siguiente = n;
        rabo = n;
    }
    longitud++;
}
```