

## Laboratorio Nro. 4

# Implementación de Listas Enlazadas

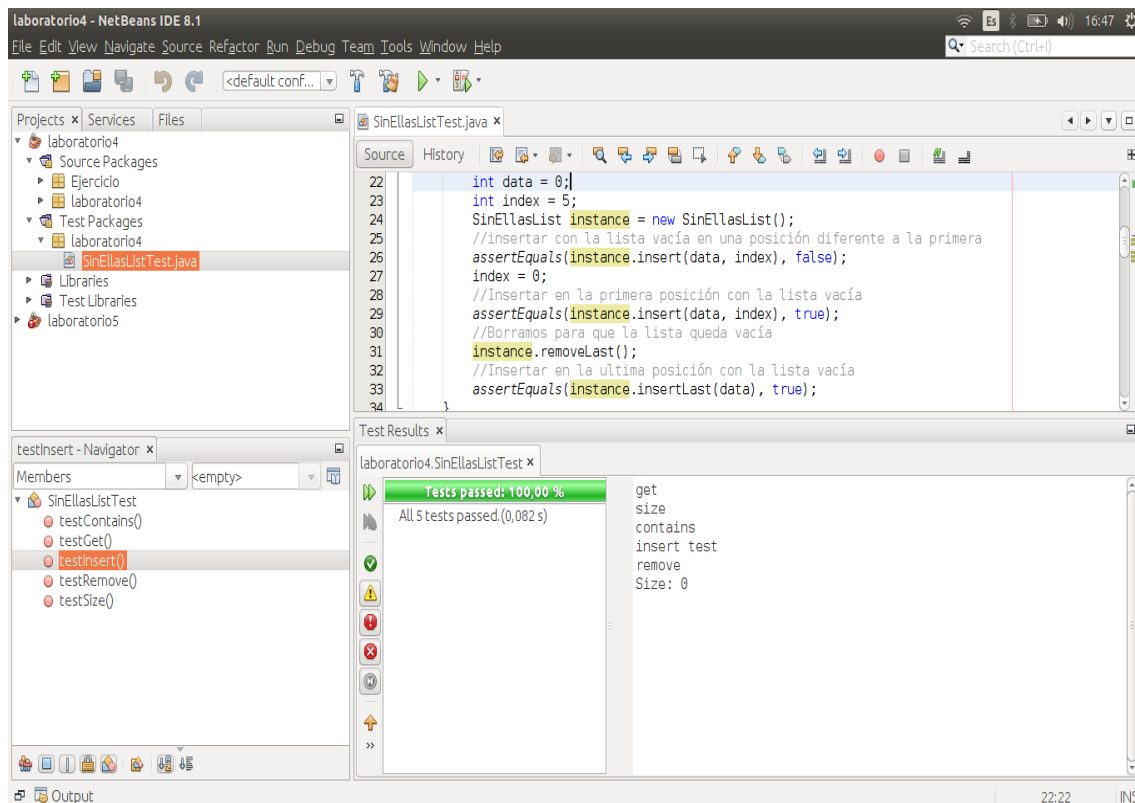
**Kevin Arley Parra Henao**  
Universidad EAFIT  
Medellín, Colombia  
kaparrah@eafit.edu.co

**Daniel Alejandro Mesa Arango**  
Universidad EAFIT  
Medellín, Colombia  
damesaa@eafit.edu.co

### 3) Simulacro de preguntas de sustentación de Proyectos

Las respuestas se dan en los numerales indicados en la guía del laboratorio.

#### 3.1) Tests del numeral 1.2 utilizando JUnit



### 3.2) Explicacion del ejercicio en línea del numeral 2.1

En primer lugar se declara un Scanner para la lectura de cadenas de caracteres y números enteros, luego de esto se lee el numero  $n$ , el cual representa el tamaño o número de cubos, luego se instancia la clase Laboratorio4 que recibe en su constructor un numero entero el cual fue mencionando anteriormente, se construye entonces un ciclo for que ira hasta  $(n-1)$  como se indica en la guía del problema, cada vez se crea una pila y en ella se coloca en la primer posición la "i" que es el iterador del for, es decir cada pila inicia con el número de su posición en el array, volviendo al main, después de hacer la construcción del array, se lee una cadena de caracteres que indica el primer comando "move", "pile" o "quit" para terminar el programa, se entra en un ciclo while el cual se ejecutara hasta que el la cadena ingresada sea "quit", luego se lee un entero que en el problema representa "a" y es el entero o cubo a mover, se procede a leer la segunda cadena de caracteres la cual indica "onto" o "over" y nos indica otra operación más que se combina con la primera leída, se procede entonces a mirar con cuál de las dos primeras palabras leídas coincide para luego mirar con cuál de las otras dos coincide, serian cuatro opciones en total que se pueden ejecutar, a continuación se procede en la explicación de cada una de ellas:

- move a onto b: si este método es ejecutado, recibe los dos enteros, si son diferentes realiza las operaciones si no, no hace nada, en caso de que no sean iguales procede a entrar en un ciclo for, el cual recorrerá todas las posiciones del array, luego tenemos dos variables booleanas que nos dirán si "a" y "b" están en la pila actual, seguidamente en un if miramos si las dos no están al mismo tiempo en dicha pila, si lo están no se hace nada, en caso de no estar en la misma pila entramos en la condición y hay dos condiciones más, una verifica si es "a" la que está en la pila y la otra verifica si es "b" la que está en la pila, si es "a", entonces mediante la variable "po" saca de la pila el último elemento en ella, luego entra en un while si este elemento es diferente de "a", se sacaran entonces los elementos y se retornaran a su posición de origen hasta que el elemento que este afuera sea igual a "a", cuando pase esto el elemento queda afuera ya que será ingresado sobre el bloque "b", por esto cuando la posición actual en el array contenga a "b", pasara igual que con "a", se sacan elementos hasta que llegue a "b", pero esta vez vuelve a ingresar "b" y luego ingresa a "a".
- move a over b: este funciona exactamente igual que onto, pero con una pequeña diferencia "a" estará en la misma pila de "b", por esto, no necesitamos sacar todos elementos que estén encima de "b", solamente procederemos a hacer lo mismo que se explicó en onto con "a", y simplemente en la pila que este "b" se agrega "a".

- **pile a onto b:** su ejecución es muy parecida a los métodos anteriores, pero esta vez será un poco más complicado, para este caso ejecutaremos las acciones siempre que “a” sea diferente que “b”, igual que en los anteriores, seguidamente declaramos dos enteros, que más adelante contendrán las posiciones de “a” y “b”, recorremos todas las posiciones del array en el peor de los casos y cada una de las pilas para encontrar las posiciones de las pilas donde se encuentra “a” y “b”, estas se guardan en las variables mencionadas anteriormente, ahora procedemos a verificar que las posiciones sean diferentes, si no lo son no se hace nada, en caso de que si sean diferentes procedemos a declarar una variable “po”, que contendrá los valores de la pila de “b”, luego en el ciclo while verificamos que la variable sacada de la pila sea diferente de “b”, si lo es se entra en el ciclo y se reubican los elementos en sus posiciones originales hasta llegar a “b”, como la sacamos para verificar que fuera “b”, volvemos a meter en la pila a “b”, luego creamos una pila auxiliar y con la misma variable utilizada para el procedimiento anteriormente descrito sacamos el primer elemento de la posición donde está “a”, y se entra en el ciclo en caso de que no sea “a” e igual que con b sacamos elementos hasta llegar a “a”, pero esta vez guardamos esos elementos en la pila auxiliar, cuando lleguemos a “a”, la metemos en la pila auxiliar y en el ciclo que sigue que se hará hasta que la pila auxiliar este vacía, meteremos encima del bloque “b” los elementos de la pila auxiliar.
- **Pile a over b:** este procedimiento es exactamente el mismo que el anterior solo que esta vez solo ejecutaremos los procedimientos para “a”, ya que no hace falta despejar “b” ya que solo es poner los elementos apilados sobre “a”, en la pila donde se encuentre “b”.

Luego el programa pide de nuevo una cadena de caracteres en caso de ser “move” o “pile” vuelve a pedir los enteros y la segunda cadena de caracteres en caso de ser “quit” se sale y se llama el método “recorrerImprimir” que se encarga de recorrer todas las posiciones del array y a su vez saca de allí y pone en una pila auxiliar los elementos, luego se imprime la pila auxiliar y se repite el procedimiento con cada una de las posiciones del array.

### 3.3) Complejidad del ejercicio en línea del numeral 2.1

```
public class Laboratorio4 {  
    private final ArrayList<Stack> pilaInicial = new ArrayList<>();  
    private final int tamaño;
```

```
    public Laboratorio4(int n) {  
        this.tamaño = n;           //c  
        for (int i = 0; i < tamaño; i++) { //cn  
            Stack pilaA = new Stack(); //cn  
            pilaA.push(i);           //cn  
            pilaInicial.add(pilaA); //cn  
        }  
    }  
}
```

$T(n) = c + cn + cn + cn + cn$   
 $O(cn)$   
 $O(n)$

```
    private void recorrerImprimir() {  
        Stack pilaActual = new Stack(); //c  
        for (int i = 0; i < tamaño; i++) { //cn  
            System.out.print(i+":"); //cn  
            while (!pilaInicial.get(i).empty()) { //c*m  
                pilaActual.push(pilaInicial.get(i).pop()); //n*m  
            }  
            while (!pilaActual.empty()) //c*m  
            {  
                System.out.print(" "+pilaActual.pop()); //n*m  
            }  
            System.out.println(); //c*n  
        }  
    }
```

$T(n) = c + cn + cn + c*m + n*m + c*m + n*m + c*n$   
 $O(c + cn + cn + c*m + n*m + c*m + n*m + c*n)$   
 $O(n*m)$

```
private void moveOnto(int a, int b) {
    if (a != b) { //c
        for (int i = 0; i < tamaño; i++) { //c*n
            boolean poA = pilalNicial.get(i).contains(a); //c*m*n
            boolean poB = pilalNicial.get(i).contains(b); //c*m*n
            if (poA != poB) { //c*n
                if (poB) { //c*n
                    int po = (int) pilalNicial.get(i).pop(); //c
                    while (po != b) { //c*m
                        pilalNicial.get(po).push(po); //c*m
                        po = (int) pilalNicial.get(i).pop(); //c*m
                    }
                    pilalNicial.get(i).push(b); //c
                    pilalNicial.get(i).push(a); //c
                }
                if (poA) { //c*n
                    int po = (int) pilalNicial.get(i).pop(); //c
                    while (po != a) { //c*m
                        pilalNicial.get(po).push(po); //c*m
                        po = (int) pilalNicial.get(i).pop(); //c*m
                    }
                }
            }
        }
    }
}
```

$T(n) = c*n + n*m*c + c*n*m + c*n + c*n + c + c*m + c*m + c*m + c + c + c + c*n + c + c*m + c*m + c*m$   
 $O(c*n + n*m*c + c*n*m + c*n + c*n + c + c*m + c*m + c*m + c + c + c + c*n + c + c*m + c*m + c*m)$   
 $O(n*m)$

```
private void moveOver(int a, int b) {  
    if (a != b) { //c  
        for (int i = 0; i < tamaño; i++) { //c*n  
            boolean poA = pilalInicial.get(i).contains(a); //c*n*m  
            boolean poB = pilalInicial.get(i).contains(b); //c*n*m  
            if (poA != poB) { //c*n  
                if (poA) { //c*n  
                    int po = (int) pilalInicial.get(i).pop(); //c  
                    while (po != a) { //c*m  
                        pilalInicial.get(po).push(po); //c*m  
                        po = (int) pilalInicial.get(i).pop(); //c*m  
                    }  
                }  
                if (poB) { //c*n  
                    pilalInicial.get(i).push(a); //c  
                }  
            }  
        }  
    }  
}
```

$T(n) = c + c*n + c*n*m + c*n*m + c*n + c*n + c + c*m + c*m + c*m + c*n + c$   
 $O(c + c*n + c*n*m + c*n*m + c*n + c*n + c + c*m + c*m + c*m + c*n + c)$   
 $O(n*m)$

```
private void moverPilaOnto(int a, int b) {  
    if (a != b) { //c  
        int posicionDeB = 0; //c  
        int posicionDeA = 0; //c  
        for (int i = 0; i < tamaño; i++) { //c*n  
            boolean poA = pilalInicial.get(i).contains(a); //c*n*m  
            boolean poB = pilalInicial.get(i).contains(b); //c*n*m  
            if (poA) { //c*n  
                posicionDeA = i; //c  
            }  
            if (poB) { //c*n  
                posicionDeB = i; //c*n  
            }  
        }  
    }  
}
```

```
if (posicionDeA != posicionDeB) { //c
    int po = (int) pilalNicial.get(posicionDeB).pop(); //c
    while (po != b) { //c*m
        pilalNicial.get(po).push(po); //c*m
        po = (int) pilalNicial.get(posicionDeB).pop(); //c*m
    }
    pilalNicial.get(posicionDeB).push(b); //c
    Stack Aux = new Stack(); //c
    po = (int) pilalNicial.get(posicionDeA).pop(); //c
    while (po != a) { //c*m
        Aux.push(po); //c*m
        po = (int) pilalNicial.get(posicionDeA).pop(); //c*m
    }
    Aux.push(a); //c
    while (!Aux.empty()) { //c*m
        pilalNicial.get(posicionDeB).push(Aux.pop()); //c*m
    }
}
}
```

$T(n) = c + c + c + c*n + c*n*m + c*n*m + c*n + c + c*n + c*n + c + c*m + c*m + c*m + c + c + c*m + c*m + c*m + c + c*m + c*m$

$O(c + c + c + c*n + c*n*m + c*n*m + c*n + c + c*n + c*n + c + c*m + c*m + c*m + c + c + c*m + c*m + c*m + c + c*m + c*m)$   
 $O(n*m)$

```
private void movePilaOver(int a, int b) {
    int posicionDeB = 0; //c
    int posicionDeA = 0; //c
    if (a != b) { //c
        for (int i = 0; i < tamaño; i++) { //c*n
            boolean poA = pilalNicial.get(i).contains(a); //c*n*m
            boolean poB = pilalNicial.get(i).contains(b); //c*n*m
            if (poA) { //c*n
                posicionDeA = i; //c
            }
            if (poB) { //c*n
                posicionDeB = i; //c
            }
        }
    }
}
```

```
if (posicionDeA != posicionDeB) {           //c
    Stack Aux = new Stack();                //c
    int po = (int) pilalNicial.get(posicionDeA).pop(); //c
    while (po != a) {                        //c*m
        Aux.push(po);                       //c*m
        po = (int) pilalNicial.get(posicionDeA).pop(); //c*m
    }
    Aux.push(a);                            //c
    while (!Aux.empty()) {                   //c*m
        pilalNicial.get(posicionDeB).push(Aux.pop()); //c*m
    }
}
}
```

$T(n) = c+c+c+c*n+c*n*m+c*n*m+c*n+c+c*n+c+c+c+c+c*m+c*m+c*m+c*c*m+c*m$   
 $O(c+c+c+c*n+c*n*m+c*n*m+c*n+c+c*n+c+c+c+c+c*m+c*m+c*m+c*c*m+c*m)$   
 $O(n*m)$

```
public static void main(String[] args) throws IOException {
    Scanner in = new Scanner(System.in); //c
    int n = in.nextInt();                //c
    Laboratorio4 uno = new Laboratorio4(n); //n
    String indicacion = in.next();        //c
    while (!indicacion.equals("quit")) {
        int a = in.nextInt();            //c
        String dos = in.next();           //c
        int b = in.nextInt();             //c
        if (indicacion.equals("move")) { //c
            if (dos.equals("onto")) { //c
                uno.moveOnto(a, b);      //n*m
            } else {
                uno.moveOver(a, b);      //n*m
            }
        } else if (indicacion.equals("pile")) { //c
            if (dos.equals("onto")) { //c
                uno.moverPilaOnto(a, b);  //n*m
            } else {
                uno.movePilaOver(a, b);   //n*m
            }
        }
    }
}
```



```
    }  
    indicacion = in.next();           //c  
  }  
  System.out.println();              //c  
  uno.recorrerImprimir();             //n*m  
}  
no se tiene en cuenta el ciclo que se encarga de pedir los datos cada vez  
T(n) = c+c+n+c+c+c+c+c+c+n*m+n*m+c+c+n*m+n*m+c+c+n*m  
O(c+c+n+c+c+c+c+c+c+n*m+n*m+c+c+n*m+n*m+c+c+n*m)  
O(n*m)  
}
```

La complejidad de cada operación es  $O(n*m)$  donde  $n$  es el tamaño del array y  $m$  es el tamaño de cada pila, en el main no se tuvo en cuenta el ciclo ya que se encarga de pedir los datos al usuario, en total, el programa tiene una complejidad de  $O(n*m)$

### 3.4) Explicación de las variables que se involucran en la complejidad del numeral 3.3

Utilizamos  $n$  y  $m$ ,  $n$  representa el tamaño del arreglo, y  $m$  representa el tamaño en las pilas ya que este varia generalizamos los tamaños de las pilas con  $m$ ,  $n$  será un tamaño fijo desde un principio ya que es el tamaño del array, pero  $m$  depende de cuantas posiciones tengan las pilas, en el peor de los casos buscar donde están los elementos será  $n*m$  porque habrá que recorrer todos los elementos del array y además todas las posiciones de la cada una de las pilas.

### 4) Simulacro de Parcial

1.
  - a) `lista.size()`
  - b) `lista.push(auxiliar.removeFirst())`
2.
  - a)
    - a. `auxiliar1.size() > 0`
    - b. `auxiliar2.size() > 0`
  - b) `personas.offer(edad)`
3. c

## 6. Trabajo en Equipo y Progreso Gradual (Opcional)

### a) Actas de reunión

Integrante	Fecha	Hecho	Haciendo	Por hacer
Daniel Mesa	7/10/2017		punto 2	Explicar el punto 2
Daniel Mesa	8/10/2017	punto 2	puntos 3.2, 3.3, 3.4	simulacro parcial
Kevin Parra	8/10/2017	Punto 1.1-1.2, Punto 3.1	Punto 1.3	Subir Punto 1.3
Daniel Mesa	8/10/2017	puntos 3.2, 3.3, 3.4 y 2 del simulacro	subiendo	

**b) El reporte de cambios en el código**

```
commit f4daf4f2f5d8778e71deb0a4de9608ba9806eb8b (HEAD -> master, origin/master, origin/HEAD)
Merge: 9c6bdb9 b4c948e
Author: evinracher <eevinley@gmail.com>
Date: Sun Oct 8 19:43:11 2017 -0500
```

Merge branch 'master' of <https://github.com/damesaa201710054010/ST0245-033>

```
commit 9c6bdb93db9690bf0ab81667050e61d31e15b72d
Merge: 6652d30 62cd6a7
Author: evinracher <eevinley@gmail.com>
Date: Sun Oct 8 19:40:22 2017 -0500
```

punto 1.3 terminado

```
commit b4c948eef70a7339f759e55408d4f0ef903ce899
Author: eafit-201710093010 <30359351+eafit-201710093010@users.noreply.github.com>
Date: Sun Oct 8 18:36:43 2017 -0500
```

Add files via upload

```
commit 92e3bd89681a740368072812ffccca2a11f505b1
Author: Daniel Mesa <damesaa@eafit.edu.co>
Date: Sun Oct 8 17:41:51 2017 -0500
```

algunos cambios

```
commit a4fbef0d9f074ea6bea0d79c686418a714cd23a7
Author: Daniel Mesa <damesaa@eafit.edu.co>
Date: Sun Oct 8 17:20:23 2017 -0500
```

subiendo plantilla de informe

```
commit b1f32c1ad097e311651d2c216862eb99b9695979
Author: eafit-201710093010 <30359351+eafit-201710093010@users.noreply.github.com>
Date: Sun Oct 8 16:43:53 2017 -0500
```


Add files via upload

```
commit 328238adca8b45a41fcd38b4144312210e7ab705
Author: Daniel Mesa <damesaa@eafit.edu.co>
Date: Sun Oct 8 16:42:15 2017 -0500
```

Punto 2 terminado

```
commit 62cd6a704e15f2ab91ffa71d17524c2cc9af5052
Author: Daniel Mesa <damesaa@eafit.edu.co>
Date: Fri Oct 6 13:06:01 2017 -0500
```

Añadiendo estructura

	<p style="text-align: center;">UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS</p>	<p>Código: ST245</p> <p>Estructura de Datos 1</p>
---	---	---

c) El reporte de cambios del informe de laboratorio

