

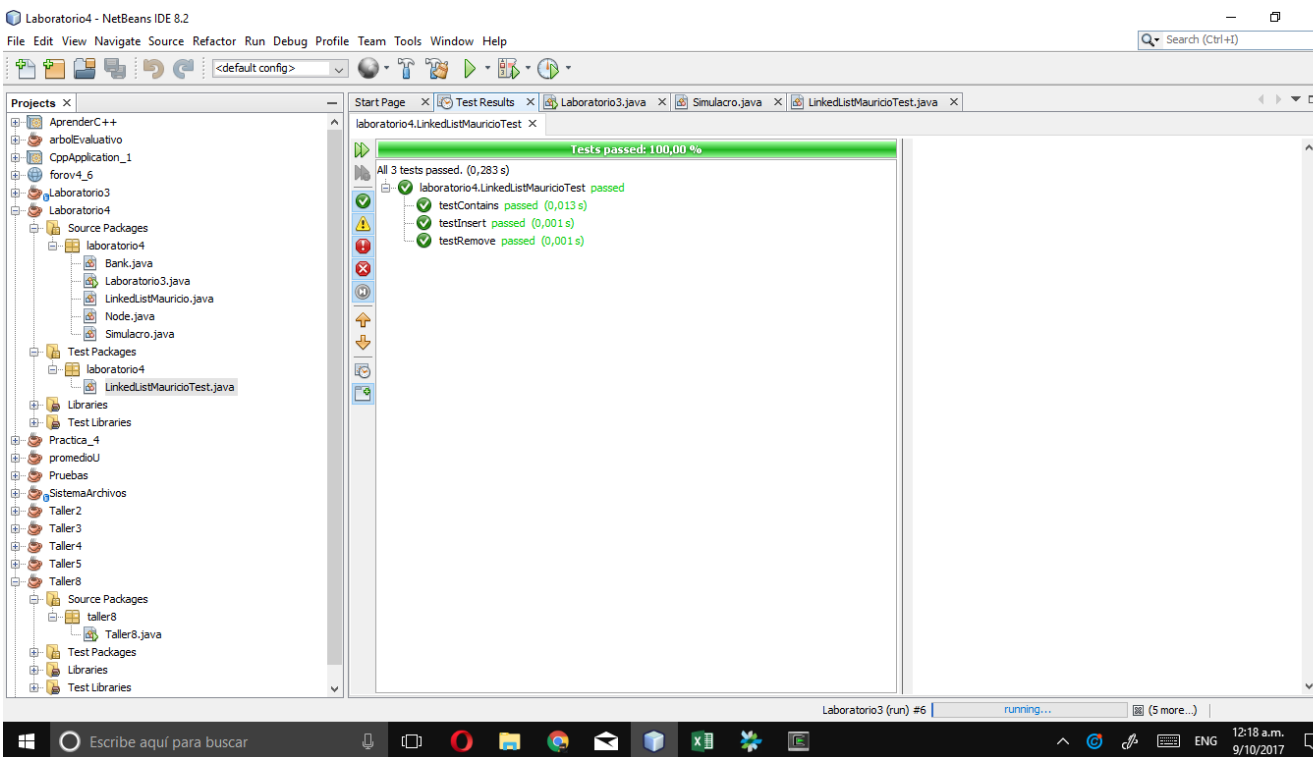
Laboratorio Nro. 4: pilas

Luis Javier palacio mesa
Universidad Eafit
Medellín, Colombia
ljpalaciom@eafit.edu.co

Santiago Castrillón Galvis
Universidad Eafit
Medellín, Colombia
scastrillg@eafit.edu.co

Kevyn Santiago Gómez Patiño
Universidad Eafit
Medellín, Colombia
ksgomez@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. 

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

2. Primero se solicita un entero n al usuario para con esta longitud crear un arreglo posteriormente llenar cada uno de sus espacios con pilas que contienen un número igual al de su posición en el arreglo (estos números corresponden a los bloques).

Luego de esto llamamos el método moverBloques. Este método va a recibir los comandos y va a identificarlos para efectuar las diferentes operaciones que son moveOnto, moveOver, pileOnto, pileOver las cuales son las que efectúan los diferentes movimientos de los bloques y las pilas. Estos procesos se repetirán hasta que se ingrese el comando “quit”, el cual procede a imprimir el estado final del arreglo y de cada una de sus pilas (corresponde a la simulación de los bloques en sus posiciones finales).

3. Complejidad $O(n^2)$

```
public class ejercicioEnLinea {

static Stack arrayBlock[]; C1

private static int buscar(int a) {
    for (int i = 0; i < arrayBlock.length; i++) { C2*n
        if (arrayBlock[i].contains(a)) { n*n
            return i;
        }
    }
    return 0;
}

public static Stack[] generarArray() {
    Scanner in = new Scanner(System.in); C3
    int n = in.nextInt(); C4
    arrayBlock = new Stack[n]; C5
    for (int i = 0; i < n; i++) { C6*n
        Stack a = new Stack(); C7
        a.push(i); C8
        arrayBlock[i] = a; C9
    }
    return arrayBlock;
}

public static void moverBloques() {
    Scanner in = new Scanner(System.in); C10
```

```
String opcion; C11
do {
    opcion = in.nextLine(); C12
    String n[] = opcion.split(" "); C13
    if (n[0].equals("move")) { C14
        if (n[2].equals("onto")) { C15
            moveAontoB(Integer.parseInt(n[1]), Integer.parseInt(n[3]));
        } else {
            moveAoverB(Integer.parseInt(n[1]), Integer.parseInt(n[3]));
        }
    } else if (n[0].equals("pile")) { C16
        if (n[2].equals("onto")) { C17
            pileAontoB(Integer.parseInt(n[1]), Integer.parseInt(n[3]));
        } else {
            pileAoverB(Integer.parseInt(n[1]), Integer.parseInt(n[3]));
        }
    }
} while (!opcion.equals("quit")); C18
quit();
}

private static void devolver(Stack pila, int lim) {
    while (pila.size() != 1) {
        int aux = (int) pila.peek(); C19*n
        if (aux == lim) { C20
            break;
        }
        arrayBlock[aux].push(aux); C21*n
        pila.pop(); C22*n
    }
}

//n^2 public static void moveAontoB(int a, int b) {
    int posA = buscar(a); n^2
    int posB = buscar(b); n^2
    if (a != b && !arrayBlock[posB].contains(a) &&
!arrayBlock[posA].contains(b)) { C23
        devolver(arrayBlock[posA], a); C24
        devolver(arrayBlock[posB], b); C25
        arrayBlock[posB].push(arrayBlock[posA].pop()); C26
    }
```

```
    }  
}  
  
// n^2 public static void moveAoverB(int a, int b) {  
    int posA = buscar(a);  
    int posB = buscar(b);  
    if (a != b && !arrayBlock[posB].contains(a) &&  
!arrayBlock[posA].contains(b)) {  
        devolver(arrayBlock[posA], a);  
        arrayBlock[posB].push(arrayBlock[posA].pop());  
    }  
}  
  
// n^2 public static void pileAontoB(int a, int b) {  
    int posA = buscar(a);  
    int posB = buscar(b);  
    devolver(arrayBlock[posB], b);  
    if (a != b && !arrayBlock[posB].contains(a) &&  
!arrayBlock[posA].contains(b)) {  
        Stack aux = new Stack();  
  
        while (!arrayBlock[posA].isEmpty()) {  
            int aux1 = (int) arrayBlock[posA].peek();  
            if (aux1 == a) {  
                break;  
            }  
            aux.push(arrayBlock[posA].pop());  
        }  
        while (!aux.isEmpty()) {  
            arrayBlock[posB].push(aux.pop());  
        }  
    }  
}  
  
// n^2 public static void pileAoverB(int a, int b) {  
    int posA = buscar(a);  
    int posB = buscar(b);  
    if (a != b && !arrayBlock[posB].contains(a) &&  
!arrayBlock[posA].contains(b)) {  
        Stack aux = new Stack();  
        while (!arrayBlock[posA].isEmpty()) {
```

```
int aux1 = (int) aux.push(arrayBlock[posA].pop());

if (aux1 == a) {
    break;
}
}
while (!aux.isEmpty()) {
    arrayBlock[posB].push(aux.pop());
}
}
}

/C*n public static void quit() {
    for (int i = 0; i < arrayBlock.length; i++) {

        System.out.println(i + ": " + arrayBlock[i]); C*n
    }
}

public static void main(String[] args) {
    generarArray();
    moverBloques();
}

}
```

$T(n) = C + C'n + C'n^2$
 $T(n)$ es $O(C + C'n + C'n^2)$
 $T(n)$ es $O(C'n^2)$
 $T(n)$ es $O(n^2)$

4.

- **static Stack arrayBlock[]:** Es el array de stacks que vamos a utilizar para simular las pilas de bloques.
- **String opción:** guarda la línea ingresada por el Scanner.
- **String n[]:** es el array donde vamos a almacenar el string opción separado por espacios.
- **int aux:** es una variable auxiliar que almacena el valor superior de la pila.

- **Int lim:** es el valor entero limite hasta donde sacaremos datos incluyéndolo.
- **Int posA:** almacena el valor de la posición donde se encuentra a.
- **Int posB:** almacena el valor de la posición donde se encuentra n.

4) *Simulacro de Parcial*

1. A: *while* (*lista.size()* > 0) {
 B: *lista.add(auxiliar.pop());*
2. A:
 Línea 12: (*!auxiliar1.isEmpty()*)
 Línea 16: (*!auxiliar2.isEmpty()*)
 B:
 Linea 18: *personas.offer(edad);*
3. $O(n^2)$