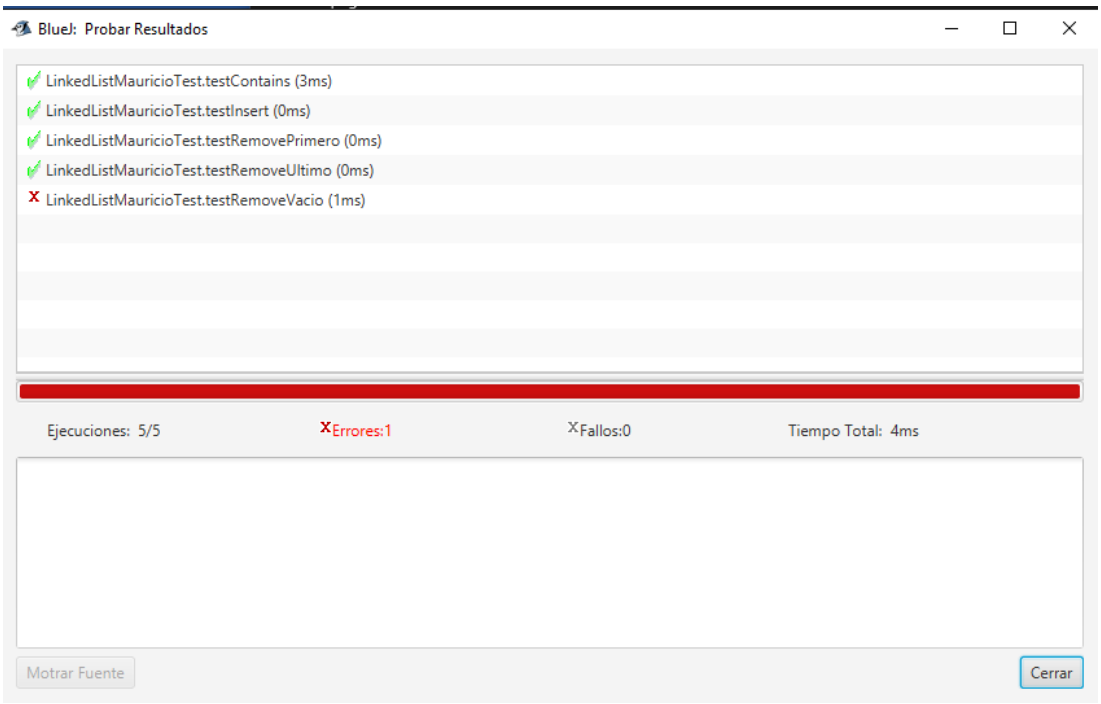
Laboratorio Nro. 4: Implementación de listar enlazadas

Eduard Damiam Londoño
Universidad Eafit
Medellín, Colombia
edlonodnog@eafit.edu.co

Esteban Osorio
Universidad Eafit
Medellín, Colombia
eosorio@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1.



Como la imagen muestra el metodo Contains paso las pruebas exitosamente, el test de insert tenia tanto las pruebas para insertar en el primero, en el ultimo y en una lista vacia y paso exitosamente, el test para remover en la

primera y en la ultima posicion tambien pasaron exitosamente y el test de remover en una lista vacias tambien ya que si bien mostro error esto era lo que se supone debe arrojar ya que no se puede borrar una lista vacia.

2. Tenemos 4 atributos en la clase, una llamada numeroBloques para el numero de bloques, un arreglo de pilas llama bloques para amontonar los bloques, un arreglo llamado posición para las posiciones, un String línea para guardar lo que inserte el usuario y dos enteros para los bloques que se moverán.

Se crea un BufferedReader para poder leer lo que inserte el usuario, numeroBloques se hace igual a la cantidad que inserte el usuario, bloques y posición tendrán un tamaño igual a numeroBloques y luego con un ciclo se llenan las posiciones de bloques (con pilas), y las posiciones de posición

Se inicia la variable línea y se hace un ciclo mientras la entrada sea distinta de quit, entonces se crea un objeto de tipo StringTokenizer (para manejar las palabras por separado), se hace una variable de tipo String llamada primero para guardar la primera palabra, a la variable A se inicializa con el primer número, se hace otra variable de tipo String llamada segundo la cual guarda la segunda palabra y por último se inicializa la variable B con el segundo número, si A y B son iguales se sale devuelve al principio del ciclo, porque el problema dice que este caso debe ser ignorado.

Si primero es igual a “move” y b es igual a “onto” entonces se ejecuta el método MoveOnto o si b es igual a “over” se ejecuta el método MoveOver

El método MoveOnto recibe los dos números que representan los bloques que se moverán, como la única diferencia entre MoveOnto y Move over es que MoveOnto remueve los bloques que estan arriba de del segundo bloque, se limpia lo que está arriba del segundo bloque y se llama al método MoveOver.

MoveOver recibe los dos bloques que se moverán y limpia lo que este encima del primero, entonces mete en la pila que se encuentra dentro del arreglo bloques en la posición del segundo bloque lo que se encuentra en la cima de la pila de bloques en la posición del primer bloque que queremos mover y por último la posición del primer bloque se hace igual a la posición del segundo bloque.

En cambio, sí primero es igual a “pile” y b es igual a “onto” se ejecuta el método PileOnto o si b es igual a over se ejecuta el meto PileOver

Como pile Onto lo unico que tiene de diferente a Pile Over es que se limpia lo que este encima del segundo bloque entonces se limpia lo que está encima del segundo bloque y se llama al método pileOver.

El pileOver se crea una pila y a esa pila se le pasan los elementos que estaban encima del bloque a, para esto se hace un ciclo mientras no se llegue al bloque a se le va pasando a la pila creada los elementos en la pila donde está el bloque a, cuando se acaba el ciclo se le para también a la pila el bloque a (porque el ciclo es hasta que encuentre el elemento), y luego con otro ciclo pasamos lo que está en la pila creada al inicio del método a la pila que representa al bloque b.

Tanto pileOnto como MoveOnto usan un método llamado limpiar este sirve para devolver los bloques a su posición original, entonces lo que se hace es que recibe el bloque que se quiere limpiar y con un ciclo mientras el primer elemento de la pila en la que se encuentra ese bloque no sea ese mismo bloque se pasan a un método llamado intial y lo que hace intial es que si recibe un bloque tenga algo encima lo que hace es que se llama así mismo (recursión) pero con elemento próximo en la pila cuando el bloque que recibe ya no tiene bloques encima mete ese bloque a su posición original y vuelve a poner la posición de ese bloque como la original.

Finalmente, cuando el usuario presiona quit se pasa a un ciclo desde 0 hasta la longitud de los bloques que imprime lo que devuelva un método llamado solve, lo que hace solve es que recibe un entero llamado index y hasta que el bloque número “index” va haciendo un String igual al elemento más arriba la pila número index (removiéndolo en el proceso) concatenado con sí mismo, y cuando este ciclo acaba retorna un String formado por el número que guarda index concatenado unos dos puntos y concatenado con si mismo

3.

```
public static void main (String[] args) throws IOException{
        BufferedReader Input = new BufferedReader(new
InputStreamReader(System.in)); //C1
        numeroBloques = Integer.parseInt(Input.readLine()); //C2
        bloques = new Stack[numeroBloques]; // C3
        posicion = new int [numeroBloques]; //C4
        for (int i = 0; i< numeroBloques; i++){ //n
            bloques[i] = new Stack<Integer>(); //C5
            bloques[i].push(i); //C6
            posicion[i]= i; //C7
```

```
    }

    linea = "";
    while (!(linea=Input.readLine()).equals("quit")){ //m
        StringTokenizer token = new StringTokenizer(linea); //C8
        String primero= token.nextToken(); // C9
        A = Integer.parseInt(token.nextToken()); //C10
        String segundo = token.nextToken(); // C11
        B = Integer.parseInt(token.nextToken()); //C12

        if (A==B  || posicion[A] == posicion[B]){ //C13
            continue; //C14
        }

        if (primero.equals("move")){ //C15
            if (segundo.equals("onto")){ //C16
                MoveOnto(A,B); //= //O(l^2*h)
            }
            else if (segundo.equals("over")){ //C17
                MoveOver(A,B); // //O(l^2*h)
            }
        }
        else if (primero.equals("pile")){ //C18
            if (segundo.equals("onto")){ //C19
                PileOnto(A,B); // //O(l^2*h)
            }
            else if (segundo.equals("over")){ //C20
                PileOver(A,B); //O(2h)
            }
        }
    }
    for (int i=0; i<bloques.length; i++){ //n
        System.out.println(Solve(i)); //(n*l)
    }

}

public static void MoveOnto(int primero,int segundo){ //O(l+l^3) = O(l^3)
    limpiar(segundo); //(l^3)
    MoveOver(primero,segundo); //o(l)
}
```

```
    public static void MoveOver(int primero, int segundo) {   //O(l^3)
        limpiar(primero); //(l^3)
        bloques[posicion[segundo]].push(bloques[posicion[primero]].pop());
//O(1)
        posicion[primero] = posicion[segundo]; //O(1)
    }

    public static void PileOnto(int primero,int segundo){ //O(l^3+2h)
      limpiar(segundo);//l^3
      PileOver(primero,segundo); //O(2h)
    }

     public static void PileOver(int primero, int segundo) { //O(2h)
      Stack<Integer> Pila = new Stack<Integer>();  //C21
      while(bloques[posicion[primero]].peek() != primero) { //h
          Pila.push(bloques[posicion[primero]].pop()); //O(1)
      }
      Pila.push(bloques[posicion[primero]].pop()); //O(1)
      while(!Pila.isEmpty()) { //h
         int Tmp = Pila.pop(); //O(1)
         bloques[posicion[segundo]].push(Tmp); //O(1)
         posicion[Tmp] = posicion[segundo]; //O(1)
      }
    }

    public static void limpiar(int bloque){ //O(l^3)
     while (bloques[posicion[bloque]].peek() != bloque){ //l
         intial(bloques[posicion[bloque]].pop()); //o(l^2)
       }
    }

    public static void intial(int bloque){ // O(l^2)
      while(!bloques[bloque].isEmpty()) { //l
          intial(bloques[bloque].pop()); //O(l)^2
       }
       bloques[bloque].push(bloque); //O(1)
       posicion[bloque] = bloque; //O(1)
    }

      public static String Solve(int Index) { //O(l)
```

```
        String Result = ""; //C22
        while(!bloques[Index].isEmpty()) { //l
         Result = " " + bloques[Index].pop() + Result; //C23

        }
        Result = Index + ":" + Result; //C24
        return Result; //C25
    }
```

T(n)= C'+n+m+ (l^2*h) +(l*n)
T(n) es O(n+m+(l^2*h) +(l*n)) l*n es mayor que n entonces por regla de la suma
T(n) es O(m+(l^2*h) +(l*n)) l^2 *h es mayor que l^2 *n entonces por regla de la suma
T(n) es O(m+(l^2*h))

4. El cálculo de la complejidad del numeral anterior tuvo 4 variables, n era la cantidad de bloques y por lo tanto también era el tamaño de la variable bloques, m representaba la cantidad de veces que el usuario digitaba alguna orden como no sabemos cuántas veces el usuario digitara una orden entonces tratamos ese ciclo con una variable, los métodos para mover recibían bloques a estoy bloques los tratamos con un variable l porque a pesar de que son 2 bloques distintos, la duración de los métodos solo dependía de uno de ellos y por ultimo una variable h que usamos para representar las veces que se ejecutaba un ciclo que dependía del tamaño de alguna pila dentro del arreglo bloques.

4) Simulacro de Parcial

1. a. *lista.size();*
   b. lista.add(auxiliar.pop) ;

2. a.*! auxiliar.isEmpty() , !auxiliar2.isEmpty()*
   b. organizar(auxiliar2);

3. *c*