

Universidad del País Vasco

Escuela Universitaria de Ingeniería Técnica Industrial de Bilbao

Estructuras Datos y Algoritmos

Laboratorío 2:DoubleLinkedList

Andrea Siles Ivan Salazar

Ingeniería Informática de Gestión y Sistemas de Información

November 2, 2015

Contents

1	Intr	roducción	3
2	Dis	eño de clases	3
3	Des	cripción de las estructuras de datos principales	4
4	Dis	seño e implementación de los métodos principales	4
	4.1	Método añadir al comienzo	4
	4.2	Método añadir al final	7
	4.3	Método añadir un elemento detrás de otro	10
	4.4	Método añadir en lista ordenada	12
	4.5	Método borrar el primero	16
	4.6	Método borrar el ultimo	19
	4.7	Método borrar un elemento concreto de la lista	22
	4.8	Método buscar	25
	4.9	Cambios Realizados en Laboratorio 1	27
5	Cor	nclusiones	30
6	cori	recciones laboratorio 1	30

1 Introducción

Conforme ha ido evolucionando la algoritmica se ha ido creando estructuras mas rapidas como por ejemplo las listas enlazas que nos permite que los metodos basicos se realicen en un tiempo muy reducido y esto nos permite procesar grandes cantidades de datos.

2 Diseño de clases

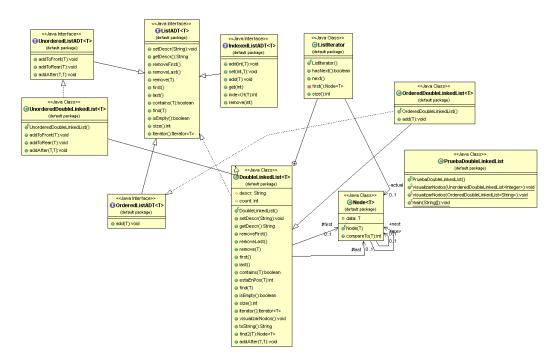


Figure 1: Diagrama de clases del proyecto

3 Descripción de las estructuras de datos principales

Hemos utilizado la siguiente estructura de datos: DoubleLinkedList.

una lista doblemente enlazada es una estructura de datos vinculados que consiste en un conjunto de forma secuencial vinculando registros llamados nodos. Cada nodo contiene dos campos, llamados enlaces, que son referencias a la anterior y al siguiente nodo en la secuencia de nodos. Ademas la lista contiene 2 referencias, el principio (First) y el final (Last) para facilitar el recorrido de la lista. Estas referencias apuntan al primer y último nodo respectivamente.

4 Diseño e implementación de los métodos principales

4.1 Método añadir al comienzo

Casos de prueba

l.inicial	elem	l.final
	X	[x]
[z]	X	[x,z]
[y,z,k]	X	[x,y,z,k]

El coste de añadir un elemento al comienzo de

una lista doblemente ligada es O(1)

```
1 public class UnorderedDoubleLinkedList<T> extends DoubleLinkedList<T> implements
                UnorderedListADT<T> {
          public void addToFront(T elem) {
               // añade un elemento al comienzo
// COMPLETAR EL CODIGO Y CALCULAR EL COSTE
 5
 6
                Node<T> nuevo = new Node<T>(elem);
nuevo.data = elem;
if (this.isEmpty()) {
 8
 9
10
                     nuevo.next = null;
nuevo.prev = null;
11
12
                     first = nuevo;
last = nuevo;
13
14
15
16
                     this.count++;
17
                } else {
18
                     Node<T> Actual = first;
19
20
                     Actual.prev = nuevo;
                     nuevo.next = Actual;
nuevo.prev = null;
21
22
23
24
25
26
27
28
                     first = nuevo;
                     this.count++;
                }
          }
```

Figure 2: Código addToFront

```
public static void visualizarNodos(UnorderedDoubleLinkedList<Integer> 12) {
                                                                                                   <terminated> PruebaDoubleLink
    Iterator<Integer> it = 12.iterator();
System.out.println("");
                                                                                                    Lista .....
    while (it.hasNext()) {
        Integer num = it.next();
        System.out.println(num);
    String k = it.next();
System.out.println(k);
public static void main(String[] args) {
    UnorderedDoubleLinkedList<Integer> 12 = new UnorderedDoubleLinkedList<Integer>();
    //l2.addToFront(1);
//l2.addToFront(3);
    //1.addToFront(6);
// 12.addToRear(7);
    //l2.addToRear(9);
//l2.addToRear(6);
    // 12.addToRear(20);
//1.addToRear(20);
System.out.print(" Lista .....");
    visualizarNodos(12);
}
```

Figure 3: Pruebas Eclipse addToFront

Figure 4: Pruebas Eclipse addToFront

Figure 5: Pruebas Eclipse addToFront

4.2 Método añadir al final

Casos de prueba

l.inicial	elem	l.final	
	X	[x]	El e
[z]	X	[z,x]	
[y,z,k]	X	[y,z,k,x]	

El coste de añadir un elemento al final de una

lista doblemente ligada es O(1)

```
public void addToRear(T elem) {
    // añade un elemento al final
    // COMPLETAR EL CODIGO Y CALCULAR EL COSTE
    Node<T> nuevo = new Node<T>(elem);
    nuevo.data = elem;
    if (this.isEmpty()) {
        nuevo.next = null;
        nuevo.prev = null;
        first = nuevo;
        last = nuevo;
        this.count++;
    } else {
        Node<T> actual = last;
        nuevo.prev = actual;
        actual.next = nuevo;
        nuevo.next = null;
        last = nuevo;
        count++;
    }
```

Figure 6: Codigo addToRear

```
public static void visualizarNodos(UnorderedDoubleLinkedList<Integer> 12) {
   Iterator<Integer> it = 12.iterator();
   System.out.println("");
   while (it.hasNext()) {
        Integer num = it.next();
        System.out.println(num);
        System.out.prin
                                                                                                                                                                                                                                                                                                                                                                                                                                         <terminated> PruebaDoubleLin
                                                                                                                                                                                                                                                                                                                                                                                                                                         Lista .....
                                      System.out.println(num);
 }
                   public static void visualizarNodos(OrderedDoubleLinkedList<String> 1) {
                                      Iterator<String> it = l.iterator();
System.out.println("");
                                      while (it.hasNext()) {
   String k = it.next();
   System.out.println(k);
                                      }
 public static void main(String[] args) {
                   UnorderedDoubleLinkedList<Integer> 12 = new UnorderedDoubleLinkedList<Integer>();
                    //l2.addToFront(1);
//l2.addToFront(3);
                    //l2.addToFront(6);
                      //12.addToFront(7):
                        12.addToRear(7);
                   //12.addToRear(9);
// 12.addToRear(6);
                    //l.addToRear(20);
System.out.print(" Lista .....");
                    visualizarNodos(12);
```

Figure 7: Pruebas Eclipse addToRear

```
public static void visualizarwodos(unordereduoubleLinkedList<integer> 12) {
     Ite static void visualizarNodos(unorde)
Iterator<Integer> it = 12.iterator();
System.out.println("");
                                                                                                                               <terminated> PruebaDoubleLi
     while (it.hasNext()) {
   Integer num = it.next();
   System.out.println(num);
                                                                                                                               Lista .....
                                                                                                                               9
     public static void visualizarNodos(OrderedDoubleLinkedList<String> 1) {
          Iterator<String> it = l.iterator();
System.out.println("");
           while (it.hasNext()) {
                String k = it.next();
System.out.println(k);
}
public static void main(String[] args) {
     UnorderedDoubleLinkedList<Integer> 12 = new UnorderedDoubleLinkedList<Integer>();
      //l2.addToFront(1);
     //12.addToFront(3);
//12.addToFront(6);
     //12.addToFront(7);
12.addToRear(7);
     12.addToRear(9);
// 12.addToRear(6);
     //l.addToRear(20);
     //1.add|Okear(20);
System.out.print(" Lista .....");
visualizarNodos(12);
```

Figure 8: Pruebas Eclipse addToRear

```
PruebaDoubleLinkedListjava ② ① UnorderedDoubleLinkedLists ②
public class PruebaDoubleLinkedList {
    public class PruebaDoubleLinkedList {
        Iterator(Integer) it = 12.iterator();
        So public static void visualizarNodos(UnorderedDoubleLinkedList(Integer) 12) {
        Iterator(Integer) it = 12.iterator();
        System.out.println(num);
    }
    public static void visualizarNodos(OrderedDoubleLinkedList<String> 1) {
        Iterator(string) it = 1.iterator();
        System.out.println(num);
    }
    public static void visualizarNodos(OrderedDoubleLinkedList<String> 1) {
        Iterator(string) it = 1.iterator();
        System.out.println(k);
    }
    public static void main(String[] args) {
        UnorderedDoubleLinkedList(Integer) 12 = new UnorderedDoubleLinkedList(Integer)();
        12.addToRear(?);
        12.a
```

Figure 9: Pruebas Eclipse addToRear

4.3 Método añadir un elemento detrás de otro

Casos de prueba

l.inicial	elem	elem2	l.final	
	a	c	"lista vacia"	
[a]	a	С	[a,c]	
[b,c,d]	k	e	[b,c,d]	El coste de add A fter es de coste $O(n/2)$
[k,d,c]	k	e	[k,e,d,c]	
[b,k,g]	k	е	[b,k,e,g]	
[b,c,k]	k	е	[b,c,k,e]	

aunque el código del método sea constante, ya que hace uso del programa find
2 que es de coste $\mathrm{O}(\mathrm{n}/2)$

```
public void addAfter(T elem, T pTarget) {
    // Añade elem detrás de otro elemento concreto, target, que ya se
    // encuentra en la lista
    // ¡COMPLETAR OPCIONAL!

Node<T> miNode = this.find2(pTarget);
Node<T> nNodo = new Node<T>(elem);
if (miNode != null) {
    nNodo.next = miNode.next;
    nNodo.prev = miNode;
    if (miNode.next != null) {
        miNode.next.prev = nNodo;
    }
    miNode.next = nNodo;
}
```

Figure 10: Codigo addAfter

```
UnorderedDoubleLinkedList<Integer> 12 = new UnorderedDoubleLinkedList<Integer>();
12.addAfter(4, 3); //En este caso probamos el caso en el que la lista esta vacia.
System.out.print(" Lista .......");
visualizarNodos(12); //Al visualizar los nodos no se debe añadic ningun nodo
12.addAfter(4, 3); // Añoar realizamos el metodo añadiendo un nodo de elemento 4 despues del unico nodo de la lista
System.out.print(" Lista ......");
visualizarNodos(12); //Al visualizar los nodos se debe añadic el nodo y visualizar 3,4
12.addToFront(1); // Añadimos un nodo de elemento 1 al principio de la lista y la lista inicial seria 1,3,4
12.addAfter(5, 1); // Añadimos un nodo de elemento 5 despues del primer nodo de una lista de n nodos
System.out.print(" Lista ......");
visualizarNodos(12); //Al visualizar los nodos se escribira por pantalla 1,5,3,4
12.addAfter(6, 5); // Añadimos un nodo de elemento 5 despues de un nodo ubicado en el medio de una lista de n nodos
System.out.print(" Lista ......");
visualizarNodos(12); //Al visualizar los nodos se debe ver 1,5,6,3,4
12.addAfter(7, 4); // Añadimos un nodo de elemento 5 despues del ultimo nodo de una lista de n nodos
System.out.print(" Lista ......");
visualizarNodos(12); //Al visualizar los nodos se debe ver 1,5,6,3,4
12.addAfter(7, 4); // Añadimos un nodo de elemento 5 despues del ultimo nodo de una lista de n nodos
System.out.print(" Lista ......");
visualizarNodos(12); //Al visualizar los nodos se debe ver 1,5,6,3,4,7
```

Figure 11: Pruebas Eclipse

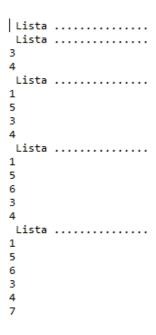


Figure 12: Resultados

4.4 Método añadir en lista ordenada

Casos de prueba

l.inicial	elem	l.final
	a	[a]
[a]	b	[a,b]
[b,c,d]	a	[a,b,c,d]
[b,d,e]	С	[b,c,d,e]
[b,d,e]	Z	[b,d,e,z]

Coste: En este metodo si la lista es vacia añadir

el elemento es O(1); si el elemento a añadir es menor que el primero de la lista el tiempo es O(1); si el elemento a añadir es mayor que el ultimo elementode la lista el tiempo es O(1); En el peor de los casos el elemento a añadir puede que este en el medio y entoces el coste seria O(n/2)

```
public void add(T elem) {
    // COMPLETAR EL CODIGO Y CALCULAR EL COSTE
    boolean enc = false;
    Node<T> nuevo = new Node<T>(elem);
    Node<T> anterior = null;
    Node<T> actual = this.first;
    if (this.isEmpty()) {
        nuevo.next = null;
nuevo.prev = null;
        first = nuevo;
        last = nuevo;
        this.count++;
        System.out.println("" + count);
    }else {
        anterior = null;
        while (actual != null && !enc) {
            if (nuevo.compareTo(actual.data) < 0) {</pre>
                 enc = true;
            } else {
                 anterior = actual;
                 actual = actual.next;
            }
        }
```

Figure 13: Código de add

```
if (!enc) {// añadir al final
        nuevo.next = null;
        nuevo.prev = anterior;
        anterior.next = nuevo;
        last = nuevo;
        count++;
    }
    else if (actual == this.first) {
        Node<T> Actual = first;
        Actual.prev = nuevo;
        nuevo.next = first;
        nuevo.prev = null;
        first = nuevo;
        this.count++;
    }
    else {// insertar en el medio
        nuevo.data = elem;
        nuevo.next = actual;
        anterior.next = nuevo;
        this.count++;
    }
// distinguimos 3 casos
// esta en el principio
// esta en el medio
// esta en el final
// insertar en lista vacia
```

Figure 14: Código de add

```
public static void main(String[] args) {
       OrderedDoubleLinkedList<String>l=new OrderedDoubleLinkedList<String>();
       1.add("bela");
       System.out.print(" Lista .....");
       visualizarNodos(1);
       1.add("era");
       System.out.print(" Lista .....");
       visualizarNodos(1);
       1.add("steve");
       1.add("andre");
       System.out.print(" Lista .....");
       visualizarNodos(1);
       1.add("fran");
       System.out.print(" Lista .....");
       visualizarNodos(Ì);
       1.add("tony");
       System.out.print(" Lista .....");
       visualizarNodos(1);
   }
}
```

Figure 15: Pruebas de add

```
1
Lista .....
bela
Lista .....
bela
Lista .....
andre
bela
era
steve
Lista .....
andre
bela
era
fran
steve
Lista .....
andre
bela
era
fran
steve
tony
```

Figure 16: Resultados

4.5 Método borrar el primero

Casos de prueba

l.inicial	l.final	resultado					
	[]	null	Costo	Fl costo	do osto	motodo	es constante
[y]	[]	null	Coste.	El coste	de este	metodo	es constante
[y,z,e]	[z,e]	у					

porque tenemos un puntero(first) que apunta al primer nodo

```
public T removeFirst() {
    // Elimina el primer elemento de la lista
    // Precondición: la lista tiene al menos un elemento
    // COMPLETAR EL CODIGO Y CALCULAR EL COSTE
    Node<T> actual = this.first;
    if (actual.next == null) {
        this.first = null;
        this.last = null;
        return actual.data;
    } else {
        this.first = actual.next;
        actual.next.prev = null; // first.prev=null
        System.out.println("te he borrado");
        return actual.next.data;
    }
}
```

Figure 17: Codigo de removeFirst

```
import java.util.Iterator;
                                                                                                        <terminated> PruebaDoubleLinked
                                                                                                         Lista .....
public class PruebaDoubleLinkedList {
                                                                                                         Lista .....
    public static void visualizarNodos(UnorderedDoubleLinkedList<Integer> 12) {
        Iterator<Integer> it = 12.iterator();
System.out.println("");
        while (it.hasNext()) {
             Integer num = it.next();
             System.out.println(num);
        }
    }
        public static void visualizarNodos(OrderedDoubleLinkedList<String> 1) {
            Iterator<String> it = l.iterator();
System.out.println("");
             while (it.hasNext()) {
                 String k = it.next();
                 System.out.println(k);
        }
    public static void main(String[] args) {
    UnorderedDoubleLinkedList<Integer> 12 = new UnorderedDoubleLinkedList<Integer>();
        12.addToRear(7);
        System.out.print(" Lista .....");
        visualizarNodos(l2);
        12.removeFirst();
System.out.print(" Lista .....");
        visualizarNodos(l2);
        12.addToRear(9);
         12.addToRear(6);
        12.addToRear(20);
```

Figure 18: Pruebas de removeFirst

```
Iterator<Integer> it = 12.iterator(
                                       <terminated> PruebaDoubleLinkedl
   System.out.println("");
   while (it.hasNext()) {
                                         Lista ......
                                        7
        Integer num = it.next();
                                        9
        System.out.println(num);
                                        6
                                        20
}
   public static void visualizarNodos( te he borrado
                                        Lista .....
        Iterator<String> it = 1.iterato
        System.out.println("");
        while (it.hasNext()) {
                                        20
            String k = it.next();
            System.out.println(k);
    }
public static void main(String[] args)
   UnorderedDoubleLinkedList<Integer>
   12.addToRear(7);
   12.addToRear(9);
    12.addToRear(6);
   12.addToRear(20);
   System.out.print(" Lista .
    visualizarNodos(12);
   12.removeFirst();
   System.out.print(" Lista ......
    visualizarNodos(12);
   12.addToRear(9);
    12.addToRear(6);
   12.addToRear(20);
    **/
```

Figure 19: Pruebas de removeFirst

4.6 Método borrar el ultimo

Casos de prueba

l.inicial	l.final	resultado	
[]	[]	null	- Coste: Coste: El coste de este metodo es con-
[y]	[]	null	Coste. Coste. El coste de este metodo es con-
[y,z,e]	[y,z]	e	

stante porque tenemos un puntero(last) que apunta al ultimo nodo

```
public T removeLast() {// /corregidooooooo
    // Elimina el último elemento de la lista
    // Precondición: la lista tiene al menos un elemento
    // COMPLETAR EL CODIGO Y CALCULAR EL COSTE
    Node<T> actual = this.last;
    if (actual == this.first) {
        this.last = null;
        this.first = null;
        this.count--;
        return actual.data;
    } else {
        actual = actual.prev;
        actual.next = null;
        last = actual;
        this.count--;
        return actual.data;
    }
}
```

Figure 20: Codigo de removeFirst

```
public class PruebaDoubleLinkedList {
                                           <terminated> PruebaDoubleLin
    public static void visualizarNodos(Unor | Lista .....
        Iterator<Integer> it = 12.iterator( 7
                                             Lista ......
        System.out.println("");
        while (it.hasNext()) {
            Integer num = it.next();
            System.out.println(num);
    }
        public static void visualizarNodos(
            Iterator<String> it = 1.iterato
            System.out.println("");
           while (it.hasNext()) {
                String k = it.next();
                System.out.println(k);
        }
    public static void main(String[] args)
        UnorderedDoubleLinkedList<Integer>
        12.addToRear(7);
        System.out.print(" Lista ......
        visualizarNodos(12);
        12.removeLast();
        System.out.print(" Lista ...
        visualizarNodos(12);
        /**
        12.addToRear(9);
        12.addToRear(6);
        12.addToRear(20);
        **/
```

Figure 21: Pruebas de removeFirst

```
Iterator<Integer> it = 12.iterator(
                                        <terminated> PruebaDoubleLinke
    System.out.println("");
                                         Lista .....
    while (it.hasNext()) {
                                        9
        Integer num = it.next();
                                        6
        System.out.println(num);
                                        20
                                         Lista .....
}
    public static void visualizarNodos(
        Iterator<String> it = 1.iterato 6
       System.out.println("");
       while (it.hasNext()) {
            String k = it.next();
            System.out.println(k);
    }
public static void main(String[] args)
    UnorderedDoubleLinkedList<Integer>
    //l2.addToFront(7);
    12.addToRear(9);
     12.addToRear(6);
    12.addToRear(20);
    System.out.print(" Lista .
    visualizarNodos(12);
    12.removeLast();
    System.out.print(" Lista
    visualizarNodos(12);
    12.addToRear(9);
     12.addToRear(6);
    12.addToRear(20);
    **/
```

Figure 22: Pruebas de removeFirst

4.7 Método borrar un elemento concreto de la lista Casos de prueba

l.inicial	elem	l.final	resultado
[]	X	[]	"mensaje"
[y,z,e]	X	[y,z,e]	"mensaje"
[x]	X	[]	X
[x,z,e]	X	[z,e]	X
[y,x,e]	X	[y,e]	X
[y,z,x]	X	[y,z]	X

El coste del metodo remove es de O(n)

ya que se suma su propio coste O(n/2) con el coste de esta EnPos que es tambien de O(n/2) y al simplificar lo da como resultado el coste de O(n)

Figure 23: Código de Remove

Figure 24: Pruebas de Remove

```
public static void visualizarNodos <a href="mailto:cterminated">cterminated</a> PruebaDoubleLinkedList [Java Application
    Iterator<String> it = 1.iterat
System.out.println("");
                                       Lista .....
    while (it.hasNext()) {
        String k = it.next();
                                      20
         System.out.println(k);
                                      no hay nada que borrar,o no exite elem
public static void main(String[] a
UnorderedDoubleLinkedList<Integer>
//l2.addToFront(7);
12.addToRear(9);
12.addToRear(6);
12.addToRear(20);
System.out.print(" Lista ......
visualizarNodos(l2);
12.remove(8);
System.out.print(" Lista ......
visualizarNodos(l2);
```

Figure 25: Pruebas de Remove

```
public static void visualizarNodos <terminated> PruebaDoubleLink
    Iterator<String> it = 1.iterat Lista .....
    System.out.println("");
                                   9
    while (it.hasNext()) {
                                   6
        String k = it.next();
                                   20
         System.out.println(k);
                                   te he borrado
                                   Lista .....
 public static void main(String[] a
UnorderedDoubleLinkedList<Integer>
 //l2.addToFront(7);
 12.addToRear(9);
 12.addToRear(6);
 12.addToRear(20);
System.out.print(" Lista ......
 visualizarNodos(12);
12.remove(9);
 System.out.print(" Lista ......
 visualizarNodos(12);
```

Figure 26: Pruebas de Remove

```
public static void visualizarNodos <terminated> PruebaDoubleLir
    Iterator<String> it = 1.iterat
                                   Lista .....
    System.out.println("");
    while (it.hasNext()) {
       String k = it.next();
        System.out.println(k);
                                  Lista .....
    }
                                  20
public static void main(String[] a
UnorderedDoubleLinkedList<Integer>
//l2.addToFront(7);
12.addToRear(9);
12.addToRear(6);
12.addToRear(20);
System.out.print(" Lista ......
visualizarNodos(12);
12.remove(6);
System.out.print(" Lista ......
visualizarNodos(12);
```

Figure 27: Pruebas de Remove

```
public static void visualizarNodos <terminated> PruebaDoubleLin
   Iterator<String> it = 1.iterat
System.out.println("");
                                    Lista .....
                                    9
    while (it.hasNext()) {
                                    6
        String k = it.next();
                                    20
        System.out.println(k);
                                    Lista .....
public static void main(String[] a
UnorderedDoubleLinkedList<Integer>
//l2.addToFront(7);
12.addToRear(9);
12.addToRear(6);
12.addToRear(20);
System.out.print(" Lista ......
visualizarNodos(12);
12.remove(20);
System.out.print(" Lista ......
visualizarNodos(12);
```

Figure 28: Pruebas de Remove

4.8 Método buscar

Casos de prueba

l.inicial	elem	resultado
[]	X	null
[y,z,e]	X	null
[x]	X	X
[x,z,e]	X	X
[y,x,e]	X	X
[y,z,x]	X	X

El coste del método find es de O(n/2)

```
public T find(T elem) {
    // Determina si la lista contiene un elemento concreto, y develve su
    // referencia, null en caso de que no esté
    // COMPLETAR EL CODIGO Y CALCULAR EL COSTE

if (isEmpty())
    return null;

Node<T> actual = first; // Empieza con el segundo elemento

while ((actual != null) && !elem.equals(actual.data))
    actual = actual.next;
if (actual == null)
    return null;
else
    return actual.data;
}
```

Figure 29: Código de find

```
23⊝
          public static void main(String[] args) {
 24
               UnorderedDoubleLinkedList<Integer> 12 = new UnorderedDoubleLinkedList<Int
 25
               System.out.println("" + 12.find(2));
 26
               12.addToRear(9);
               System.out.println("" + 12.find(9));
 27
 28
               12.addToRear(6);
               12.addToRear(20);
 29
 30
               System.out.println("" + 12.find(8));
               System.out.println("" + 12.find(9));
 31
               System.out.println("" + 12.find(6));
System.out.println("" + 12.find(20));
 32
 33

    × ¾ | B<sub>1</sub> M P P

🧖 Problems 🔞 Javadoc 🗟 Declaration 📮 Console 🔀
<terminated> PruebaDoubleLinkedList (1) [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (2/11/
null
9
null
9
6
20
```

Figure 30: Pruebas de find

4.9 Cambios Realizados en Laboratorio 1

```
□ import java.util.Iterator;
  public class Actor implements Comparable<Actor> {
      private String nombre;
      private UnorderedDoubleLinkedList<String> ListaPe;///esto hemos cambiado
      public Actor(String o) {
\odot
          this.nombre = o;
          ListaPe = new UnorderedDoubleLinkedList<String>();
\Theta
      public String getNombre() {
          return nombre;
\Theta
      public void setNombre(String nombre) {
          this.nombre = nombre;
\Theta
      public LisPeli getListaP() {
Į.
          return null;
      public UnorderedDoubleLinkedList<String> getDouble() {
\Theta
3
          return this.ListaPe;
\Theta
      @Override
)
      public int compareTo(Actor A) {
          // TODO Auto-generated method stub
          return this.nombre.compareTo(A.nombre);
      }
```

Figure 31: Cambios laboratorio 1

```
public void CargarDatosFichero() throws IOException {
          Stopwatch timer = new Stopwatch();
          String cadena;
          Scanner entrada = new Scanner(new FileReader(
                    "src/fichero/actors-small.txt"));
          while (entrada.hasNext()) {
               cadena = entrada.nextLine();
               String f[] = cadena.split(" ### ");
String Act = f[0];
               CatalogoA.anadir(Act);
for (int i = 1; i < f.length; i++) {
    this.CatalogoP.anadir(f[i]);</pre>
                    if(!CatalogoA.buscarActor(Act).getDouble().contains(f[i]))///-->Esto hemos cambiado
                    CatalogoA.buscarActor(Act).getDouble().addToRear(f[i]);///->Esto tambien hemos cambiado
                                                                                                 № The word 'Esto' is not correctly spelled
               }
                                                                                                 5 quick fixes available:
          this.cargarActoresPorPeli();
                                                                                                   Change to 'East'
          entrada.close();
System.out.println(timer.elapsedTime());
                                                                                                   A Change to 'West'

▼ Disable spell checking

///Esto tambien hemos cambiado

☒ Ignore 'Esto' during the current session

    public void cargarActoresPorPeli() throws IOException {
                                                                                                   ♣ Add 'Esto' to dictionary
          for (Entry<String, Actor> unActor : CatalogoA.getLista().entrySet())
                                                                                                                              Press 'F2' for foc
               UnorderedDoubleLinkedList<String> listaP = unActor.getValue()
              .getDouble();
Iterator<String> it = listaP.iterator();
System.out.println("");
while (it.hasNext()) {
   String unPeli = it.next();
}
                    CatalogoP.getLp().get(unPeli).getLa()
.anadir(unActor.getValue().getNombre());
          }
    }
```

Figure 32: Cambios laboratorio 1

```
static String pNombre;
static int botton;
                                                                                               X ¾ B I F F T
                                           Lanzador (2) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_20\bin\javaw.exe (30/10/2015 14:53:23)
                                            ***************************
public static void main(String[] args
                                           // TODO Auto-generated method stu
                                            - Opcion 1: Busqueda de un actor/actriz
    try {
                                            - Opcion 2: Insercion de nuevo actor/actriz
                                           - Opcion 3: Devolver las peliculas de un actor dado
- Opcion 4: Devolver los actores de una pelicula dada
         Gestor.getGestor().CargarDatc
                                            - Opcion 5: Incrementar el dinero recaudado por una pelicula en un valor dado
                                            - Opcion 6: Borrado de un@ actor/actriz
                                            - Opcion 7: Guardar lista en fichero
                                           - Opcion 8: Obtener una lista de actores ordenada
    } catch (IOException e) {
                                           - Opcion 9: salir del menu.
        // TODO Auto-generated catch
e.printStackTrace();
                                           Introduce un actor para mostrar su lista de peliculas:
                                           Brock, Shanna
Actor Brock, Shanna
    BufferedReader sc = new BufferedF
                                           Me gusta cuando callas
    System.out
                                           Daniel & Ana
             .println("Se va a cargar
                                           Lo que podriamos ser
Toda la Suerte del Mundo
    Gestor.getGestor().CargarDatosFic
    System.out
                                           Detras del Poder
             .println("***********
                                           Mirada de mujer: El regreso
Dos chicos de cuidado en la ciudad
    System.out
             .println("XDXDXDXDXDXDXDXDX
                                           La hija del jardinero
    System.out
                                           El otro Diego
             .println("***********
    System.out.println("- Opcion 1: E
System.out.println("- Opcion 2: ]
                                           Efectos secundarios
                                           El guapo
Cada quien su santo
    System.out
             .println("- Opcion 3: Dev
    System.out
              .println("- Opcion 4: Devolver los actores de una pelicula dada");
    System.out
    .println("- Opcion 5: Incrementar el dinero recaudado por una pelicula en un valor dado");
System.out.println("- Opcion 6: Borrado de un@ actor/actriz");
System.out.println("- Opcion 7: Guardar lista en fichero");
```

Figure 33: Prueba de cargar

5 Conclusiones

Como resultado de los cambios podemos decir que el coste de añadir y borrar en una lista doblemente enlazada es igual de eficiente que las estructura que habíamos utilizado en un principio (tablas hash)

Durante la realización de este laboratorio nos han surgido las siguientes dificultades: No actualizar correctamente los punteros first y last. La utilización del compareTo en el método add de ListaOrdenada, pero en la medida de lo posible lo hemos salvaguardado

En un futuro el método cargar datos se podra hacer mas eficiente en cuanto al espacio utilizado en memoria y se podrian guardar referencias en vez de strings. Esto es una idea teórica y habría que comprobarlo

6 correcciones laboratorio 1

Comentario Koldo: El diagrama de clases está incompleto. En el fichero pdf entregado, se ven flechas que aparecen en la parte de abajo del gráfico, por lo que parece que hay clases que no se ven. Comentario Koldo: ?Coste: el

algoritmo, en el caso peor ,puede que el coste sea O(n),Pero al ser hashMap nos presentan algoritmos de búsqueda muy rápidas y normalmente su coste suele ser O(1).? Al volver a analizar el coste nos retratamos y nos damos

cuenta que el coste es O(1) gracias a la estructura hash.

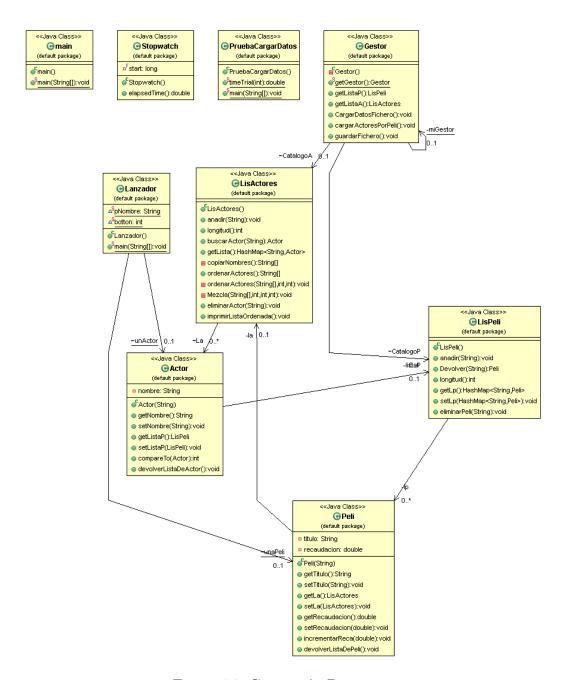


Figure 34: Corrección Diagrama

Comentario Koldo: En la página 4 de la memoria (buscarActor) no aparece el algoritmo. El coste solo se puede calcular si se ve el algoritmo. Si no hay que creérselo, y eso no vale.

Lista iniciai	anorea	Resultao
	"andrea"	"lista vacia"
["pepe","mario"]	"andrea"	Null
["ana","karen","joe"]	"andrea"	Null
["andrea", "karen", "joe"]	"andrea"	Actor entero("andrea", Actor)
["karen","andrea","joe"]	"andrea"	Actor entero("andrea", Actor)
["joe","karen","andrea"]	"andrea"	Actor entero("andrea", Actor)
		•

Coste: el algoritmo, en el caso peor ,puede que el coste sea O(n),Pero al ser hashMap nos presentan algoritmos de búsqueda muy rápidas y normalmente su coste suele ser O(1).

Figure 35: Código buscar

```
public Actor buscarActor(String A)
{
    Actor unActor = null;
    if (!this.La.isEmpty()) {
        unActor = this.La.get(A);
    }
    return unActor;
}
```

Figure 36: Código buscar

4.2 Método ordenarActores

public String[] ordenarActores() {

Precondición: La lista contendrá al menos un actor, y estará desordenada Postcondición: el array estará ordenado alfabéticamente

He utilizado el algoritmo de ordenación mergeSort() por <u>John Von Neumann</u>

- 1. Si la longitud de la lista es 0 ó 1, entonces ya está ordenada. En otro caso:
- 2. Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.
- $3. \ \ Ordenar \ cada \ sublista \ \underline{recursivamente} \ aplicando \ el \ ordenamiento \ por \ mezcla.$
- 4. Mezclar las dos sublistas en una sola lista ordenada.

Casos Prueba:

Lista Inicial	Lista Final
[a]	[a]
[b,a,c,d,e]	[a,b,c,d,e]
[e,b,d,c,a]	[a,b,c,d,e]

La primera partición tendrá un coste O(n),,luego n/2...n/4 ..etc

En el caso peor, medio y mejor este algoritmo tendrá un coste :O(n.logn)

Figure 37: Código ordenar

```
private void Mezcla(String[] actores, int inicio, int centro, int fin) {
    String[] laMezcla = new String[fin - inicio + 1];
    int izq = inicio;
    int der = centro + 1;
    int k = 0;
    while (izq <= centro && der <= fin) {
        if (actores[izq].compareTo(actores[der]) <= 0) {</pre>
            laMezcla[k] = actores[izq];
             k++;
             izq++;
        } else {
             laMezcla[k] = actores[der];
            k++;
            der++;
    if (izq > centro) {
   while (der <= fin) {</pre>
            laMezcla[k] = actores[der];
            k++;
            der++;
    } else {
        while (izq <= centro) {
            laMezcla[k] = actores[izq];
            k++;
            izq++;
        }
    for (int j = inicio; j <= fin; j++) {</pre>
        actores[j] = laMezcla[j - inicio];
}
```

Figure 38: Código mezclar

```
public String[] ordenarActores() {
    String[] actores = this.copiarNombres();
    ordenarActores(actores, 0, actores.length - 1);
    return actores;
}

private void ordenarActores(String[] actores, int inicio, int fin) {
    if (inicio < fin) {
        ordenarActores(actores, inicio, ((inicio + fin) / 2));
        ordenarActores(actores, ((inicio + fin) / 2) + 1, fin);
        Mezcla(actores, inicio, ((inicio + fin) / 2), fin);
    }
}</pre>
```

Figure 39: Código ordenar