eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Bilboko Ingeniaritza Eskola

Kudeaketaren eta Informazio Sistemen Informatikaren Ingeniaritzako Gradua

Aktoreak eta pelikulak kudeatu (2. eginkizuna)

Egileak:
Adei Arias
Jon Barbero
Ander Prieto

Arloa:
Datu-Egiturak eta Algoritmoak
2. maila
46. taldea
1. lauhilabetea

Aurkibidea

1. Sarrera eta arazoaren aurkezpena

Datu-Egiturak eta Algoritmoak ikasgaieko proiektua aktore eta pelikulen kudeaketa egitea da.

Ikasgai honetan, garrantzitsua da programaren kostua. Horretarako, hasiera-hasieratik azpimarratu dugu zer den kostua eta nola zeregin berdin baterako hainbat inplementazio ezberdin dagoen.

Beraz, hau argi ikusteko, hainbat eginkizun bete beharko ditugu lauhilabetean zehar.

Bigarren eginkizun[1] honetan, datu egitura berri bat ikasi eta inplementatuko dugu: *DoubleLinkedList*.

Laborategi honek bi helburu nagusi dituela esan daiteke. Lehenik eta behin, lista estekatuak ondo inplementatzen ikastea da. Bestalde, lista hauek aurreko laborategiko *ArrayList*-engatik ordezkatzea.

Programa honek hainbat ekintza bideratuko ditu: Lista estekatu batean elementuak era ordenatuan txertatzea, elementuak bai hasiera eta bai amaieran txertatzea, elementu baten bilaketa, etab.

Printzipioz, hiru klase izango ditugu; *DoubleLinkedList*, *UnorderedDoubleLinkedList* eta *OrderedDoubleLinkedList*. Baina, lehen esan den bezala, lista estekatu hauek aurreko laborategiko lista bat edo gehiagotan ordezkatu beharko ditugu. Honen ondorioz, aldatutako klaseak jarri beharko ditugu. Gure kasuan, *ArrayPelikulak* eta *ArrayAktoreak* izan dira. Guztira 5 klase izango ditugu.

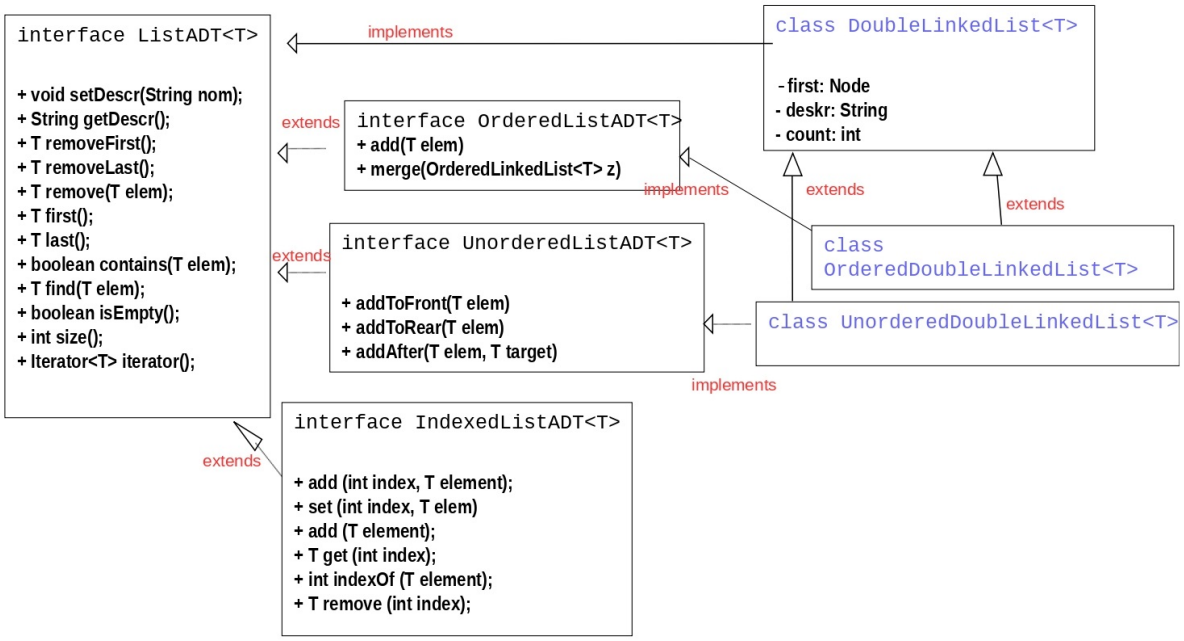
Berriz ere, Eclipse gure "dantzarako bikote" izango dugu programatzeko momentuan, baita LaTeX ere lortutako kodea, emaitzak eta hauen erreferentziak idazteko eta islatzeko.

2. Diseinua

Bigarren eginkizun honetarako, lehenengo zatiaren klase diagrama enuntziatuan zegoen (1. irudia).

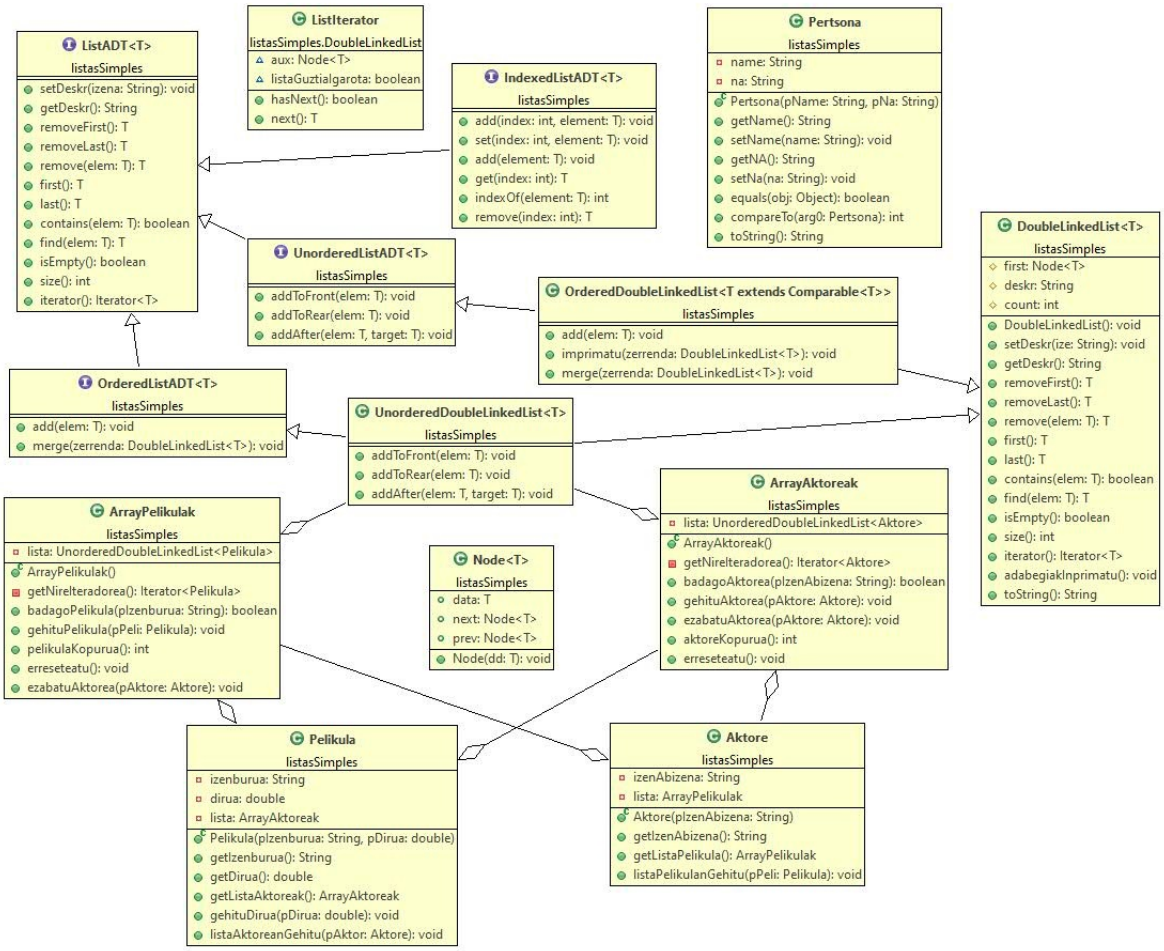
Irudian ikus dezakegunez, lista egituraren azpiklaseak agertzen dira.

Klase hauek bigarren zatian erabiliko ditugu, oraingo eginkizuna klase horietako metodoak inplementatzea izan da; metodo batzuk komunean daude (*+ T remove(T elem)*, adibidez), eta beste batzuk klase bakoitzean inplementatu beharko dira (esaterako, *+ add(T elem)*)



1. irudia: Enuntziatuan emandako diseinua

Bigarren zatia lehenengo eginkizuneko *ArrayList* guztiak *DoubleLinkedList*-en ordez aldatzean datza. Horretarako, *ArrayList*-ei deiak aldatu ditugu, bai eta *ArrayAktore* eta *ArrayPelikulak* klaseetan dauden metodoak egokitu (2. irudia).



2. irudia: Aldaketaren ondoren geratzen den diseinua

3. Datu egituren diseinua

Datu egitura mota bakarra erabili dugu eginkizun honetan: *DoubleLinkedList*-a (zuzenagoak izateko, *CircularDoubleLinkedList*)

Datu egitura mota honen berezitasuna nodoak aurreko eta atzean duen nodoekin lotuta daude. Elementuak txertatu edo ezabatzerakoan *ArrayList*-ak baino askoz ere eraginkorrak dira. Izan ere, *ArrayList* baten elementu bat ezabatzerakoan, atzetik dauden elementuak toki bat ezkerrera mugitu behar dira; lehenengoa ezabatu behar bada, beste elementu guztiak mugitu beharko dira *O(n)*-ko kostua izanik. *(Circular)DoubleLinkedList* batean, aldiz, kostua *O(1)* da, soilik 4-5 agindu bete behar direlako.

Bukatzeko, *DoubleLinkedList*-a lehenengo eginkizunean erabiliko dugunez, bertako *HashMap*[2] egitura ere izango dugu.

4. Metodo nagusien diseinu eta inplementazioa

4.1 DoubleLinkedList

4.1.1 Lehenengo elementua kendu

public T removeFirst():

// Aurre: Zerrenda ez da hutsa

// Post: Elementua kenduko da eta return baten bidez itzuliko da

- Proba kasuak:

Zerrenda	**Emaitza**
(a)	()
(a b c)	(b c)

- Kostua: $O(1)$

4.1.2 Azkenengo elementua kendu

public T removeLast():

// Aurre: Zerrenda ez da hutsa

// Post: Elementua kenduko da eta return baten bidez itzuliko da

- Proba kasuak:

Zerrenda	**Emaitza**
(a)	()
(a b c)	(a b)

- Kostua: $O(1)$

4.1.3 Elementua kendu

public T remove(T elem):

// Aurre: Zerrenda ez da hutsa

// Post: Elementua kenduko da eta return baten bidez itzuliko da

- Proba kasuak:

Zerrenda	**T elem**	**Emaitza**
(a b)	c	null
(a)	a	()
(a b c d)	c	(a b d)
(a b)	a	(b)
(a b c)	c	(a b)

- Kostua: $O(n)$ // *n: zeharkatu beharreko nodoak*

4.1.4 Lehenengo elementua lortu

public T first():

// Aurre: --

// Post: Lehenengo elementua lortuko da

- Proba kasuak:

Zerrenda	**Emaitza**
()	null
(a b c)	a

- Kostua: $O(1)$

4.1.5 Azkenengo elementua lortu

public T last():

// Aurre: --

// Post: Azkenengo elementua lortuko da

- Proba kasuak:

Zerrenda	**Emaitza**
()	null
(a b c)	a

- Kostua: $O(1)$

4.1.6 Badago elementua

public boolean contains(T elem):

// Aurre: Zerrenda ez da hutsa

// Post: elem elementua bilatuko da eta true edo false itzuliko du

- Proba kasuak:

Zerrenda	**T elem**	**Emaitza**
()	c	false
(a)	a	true
(a b c d)	c	true
(a b c)	e	false

- Kostua: $O(n)$ *// n: zeharkatu beharreko nodoak*

4.1.7 Bilatu elementua

public T find(T elem):

// Aurre: Zerrenda ez da hutsa

// Post: elem elementua bilatuko da eta true edo false itzuliko du

- Proba kasuak:

Zerrenda	**T elem**	**Emaitza**
()	c	null
(a)	a	a
(a b c d)	c	c
(a b c)	e	null

- Kostua: $O(n)$ // *n: zeharkatu beharreko nodoak*

4.2 UnorderedDoubleLinkedList

4.2.1 Gehitu elementua hasieran

public T addToFront(T elem):

// Aurre: --

// Post: elem elementua listaren hasieran gehituko da

- Proba kasuak:

Zerrenda	**T elem**	**Emaitza**
()	a	(a)
(a b c d)	e	(e a b c d)

- Kostua: $O(1)$

4.2.2 Gehitu elementua bukaeran

public T addToRear(T elem):

// Aurre: --

// Post: elem elementua listaren amaieran gehituko da

- Proba kasuak:

Zerrenda	**T elem**	**Emaitza**
()	a	(a)
(a b c d)	e	(a b c d e)

- Kostua: $O(1)$

4.2.3 Gehitu elementua beste elementu baten ondoren

public void addAfter(T elem, T target):

// Aurre: --

// Post: elem elementua target elementuaren atzean gehituko du

- Proba kasuak:

Zerrenda	**T target**	**T elem**	**Emaitza**
()	null	a	()
(a b c d)	e	f	(a b c d)
(a b c d)	d	e	(a b c d e)
(a b c d)	c	e	(a b c e d)

- Kostua: $O(n)$ // *n: zeharkatu beharreko nodoak*

4.3 OrderedDoubleLinkedList

4.3.1 Gehitu elementua ordenean

public void add(T elem):

// Aurre: Elementua ezin da null izan

// Post: elem elementua modu ordenatuan gehituko da

- Proba kasuak:

Zerrenda	T elem	Emaitza
()	a	(a)
(a b c d)	e ; (e<a)	(e a b c d)
(a b c d)	e ; (e>d)	(a b c d e)
(a b c d)	e ; (c<e<d)	(a b c e d)

- Kostua: $O(n)$ // *n: zeharkatu beharreko nodoak*

4.3.2 Bi lista ordenean batu

public void merge(DoubleLinkedList<T>zerrenda):

// Aurre: --

// Post: Zerrendak batuko dira

- Proba kasuak:

Zerrenda	DoubleLinkedList<T>zerrenda	Emaitza
()	(a b c d)	(a b c d)
()	()	()
(a b c)	(d e f) ; (a<d<b<e<f<c)	(a d b e f c)

- Algoritmoa:

```
OrderedDoubleLinkedList<T> listaB = new OrderedDoubleLinkedList<T>();
Node<T> firstB = null;
while(bi listetako bat igaro arte egongo da while hau){
   if(Lehenengo listaren elementua, bigarrenaren handiago bada){
      if(lista berria == null){
         nodoa sortu lista berrian txertatu
      } else {
         nodoa sortu lista berriaren bukaeran txertatu
      }
      if(bigarren lista guztiz zeharkatu bada){
         lista2Igarota = true;
      }
      bigarren lista iteratu
```

```
   } else if(lehenengo listaren elementua, bigarrenaren berdina bada){
      if(lista berria == null){
         nodoa sortu lista berrian txertatu
      } else {
         nodoa sortu eta lista berriaren bukaeran txertatu
      }
      if(lehenengo lista guztiz zeharkatu bada){
         lista1Igarota = true;
      } else if(bigarren lista guztiz zeharkatu bada){
         lista2Igarota = true;
      } else {}
      lehenengo eta bigarren lista iteratu
   } else {
      if(lista berria == null){
         nodoa sortu eta listan txertatu
      } else {
         nodoa sortu eta lista berriaren bukaeran txertatu
         }
      lehenengo lista iteratu
      if(lehenengo lista guztiz zeharkatu bada){
         lista1Igarota = true;
      }
   }
if(lehenengo lista zeharkatu bada, baina bigarrena ez){
   while(bigarren lista igaro arte){
      nodoa sortu eta listaren bukaeran elementua txertatu
      bigarren lista iteratu
   }
} else if(bigarren lista zeharkatu bada, baina lehenengoa ez){
   while(lehenengo lista igaro arte){
      nodoa sortu eta listaren bukaeran elementua txertatu
      lehenengo lista iteratu
   }
}
} else{}
listaB.first = firstB;
listaB.adabegiakInprimatu();
```

- Kostua: $O(n)$ // *n: zeharkatu beharreko nodoak*

5. Kodea

5.1 Lehen eginkizuneko kode berria

5.1.1 ArrayPelikulak.java

```
1  package aktorePelikulaPackage;
2  import java.util.ArrayList;
3  import java.util.Iterator;
4
5  public class ArrayPelikulak {
6
7     private UnorderedDoubleLinkedList<Pelikula> lista;
8
9     public ArrayPelikulak(){
10       this.lista = new UnorderedDoubleLinkedList<Pelikula>();
11    }
12
13    private Iterator<Pelikula> getNireIteradorea(){
14       return this.lista.iterator();
15    }
16
17    public boolean badagoPelikula(String pIzenburua){//Pelikula bat emanda, listan
      ↪  dagoen esango digu
18       boolean dago = false;
19       Pelikula pelikula = null;
20       Iterator<Pelikula> itr = this.getNireIteradorea();
21       while(itr.hasNext() && !dago){
22          pelikula = itr.next();
23          if(pelikula.getIzenburua().equals(pIzenburua)){
24             dago = true;
25          }
26       }
27       return dago;
28    }
29
30    public void gehituPelikula(Pelikula pPeli){
31          this.lista.addToRear(pPeli);
32    }
33
34    public int pelikulaKopurua() {
35       return this.lista.size();
```

```
36      }
37
38      public void erreseteatu() {
39          this.lista.first = null;
40          this.lista.count=0;
41
42      }
43
44      public void ezabatuAktorea(Aktore pAktore) {
45          Pelikula peli = null;
46          ArrayAktoreak lista=null;
47          Iterator<Pelikula> itr = this.getNireIteradorea();
48          while(itr.hasNext()) {
49              peli = itr.next();
50              lista=peli.getListaAktoreak();
51              lista.ezabatuAktorea(pAktore);
52          }
53      }
54  }
```

5.1.2 ArrayAktoreak.java

```
1   package aktorePelikulaPackage;
2   import java.util.*;
3
4   public class ArrayAktoreak {
5
6       private UnorderedDoubleLinkedList<Aktore> lista;
7
8       public ArrayAktoreak(){
9           this.lista = new UnorderedDoubleLinkedList<Aktore>();
10      }
11
12      private Iterator<Aktore> getNireIteradorea(){
13          return this.lista.iterator();
```

```
14      }
15
16      public boolean badagoAktorea(String pIzenAbizena){//Aktore bat pasata, listan
        ↪  dagoen esango digu
17          boolean dago = false;
18          Aktore aktor = null;
19          Iterator<Aktore> itr = this.getNireIteradorea();
20          while(itr.hasNext() && !dago){
21              aktor = itr.next();
22              if(aktor.getIzenAbizena().equals(pIzenAbizena)){
23                  dago = true;}
24          }
25          return dago;
26      }
27
28      public void gehituAktorea(Aktore pAktore){
29          if(!(this.badagoAktorea(pAktore.getIzenAbizena()))){
30              this.lista.addToRear(pAktore);
31          }
32      }
33
34      public void ezabatuAktorea(Aktore pAktore) {
35          this.lista.remove(pAktore);
36      }
37
38      public int aktoreKopurua() {
39          return this.lista.size();
40      }
41
42      public void erreseteatu() {
43          this.lista.first=null;
44          this.lista.count=0;
45      }
46  }
```

5.2 Interfazeak

5.2.1 ListADT.java

```java
package aktorePelikulaPackage;

import java.util.Iterator;

public interface ListADT<T> {


public void setDeskr(String izena); // Listaren izena eguneratzen du

public String getDeskr(); // Listaren izena bueltatzen du


public T removeFirst(); // listako lehen elementua kendu da

public T removeLast(); // listako azken elementua kendu da

public T remove(T elem); // Balio hori listan baldin badago, bere lehen agerpena ezabatuko dut. Kendutako objektuaren erreferentzia bueltatuko du (null ez baldin badago)

public T first(); // listako lehen elementua ematen du

public T last(); // listako azken elementua ematen du

public boolean contains(T elem); // Egiazkoa bueltatuko du aurkituz gero, eta false bestela

public T find(T elem); // Elementua bueltatuko du aurkituz gero, eta null bestela

public boolean isEmpty(); // hutsa den esango du

public int size(); // Listako elementu-kopurua

public Iterator<T> iterator(); // Listako elementuen iteradorea


}
```

5.2.2 *IndexedListADT.java*

```
1  /**
2   * IndexedListADT defines the interface to an indexed list collection. Elements
3   * are referenced by contiguous numeric indexes.
4   * @author Dr. Lewis
5   * @author Dr. Chase
6   * @version 1.0, 08/13/08
7   */
8
9  package aktorePelikulaPackage;
10
11 public interface IndexedListADT<T> extends ListADT<T>
12 {
13    /**
14     * Inserts the specified element at the specified index.
15     *
16     * @param index   the index into the array to which the element is to be
17     *                inserted.
18     * @param element the element to be inserted into the array
19     */
20    public void add (int index, T element);
21
22    /**
23     * Sets the element at the specified index.
24     *
25     * @param index   the index into the array to which the element is to be set
26     * @param element the element to be set into the list
27     */
28    public void set (int index, T element);
29
30    /**
31     * Adds the specified element to the rear of this list.
32     *
33     * @param element  the element to be added to the rear of the list
34     */
35    public void add (T element);
36
37    /**
38     * Returns a reference to the element at the specified index.
39     *
```

```
40      * @param index  the index to which the reference is to be retrieved from
41      * @return       the element at the specified index
42      */
43     public T get (int index);
44
45     /**
46      * Returns the index of the specified element.
47      *
48      * @param element  the element for the index is to be retrieved
49      * @return         the integer index for this element
50      */
51     public int indexOf (T element);
52
53     /**  Removes and returns the element at the specified index. */
54     public T remove (int index);
55 }
```

5.2.3 *OrderedListADT.java*

```
1  package aktorePelikulaPackage;
2
3  public interface OrderedListADT<T>  extends ListADT<T> {
4
5     public void add(T elem);
6     // elementu bat gehitzen du listan (dagokion tokian)
7
8     public void merge(DoubleLinkedList<T> zerrenda);
9
10 }
```

5.2.4 UnorderedListADT.java

```
1  package aktorePelikulaPackage;
2
3  public interface UnorderedListADT<T> extends ListADT<T> {
4
5     public void addToFront(T elem);
6     // elementu bat gehituko du hasieran
7
8     public void addToRear(T elem);
9     // elementu bat gehituko du bukaeran
10
11    public void addAfter(T elem, T target);
12    // "elem" elementua "target" elementuaren ondoren gehitzen du ("target" listan
      ↪  zegoen)
13
14
15 }
```

5.3 Bigarren eginkizuneko inplementazioak

5.3.1 Node.java

```java
package aktorePelikulaPackage;

   public class Node<T> {
      public T data;           // dato del nodo
      public Node<T> next;     // puntero al siguiente nodo de la lista
      public Node<T> prev;     // puntero al anterior nodo de la lista
      // ----------------------------------------------------------------

      public Node(T dd)        // constructor
      {
         data = dd;
         next = null;
         prev = null;
      }
   }
```

5.3.2 DoubleLinkedList.java

```java
package aktorePelikulaPackage;

import java.util.Iterator;

public class DoubleLinkedList<T> implements ListADT<T> {

   // Atributuak
   protected Node<T> first; // lehenengoaren erreferentzia
   protected String deskr;  // deskribapena
   protected int count;

   public DoubleLinkedList() {
      first = null;
      deskr = "";
```

```
15        count = 0;
16    }
17
18    public void setDeskr(String ize) {
19      deskr = ize;
20    }
21
22    public String getDeskr() {
23      return deskr;
24    }
25
26    public T removeFirst() {
27    // listako lehen elementua kendu da
28    // Aurrebaldintza: zerrenda ez da hutsa
29        Node<T> aux = first;
30        if (first.next==first) { //elementu bakarra
31           first=null;
32
33        }
34        else{
35           first.prev.next=first.next;
36           first.next.prev=first.prev;
37           first=first.next;
38        }
39        --count;
40
41        return aux.data;
42    }
43
44    public T removeLast() {
45    // listako azken elementua kendu da
46    // Aurrebaldintza: zerrenda ez da hutsa
47
48        Node<T> aux = first.prev;
49        if (first.next==first) { //elementu bakarra
50           first=null;
51
52        }
53        else{
54           aux.prev.next=first;
55           first.prev=aux.prev;
56        }
57        --count;
58        return aux.data;
59     }
60
```

```
61
62    public T remove(T elem) {
63    // Aurrebaldintza: zerrenda ez da hutsa
64    // Balio hori listan baldin badago, bere lehen agerpena ezabatuko dut. Kendutako
      ↪  objektuaren erreferentzia
65         //  bueltatuko du (null ez baldin badago)
66       if(!contains(elem)){return null;}//Elementua lista ez badago
67       else{//Elementua lista dago
68          Node<T> aux = first.prev;
69          boolean badago=false;
70          while(!badago){
71             aux=aux.next;
72             if(aux.data.equals(elem)){
73                badago=true;
74                if(aux.next.equals(first)){ return removeLast(); }
75                else if (aux.equals(first)){ return removeFirst(); }
76                else{//listaren erdian dago kendu beharko den elementua
77                   --count;
78                   aux.prev.next=aux.next;
79                   aux.next.prev=aux.prev;
80                   return aux.data;
81                }
82             }
83          }
84       return null; //eclipsek behartuta
85       }
86     }
87
88    public T first() {
89    // listako lehen elementua ematen du
90          if (isEmpty()){
91              return null;
92          }else{ return first.data;}
93    }
94
95    public T last() {
96    // listako azken elementua ematen du
97          if (isEmpty()){
98              return null;}
99          else{ return first.prev.data;}
100   }
101
102   public boolean contains(T elem) {
103   // Egiazkoa bueltatuko du aurkituz gero, eta false bestel
104      if (isEmpty()){
105         return false;}
```

```
106         else{
107            Node<T> aux = first;
108            if (first.data.equals(elem)){return true;}//Lehenengo elementua
109            boolean badago=false;
110            while(!aux.equals(first.prev) && !badago){
111               aux=aux.next;
112               if(aux.data.equals(elem)){
113                  badago=true;
114               }
115            }
116            return badago;
117         }
118      }
119
120      public T find(T elem) {
121      // Elementua bueltatuko du aurkituz gero, eta null bestela
122
123         if (isEmpty()){
124            return null;}
125         else{
126            if (contains(elem)) {
127               //System.out.println("aurkitua");
128               return elem;
129            }
130            else {
131               //System.out.println("ez da aurkitu");
132               return null;
133            }
134         }
135      }
136
137      public boolean isEmpty()
138      { return first == null;};
139
140      public int size()
141      { return count;};
142
143      /** Return an iterator to the stack that iterates through the items . */
144         public Iterator<T> iterator() { return new ListIterator(); }
145         // an iterator, doesn't implement remove() since it's optional
146         private class ListIterator implements Iterator<T> {
147           Node<T> aux = first;
148           boolean listaGuztiaIgarota=false;//lista guztia zeharkatu dugun jakiteko
              ↪ boolean-a
149           public boolean hasNext(){
150              if(isEmpty()){ return false; }
```

```
151             else if (aux.equals(first) && listaGuztiaIgarota) {return false;}
152              else {
153                 listaGuztiaIgarota=true;
154                 return true;
155              }
156           }
157           public T next(){
158              T emaitza = aux.data;
159              aux=aux.next;
160              return emaitza;}
161        } // private class
162
163
164        public void adabegiakInprimatu() {
165           System.out.println(this.toString());
166        }
167
168
169        @Override
170        public String toString() {
171           String result = new String();
172           Iterator<T> it = iterator();
173           while (it.hasNext()) {
174              T elem = it.next();
175              result = result + "[" + elem.toString() + "] \n";
176           }
177           return "\nDoubleLinkedList: \n" + result;
178        }
179
180  }
```

5.3.3 *OrderedDoubleLinkedList.java*

```
1  package aktorePelikulaPackage;
2
3  public class OrderedDoubleLinkedList<T extends Comparable<T>> extends
   ↪  DoubleLinkedList<T> implements OrderedListADT<T> {
4
```

```
5
6     public void add(T elem){
7        Node<T> berria = new Node<T>(elem);
8        boolean aurk = false;//elementuaren posizio egokia aurkitzean aktibatutako
         ↪  boolean-a
9        boolean buelta = false;//boolean hau erabiliko dugu jakiteko ea lista guztitik
         ↪  pasatu garen ala ez, lista zirkularra delako
10       boolean atera = false;//jasotako elementua, jadanik lista balego, ez litzateke
         ↪  listan sartuko eta boolean hau aktibatuko liteke
11       if(first == null){//Lista hutsa bada
12          count++;
13          first = new Node<T>(elem);
14          first.next = first;
15          first.prev = first;
16
17       }else{
18          Node<T> egungoa = first;
19          while(!aurk && !buelta && !atera){
20             if(egungoa.data.compareTo(elem)==1){//elementua listan sartu
21                count++;
22                if(egungoa == first){//hasieran txertatu
23                   berria.next =first;
24                   berria.prev = first.prev;
25                   first.prev.next = berria;
26                   first.prev = berria;
27                   first = berria;
28                }else{//erdialdean txertatu
29                   berria.prev = egungoa.prev;
30                   berria.next = egungoa;
31                   egungoa.prev.next = berria;
32                   egungoa.prev = berria;
33                }
34                aurk = true;
35             }else if(egungoa.data.equals(elem)){//ezer ez txertatu
36                atera = true;
37             }else{//iteratu
38                egungoa = egungoa.next;
39             }
40             if(egungoa == first){
41                buelta = true;
42             }
43          }
44          if(buelta){//bukaeran txertatu
45             count++;
46             berria.prev = egungoa.prev;
47             berria.next = egungoa;
```

```
48              egungoa.prev.next = berria;
49              egungoa.prev = berria;
50          }
51      }
52  }
53
54  public void imprimatu (DoubleLinkedList<T> zerrenda){
55      Node<T>aux=first;
56      if (isEmpty()){
57      }
58      else{
59          while(!aux.equals(first.prev)){
60              System.out.print ( aux.data+ " ");
61              aux=aux.next;
62          }
63      }
64  }
65
66  public void merge(DoubleLinkedList<T> zerrenda){
67      OrderedDoubleLinkedList<T> listaB = new OrderedDoubleLinkedList<T>();
68      Node<T> act1 = null;
69      Node<T> act2 = null;
70      Node<T> act3 = null;
71      Node<T> berria = null;
72      Node<T> firstB = null;
73      act1 = zerrenda.first;
74      act2 = this.first;
75      //BI BOOLEAN JARRIKO DITUGU LISTA OSOA ZEHARKATU DUGUN JAKITEKO, BAT LISTA
        ↪  BAKOITZEKO
76      boolean lista1Igarota = false;
77      boolean lista2Igarota = false;
78      while(!lista1Igarota  && !lista2Igarota){
79          if(act1.data.compareTo(act2.data)==1){//BIGARREN ELEMENTUA LEHENENGO
            ↪  ELEMENTUA BAINO HANDIAGOA BADA
80              if(firstB==null){//BUKAERA LISTA HUTSA BADA
81                  firstB = new Node<T>(act2.data);
82                  firstB.prev = firstB;
83                  firstB.next = firstB;
84                  act2 = act2.next;
85                  act3 = firstB;
86              }else{
87                  berria = new Node<T>(act2.data);//BESTELA, LISTA BERRIKO AZKEN
                    ↪  ELEMENTUAREN ATZETIK JARRIKO DUGU NODO BERRIA
88                  berria.prev = act3;
89                  berria.next = firstB;
90                  firstB.prev = berria;
```

```
91                act3.next = berria;
92                act2 = act2.next;
93                act3 = act3.next;
94             }
95             if(act2 == this.first){//IF HAU ERABILIKO DUGU, BIGARREN LISTA OSOA
               ↪  ZEHARKATU DUGUN JAKITEKO
96                lista2Igarota = true;
97             }
98          }else if(act1.data.compareTo(act2.data)==0){//LEHENENGO ELEMENTUA, BIGARREN
            ↪  ELEMENTUAREN BERDINA BADA
99             //KASU HONETAN BAKARRIK ZENBAKI BAT JARRIKO DUGU(EZ DUGU ZENBAKIRIK
               ↪  ERREPIKATUKO)
100            if(firstB==null){//LISTA HUTSA BALDIN BADA
101               firstB = new Node<T>(act1.data);
102               firstB.prev = firstB;
103               firstB.next = firstB;
104               act1 = act1.next;
105               act2 = act2.next;
106               act3 = firstB;
107            }else{
108               berria = new Node<T>(act1.data);//BESTELA, LISTA BERRIKO AZKEN
                  ↪  ELEMENTUAREN ATZEAN JARRIKO DUGU NODO BERRIA
109               berria.prev = act3;
110               berria.next = firstB;
111               firstB.prev = berria;
112               act3.next = berria;
113               act1 = act1.next;
114               act2 = act2.next;
115               act3 = act3.next;
116            }
117            //KASU HONETAN BI IF JARRIKO DITUGU. AZKEN BATEAN, BI LISTEN ELEMENTUA
               ↪  BERDINA DENEZ,
118            //BI LISTAK ITERATUKO DITUGU ETA BALITEKE LISTA BAT HUTSA EGOTEA
119            if(act1 == zerrenda.first){
120               lista1Igarota = true;
121            }else if(act2 == this.first){
122               lista2Igarota = true;
123            }else{
124            }
125         }else{//LEHENENGO ELEMENTUA, BIGARREN ELEMENTUA BAINO HANDIAGOA BADA
126            if(firstB==null){//LISTA HUTSA
127               firstB = new Node<T>(act1.data);
128               firstB.prev = firstB;
129               firstB.next = firstB;
130               act1 = act1.next;
131               act3 = firstB;
```

```
132             }else{//BESTELA, LISTA BERRIKO AZKEN ELEMENTUAREN ATZEAN JARRIKO DUGU
                ↪  NODO BERRIA
133                berria = new Node<T>(act1.data);
134                berria.prev = act3;
135                berria.next = firstB;
136                firstB.prev = berria;
137                act3.next = berria;
138                act1 = act1.next;
139                act3 = act3.next;
140             }
141             if(act1 == zerrenda.first){//IF HAU ERABILIKO DUGU, LEHENENGO LISTA OSOA
                ↪  ZEHARKATU DUGUN JAKITEKO
142                lista1Igarota = true;
143             }
144          }
145       }
146       //WHILETIK ATERATZEAN, HIRU KASU POSIBLE EGON DAITEZKE:
147       //1.- LEHENENGO LISTA HUTSA EGOTEA
148       //2.- BIGARREN LISTA HUTSA EGOTEA
149       //3.- BI LISTAK HUTSAK EGOTEA
150       if(lista1Igarota && !lista2Igarota){//LEHENENGO KASUAN IZANGO LITZATEKE.
          ↪  LEHENENGO LISTA HUTSA ETA BIGARRENA EZ
151          while(!lista2Igarota){//KASU HONETAN, WHILE HAU ERABILIKO DUGU, BIGARREN
             ↪  LISTAN GELDITZEN DIREN ELEMENTU GUZTIAK, LISTA BERRIAREN ATZEAN
             ↪  TXERTATZEKO
152             berria = new Node<T>(act2.data);
153             berria.prev = act3;
154             berria.next = firstB;
155             firstB.prev = berria;
156             act3.next = berria;
157             act2 = act2.next;
158             act3 = act3.next;
159             if(act2 == this.first){
160                lista2Igarota = true;
161             }
162          }
163       }else if(lista2Igarota && !lista1Igarota){//BIGARREN KASUAN IZANGO LITZATEKE.
          ↪  BIGARREN LISTA HUTSA ETA LEHENENGOA EZ
164          while(!lista1Igarota){//KASU HONETAN, WHILE HAU ERABILIKO DUGU, LEHENENGO
             ↪  LISTAN GELDITZEN DIREN ELEMENTU GUZTIAK, LISTA BERRIAREN ATZEAN
             ↪  TXERTATZEKO
165             berria = new Node<T>(act1.data);
166             berria.prev = act3;
167             berria.next = firstB;
168             firstB.prev = berria;
169             act3.next = berria;
```

```
170             act1 = act1.next;
171             act3 = act3.next;
172             if(act1 == zerrenda.first){
173                lista1Igarota = true;
174             }
175          }
176       }else{//HIRUGARREN KASUAN IZANGO LITZATEKE. BI LISTAK HUTSAK DAUDE
177       }//KASU HONETAN EZ GENUKE EZER EGINGO
178       listaB.first = firstB;
179       listaB.adabegiakInprimatu();
180    }
181
182 }
```

5.3.4 *UnorderedDoubleLinkedList.java*

```
1  package aktorePelikulaPackage;
2
3
4  public class UnorderedDoubleLinkedList<T> extends DoubleLinkedList<T> implements
   ↪  UnorderedListADT<T> {
5
6     public void addToFront(T elem) {
7     // hasieran gehitu
8        Node<T> berria = new Node<T>(elem);
9        if(isEmpty()) {
10          first=berria;
11          berria.next=berria;
12          berria.prev=berria;
13       }
14       else{
15          first.prev.next=berria;
16          berria.prev=first.prev;
17          first.prev=berria;
18          berria.next=first;
19          first=berria;
20       }
21       count++;
```

```
22      }
23
24      public void addToRear(T elem) {
25      // bukaeran gehitu
26          Node<T> berria = new Node<T>(elem);
27          if(isEmpty()) {
28              first=berria;
29              berria.next=berria;
30              berria.prev=berria;
31          }
32          else{
33
34              first.prev.next=berria;
35              berria.prev=first.prev;
36              berria.next=first;
37              first.prev=berria;
38
39          }
40          count++;
41      }
42
43      public void addAfter(T elem, T target) {
44          Node<T> berria = new Node<T>(elem);
45          Node<T> egungo = first;
46          if(isEmpty()) {//Lista hutsa bada
47              first=berria;
48              berria.next=berria;
49              berria.prev=berria;
50              count++;
51          }
52          else{
53                boolean aurk = false;
54                while(!aurk){
55                   if(egungo.data.equals(target)){
56                      aurk = true;
57                   }else{
58                      egungo = egungo.next;
59                   }
60                }
61                //Momentu honetan elementua sartzeko posizio egokia aurkitu dugu eta
                  ↪  elementua txertatuko dugu
62                berria.next=egungo.next;
63                berria.prev = egungo;
64                egungo.next.prev = berria;
65                egungo.next = berria;
66                count++;
```

```
67          }
68      }
69  }
```

5.3.5 *Pertsona.java*

```
1   package aktorePelikulaPackage;
2
3   public class Pertsona implements Comparable<Pertsona> {
4
5       // atributuak
6       private String name;
7            private String na;
8
9       public Pertsona(String pName, String pNa) { // Eraikitzailea
10          name = pName;
11          na = pNa;
12      }
13
14      public String getName() { return name; }
15
16      public void setName(String name) { this.name = name; }
17
18      public String getNA() { return this.na; }
19
20      public void setNa(String na) { this.na = na; }
21
22      @Override
23      public boolean equals(Object obj) {
24          if (this == obj)
25              return true;
26          if (obj == null)
27              return false;
28          if (getClass() != obj.getClass())
29              return false;
30          Pertsona other = (Pertsona) obj;
31          if (na == null) {
32              if (other.na != null)
```

```
33              return false;
34          } else if (!na.equals(other.na))
35             return false;
36          return true;
37       }
38
39       @Override
40       public int compareTo(Pertsona arg0) {
41          return name.compareToIgnoreCase(arg0.name);
42       }
43
44       public String toString() {
45          return name + " " + na;
46       }
47
48    } // end Pertsona
```

6. JUnitak

6.1 Lista estekatuen JUnitak

6.1.1 TestDoubleLinkedList.java

```
1  package aktorePelikulaPackage;
2
3  import static org.junit.Assert.*;
4
5  import java.util.Iterator;
6
7  import org.junit.After;
8  import org.junit.Before;
9  import org.junit.Test;
10
11 public class TestDoubleLinkedList {
12    UnorderedDoubleLinkedList<Integer> listaZenbaki1;
13    UnorderedDoubleLinkedList<Integer> listaZenbaki2;
14
15    @Before
16    public void setUp() throws Exception {
17       listaZenbaki1 = new UnorderedDoubleLinkedList<Integer>();
18       listaZenbaki2 = new UnorderedDoubleLinkedList<Integer>();
19    }
20
21    @After
22    public void tearDown() throws Exception {
23    }
24
25    @Test
26    public void testSetDeskr() {
27       listaZenbaki1.setDeskr("lista1");
28       assertNotNull(listaZenbaki1.getDeskr());
29    }
30
31    @Test
32    public void testGetDeskr() {
33       listaZenbaki1.setDeskr("lista1");
34       assertEquals(listaZenbaki1.getDeskr(),"lista1");
35    }
36
```

```
37      @Test
38      public void testRemoveFirst() {
39          //elementu asko
40          listaZenbaki1.addToRear(1);
41          listaZenbaki1.addToRear(2);
42          listaZenbaki1.addToRear(3);
43          listaZenbaki1.addToRear(4);
44          listaZenbaki1.addToRear(5);
45          listaZenbaki1.addToRear(6);
46          listaZenbaki1.addToRear(7);
47          listaZenbaki1.removeFirst();
48          assertEquals(listaZenbaki1.first.data,new Integer(2));
49          //elementu bakarra
50          listaZenbaki2.addToRear(1);
51          listaZenbaki2.removeFirst();
52          assertNull(listaZenbaki2.first);
53      }
54
55      @Test
56      public void testRemoveLast() {
57          //elementu asko
58          listaZenbaki1.addToRear(1);
59          listaZenbaki1.addToRear(2);
60          listaZenbaki1.addToRear(3);
61          listaZenbaki1.addToRear(4);
62          listaZenbaki1.addToRear(5);
63          listaZenbaki1.addToRear(6);
64          listaZenbaki1.addToRear(7);
65          listaZenbaki1.removeLast();
66          assertEquals(listaZenbaki1.first.prev.data,new Integer(6));
67          //elementu bakarra
68          listaZenbaki2.addToRear(1);
69          listaZenbaki2.removeLast();
70          assertNull(listaZenbaki2.first);
71      }
72
73      @Test
74      public void testRemove() {
75          //elementu asko
76          listaZenbaki1.addToRear(1);
77          listaZenbaki1.addToRear(2);
78          listaZenbaki1.addToRear(3);
79          listaZenbaki1.addToRear(4);
80          listaZenbaki1.addToRear(5);
81          listaZenbaki1.addToRear(6);
82          listaZenbaki1.addToRear(7);
```

```
83          listaZenbaki1.remove(5);
84          assertFalse(listaZenbaki1.contains(5));
85          //elementu bakarra
86          listaZenbaki2.addToRear(1);
87          listaZenbaki2.remove(1);
88          assertFalse(listaZenbaki2.contains(1));
89
90      }
91
92      @Test
93      public void testFirst() {
94          listaZenbaki1.addToRear(1);
95          listaZenbaki1.addToRear(2);
96          listaZenbaki1.addToRear(3);
97          listaZenbaki1.addToRear(4);
98          listaZenbaki1.addToRear(5);
99          assertEquals(listaZenbaki1.first(), new Integer(1));
100     }
101
102     @Test
103     public void testLast() {
104         listaZenbaki1.addToRear(1);
105         listaZenbaki1.addToRear(2);
106         listaZenbaki1.addToRear(3);
107         listaZenbaki1.addToRear(4);
108         listaZenbaki1.addToRear(5);
109         assertEquals(listaZenbaki1.last(), new Integer(5));
110     }
111
112     @Test
113     public void testContains() {
114         //elementu asko
115         listaZenbaki1.addToRear(1);
116         listaZenbaki1.addToRear(2);
117         listaZenbaki1.addToRear(3);
118         listaZenbaki1.addToRear(4);
119         listaZenbaki1.addToRear(5);
120         listaZenbaki1.addToRear(6);
121         listaZenbaki1.addToRear(7);
122         assertTrue(listaZenbaki1.contains(5));
123         assertFalse(listaZenbaki1.contains(53));
124         //elementu bakarra
125         listaZenbaki2.addToRear(1);
126         assertTrue(listaZenbaki2.contains(1));
127         assertFalse(listaZenbaki2.contains(5));
128     }
```

```
129
130     @Test
131     public void testFind() {
132        //elementu asko
133        listaZenbaki1.addToRear(1);
134        listaZenbaki1.addToRear(2);
135        listaZenbaki1.addToRear(3);
136        listaZenbaki1.addToRear(4);
137        listaZenbaki1.addToRear(5);
138        listaZenbaki1.addToRear(6);
139        listaZenbaki1.addToRear(7);
140        assertEquals(listaZenbaki1.find(3),new Integer(3));
141        assertNotEquals(listaZenbaki1.find(43),new Integer(43));
142        //elementu bakarra
143        listaZenbaki2.addToRear(1);
144        assertEquals(listaZenbaki2.find(1),new Integer(1));
145        assertNotEquals(listaZenbaki2.find(43),new Integer(43));
146     }
147
148     @Test
149     public void testIsEmpty() {
150        listaZenbaki1.addToRear(1);
151        assertEquals(listaZenbaki1.size(),1);
152        listaZenbaki1.removeLast();
153        assertEquals(listaZenbaki1.size(),0);
154        assertTrue(listaZenbaki1.isEmpty());
155     }
156
157     @Test
158     public void testSize() {
159        //elementu asko
160        listaZenbaki1.addToRear(1);
161        listaZenbaki1.addToRear(2);
162        listaZenbaki1.addToRear(3);
163        listaZenbaki1.addToRear(4);
164        listaZenbaki1.addToRear(5);
165        listaZenbaki1.addToRear(6);
166        listaZenbaki1.addToRear(7);
167        assertEquals(listaZenbaki1.size(),7);
168        listaZenbaki1.remove(5);
169        listaZenbaki1.remove(4);
170        assertEquals(listaZenbaki1.size(),5);
171        //elementu bakarra
172        listaZenbaki2.addToRear(1);
173        assertEquals(listaZenbaki2.size(),1);
174        //hutsa
```

```
175         listaZenbaki2.remove(1);
176         assertEquals(listaZenbaki2.size(),0);
177     }
178 
179     @Test
180     public void testIterator() {
181         listaZenbaki1.addToRear(1);
182         listaZenbaki1.addToRear(2);
183         listaZenbaki1.addToRear(3);
184         listaZenbaki1.addToRear(4);
185         listaZenbaki1.addToRear(5);
186         listaZenbaki1.addToRear(6);
187         listaZenbaki1.addToRear(7);
188         listaZenbaki1.adabegiakInprimatu();
189         Iterator<Integer> itr = listaZenbaki1.iterator();
190         while(itr.hasNext()){
191             int datua = itr.next();
192             System.out.println(datua);
193         }
194         assertNotNull(listaZenbaki1.iterator());
195 
196     }
197 
198 }
```

6.1.2 TestOrderedDoubleLinkedList.java

```
1  package aktorePelikulaPackage;
2 
3  import static org.junit.Assert.*;
4 
5  import org.junit.After;
6  import org.junit.Before;
7  import org.junit.Test;
8 
9  public class TestOrderedDoubleLinkedList {
10     OrderedDoubleLinkedList<Integer> lista1;
11     OrderedDoubleLinkedList<Integer> lista2;
```

```
12      @Before
13      public void setUp() throws Exception {
14          lista1 = new OrderedDoubleLinkedList<Integer>();
15          lista2 = new OrderedDoubleLinkedList<Integer>();
16      }
17
18      @After
19      public void tearDown() throws Exception {
20      }
21
22      @Test
23      public void testAdd() {
24          lista1.add(3);
25          lista1.add(1);
26          lista1.add(5);
27          lista1.add(7);
28          lista1.add(9);
29          lista1.add(2);
30          lista1.add(45);
31          assertEquals(lista1.size(), 7);
32          lista1.adabegiakInprimatu();
33          System.out.println("9 elementua kenduz gero");
34          lista1.remove(9);
35          assertEquals(lista1.size(), 6);
36          lista1.adabegiakInprimatu();
37      }
38
39      @Test
40      public void testMerge() {
41          System.out.println("LEHENENGO LISTA:");
42          lista1.add(3);
43          lista1.add(1);
44          lista1.add(5);
45          lista1.add(7);
46          lista1.add(9);
47          lista1.add(2);
48          lista1.add(45);
49          lista1.adabegiakInprimatu();
50          System.out.println("");
51          System.out.println("BIGARREN LISTA:");
52          lista2.add(2);
53          lista2.add(67);
54          lista2.add(1);
55          lista2.add(8);
56          lista2.add(6);
57          lista2.add(0);
```

```
58        lista2.add(4);
59        lista2.adabegiakInprimatu();
60        System.out.println("");
61        System.out.println("BI LISTEN ARTEKO MERGE-A EGIN ONDOREN:");
62        lista1.merge(lista2);
63
64     }
65
66  }
```

6.1.3 TestUnorderedDoubleLinkedList.java

```
1   package aktorePelikulaPackage;
2
3   import static org.junit.Assert.*;
4
5   import org.junit.After;
6   import org.junit.Before;
7   import org.junit.Test;
8
9   public class TestUnorderedDoubleLinkedList {
10
11     UnorderedDoubleLinkedList<Integer> listaZenbaki1;
12     @Before
13     public void setUp() throws Exception {
14        listaZenbaki1 = new UnorderedDoubleLinkedList<Integer>();
15
16     }
17
18     @After
19     public void tearDown() throws Exception {
20     }
21
22     @Test
23     public void testAddToFront() {
24        assertTrue(listaZenbaki1.isEmpty());
25        assertEquals(listaZenbaki1.count,0);
26        listaZenbaki1.addToFront(1);
```

```
27        assertFalse(listaZenbaki1.isEmpty());
28        listaZenbaki1.adabegiakInprimatu();
29        listaZenbaki1.addToFront(2);
30        listaZenbaki1.addToFront(3);
31        listaZenbaki1.addToFront(4);
32        listaZenbaki1.addToFront(5);
33        listaZenbaki1.addToFront(6);
34        listaZenbaki1.addToFront(7);
35        assertEquals(listaZenbaki1.count,7);
36        listaZenbaki1.adabegiakInprimatu();
37    }
38
39    @Test
40    public void testAddToRear() {
41        assertTrue(listaZenbaki1.isEmpty());
42        assertEquals(listaZenbaki1.count,0);
43        listaZenbaki1.addToRear(1);
44        assertFalse(listaZenbaki1.isEmpty());
45        assertEquals(listaZenbaki1.count,1);
46        listaZenbaki1.adabegiakInprimatu();
47        listaZenbaki1.addToRear(2);
48        listaZenbaki1.addToRear(3);
49        listaZenbaki1.addToRear(4);
50        listaZenbaki1.addToRear(5);
51        listaZenbaki1.addToRear(6);
52        listaZenbaki1.addToRear(7);
53        assertEquals(listaZenbaki1.count,7);
54        listaZenbaki1.adabegiakInprimatu();
55    }
56
57    @Test
58    public void testAddAfter() {
59        assertTrue(listaZenbaki1.isEmpty());
60        listaZenbaki1.addAfter(1,null);
61        assertFalse(listaZenbaki1.isEmpty());
62        listaZenbaki1.addAfter(2,1);
63        listaZenbaki1.addAfter(3,2);
64        listaZenbaki1.addAfter(4,3);
65        listaZenbaki1.addAfter(5,4);
66        listaZenbaki1.addAfter(6,5);
67        listaZenbaki1.addAfter(7,6);
68        assertEquals(listaZenbaki1.count,7);
69        System.out.println("8 elementua, 6 elementuaren ondoren txertatu");
70        listaZenbaki1.addAfter(8,6);
71        listaZenbaki1.adabegiakInprimatu();
72    }
```

```
73 }
```

7. Ondorioak

Lehenik eta behin, laborategi honetan lista estekatuak modu eraginkor batean erabiltzen ikasi dugu.

Gainera, *ArrayList*-ak eta lista estekatuak guztiz ordezkagarriak direla konturatu gara. Azken batean, laborategi honen helburua, *ArrayList*-ak lista estekatuekin ordezkatzea zen.

Lista estekatuak, *ArrayList*-ak baino denbora gehiago behar dute elementuak aurkitzeko edota txertatzeko. Hala ere, denbora hau oso txikia da.

Bukatzeko, lista estekatuak erabiltzea (*DoubleLinkedList*), oso ondo datorkigu, azken batean azkenengo elementua edo lehenengo elementua aurkitzeko edota txertatzeko kostu konstantea duelako.

Erreferentziak

[1] Gojenola, Koldo. Datu-Egiturak eta Algoritmoak: proiektua – Aktoreak eta pelikulak kudeatu – 2. eginkizuna (*DoubleLinkedList*). egela.ehu.eus, 2019. URL `https://egela.ehu.eus/pluginfile.php/2282877/mod_resource/content/13/lab-egitura-estekatuak.pdf`.

[2] Oracle. HashMap (Java Platform SE 7). docs.oracle.com, 2018. URL `https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html`.