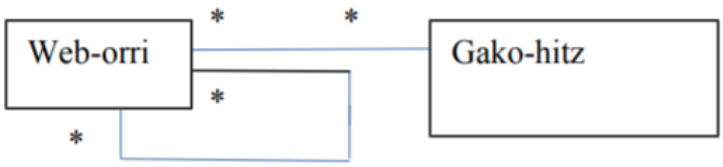
eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Web-a kudeatzeko aplikazioa (2. eginkizuna)



Egileak:
Aitor San José, Martin Amezola, Leire Garcia

Irakasgia:
Datu-Egiturak eta Algoritmoak

Irakasleak:
Iñigo Mendialdua eta Koldobika Gojenola

2. maila

46. taldea

2020.eko azaroaren 1

Aurkibidea

1 Sarrera eta helburuak

Datu-Egiturak eta Algoritmoak ikasgaiako proiektua Web-orri kopuru handia kudeatuko dituen aplikazioa sortzea da.

Ikasgai honetan, asko azpimarratzen da programaren kostua, beraz, inplementatzerako orduan datuen maneiatzearen arabera datu egitura bat edo bestea erabili dugu, inplementazioa gero eta eraginkorragoa izatearren.

Beraz, aurrean aipatutakoa argi ikusteko, zenbait eginkizun bete beharko ditugu lahuhilekoan zehar.

Bigarren eginkizun honetan, datu egitura berri bat ikasi eta inplementatuko dugu: *CircularLinkedList*.

Laborategi honek bi helburu nagusi ditu. Lehenengoa, lista estekatuak inplementatzen ikastea. Bigarrena, lista hauek aurreko laborategiko *ArrayList* betengatik ordezkatzea.

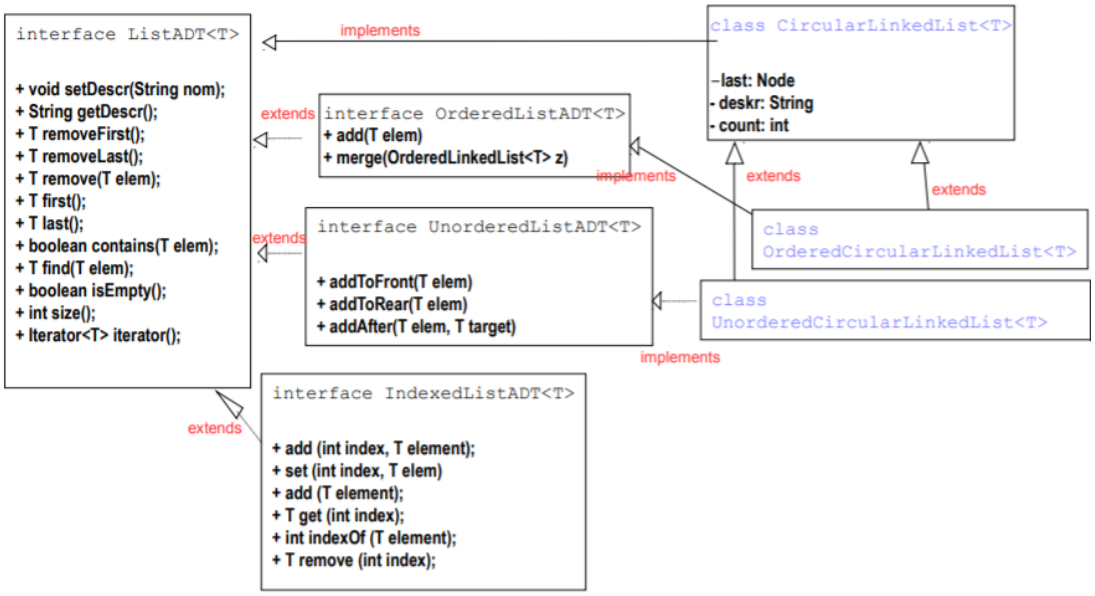
Printzipioz *CircularLinkedList*, *UnorderedCircularLinkedList*, eta *OrderedCircularLinkedList* klaseak izango ditgu, azkenengoaren inplementazioa aukerazkoa izanik. *IndexedList*-aren implementazioa ez da eskatzen. Lehen aipatu dugun bezala, lista estekatu hauek aurreko laborategiko lista bat edo gehiagotan ordezkatu beharko ditugu. Horren ondorioz, aldatutako klaseak jarri beharko ditugu. Guk Hitza klasea aukeratu dugu.

2 Diseinua

Bigarren eginkizun honetan, eskatutakoaren klase diagrama enuntziatuan zegoen, gure lehenengo enuntziatuaren klaseak faltan izanda (1. irudia).

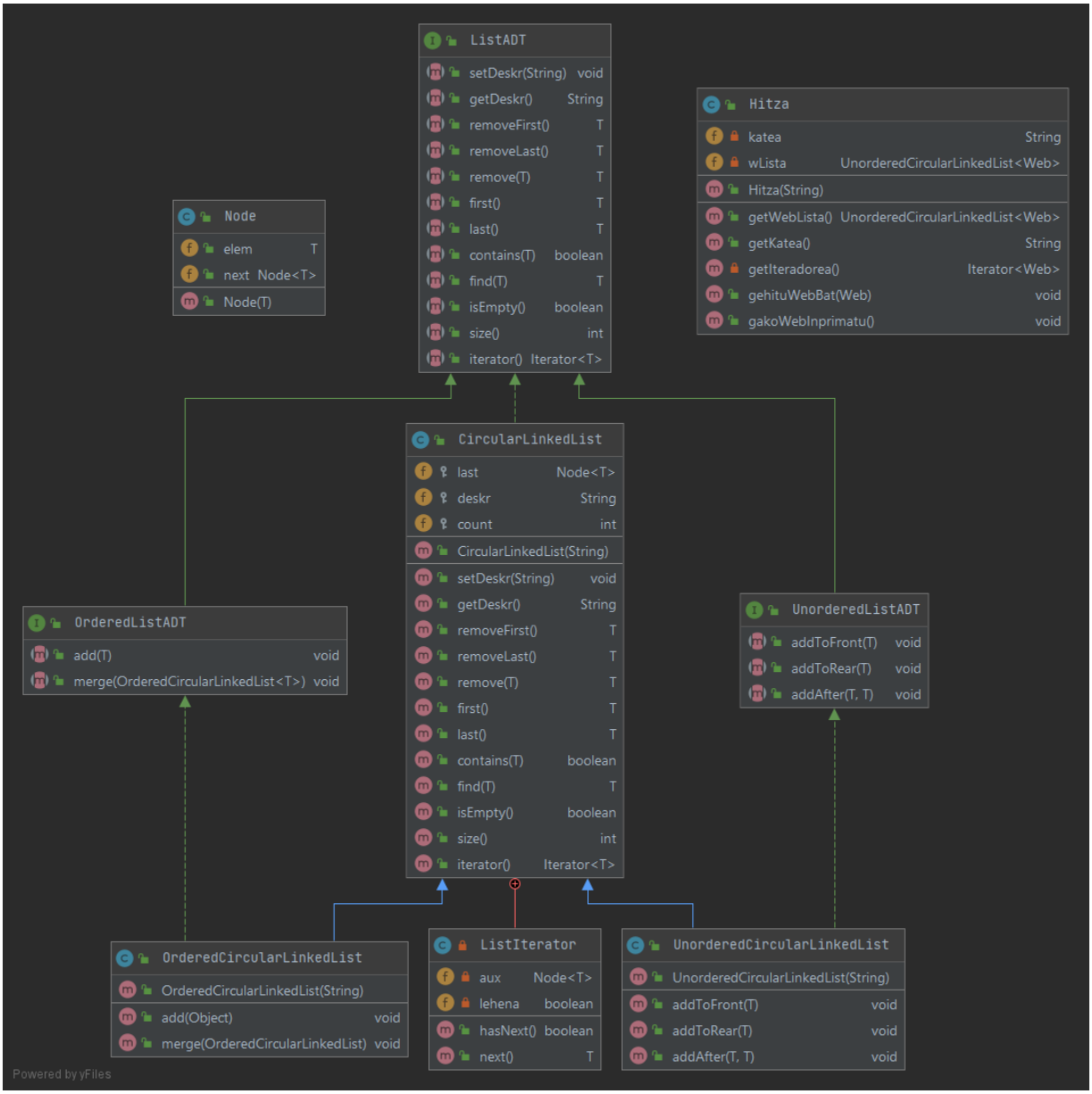
Klase diagraman (1. irudia) ikus daitekeenez lista estekatuen azpiklaseak agertzen dira.

Klase hauek eginkizunaren bigarren zatian erabiliko ditugu; oraingo atalan; klase horietako metodoak inplementatuko ditugu. Metodo batzuk amankomunean dituzte, esate baterako *+boolean contains(T elem)*; eta beste batzuk, klase bakoitzean inplementatu beharko dira *+add(T elem)* eta *+ addToFront(T elem)*, adibidez.



1. irudia: Enuntziatuan emandako diseinua.

Eginkizunaren bigarren atalean, 1. eginkizuneko zerrenda bat (*ArrayList*), gure kasuan Hitza klasean dagoena, *CircularLinkedList*-en ordez aldatuko dugu. Horretarako, Hitza klasean *ArrayList* deiak aldatuko ditugu, bai eta klasean bertan dauden metodoak egokitu (2. irudia).



2. irudia: Aldaketa ostean geratzen den diseinua.

3 Datu egitura nagusien deskribapena

Datu egitura mota bakarra erabili dugu eginkizun honetan: *LinkedList*-a (zuzenagoak izateko, *CircularLinkedList*).

Datu egitura mota honen berezitasuna honako da: nodo bakoitza aurrean daukan nodoarekin lotuta dago eta azken nodoa lehen nodoarekin lotuta dago. Elementuak txertatu edo ezabatzerakoan *ArrayList*-ak baino askoz ere eraginkorragoak dira. Izan ere, *ArrayList* baten elementu bat hasieran ezabatzerakoan, atzetik dauden elementuak ezkerrera mugitu behar dira, eta berdina gertatzen da lehenengo tokian txertatu behar bada, O(n)-ko kostua izanik. (*Circular*)*LinkedList* batean, aldiz, kostua O(1) da, soilik 4-5 agindu bete behar direlako.

4 Metodo nagusien diseinu eta inplementazioa

4.1 CircularLinkedList

4.1.1 Lehenengo elementua ezabatu: *removeFirst()*

public T removeFirst()
// Aurrebaldintza:
// Postbaldintza: Listako lehen elementua ezabatuta dago.

- Proba-Kasuak:

Zerrenda	Emaitza
()	()
(a)	()
(a, b, c)	(b, c)

- Algoritmoa:

```
1        baldin eta (last == null)
2        {
3            bueltatu null;
4        }
5        bestela
6        {
7            ezabatu lehen elementua;
8            kontadorea eguneratu;
9        }
```

- Kostua:

  Kostu konstantea O(1).

4.1.2 Azkenengo elementua ezabatu: *removeLast()*

public T removeLast()
// Aurrebaldintza:
// Postbaldintza: Listako azken elementua ezabatuta dago.

- Proba-Kasuak:

Zerrenda	Emaitza
()	()
(a)	()
(a, b, c)	(a, b)

- Algoritmoa:

```
1        baldin eta (last == null)
2        {
3            bueltatu null;
4        }
5        bestela
6        {
7            ezabatu azken elementua;
8            kontadorea eguneratu;
9        }
```

- Kostua: O(n).

  Non n = Listako elementu kopurua.

4.1.3 Elementua ezabatu: *remove(T elem)*

public T remove(T elem)
// Aurrebaldintza: Borratuko den elementua sartu.
// Postbaldintza: Sartutako elementuaren lehen agerpena borratu da eta erreferentzia bueltatu da. Elementua ez badago null bueltatuko du.

- Proba-Kasuak:

Zerrenda	Sarrera	Emaitza	Zerrenda amaieran
()	a	null	()
(a)	null	null	(a)
(a)	a	a	()
(a,b,a)	a	a	(b,a)

- Algoritmoa:

```
1                boolean aurk=false;
2                baldin eta (last==null) {
3                        bueltatu null;
4                }
5                baldin eta (elem==null) {
6                        bueltatu null;
7                }
8                apuntadore auxiliarra sortu;
9
10
11               elementua aurkitu ez duen bitartean errepikatu {
12                   apuntadorea elementua apuntatzen badu
13                   {
14                       ezabatu elementu hori;
15                   }
16                   bestela jarraitu;
17               }
18               baldin (ez du aurkitu)
19               {
20                   bueltatu null;
21               }bestela
22               {
23                   kontadorea eguneratu;
24                   bueltatu elementua;
25               }
```

- Kostua: O($\frac{n}{2}$)

  Non n = Listako elementu kopurua.

4.1.4 Lehenengo elementua bueltatu: *first()*

public T first()
// Aurrebaldintza:
// Postbaldintza: Listako lehen elementua bueltatzen du.

- Proba-Kasuak:

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

- Algoritmoa:

```
1       baldin eta last.next == null {
2                       bueltatu null;
3               }
4               bestela
5               {
6                   bueltatu last apuntadorearen hurrengoan dagoen elementua;
7               }
```

- Kostua: O(1) konstantea.

4.1.5 Azkenengo elementua bueltatu: *last()*

public T last()
// Aurrebaldintza:
// Postbaldintza: Listako azken elementua bueltatzen du.

- Proba-Kasuak:

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

- Algoritmoa:

```
1       baldin eta last == null {
2                       bueltatu null;
3               }
4               bestela
5               {
6                   bueltatu last apuntadorean dagoen elementua;
7               }
```

- Kostua: O(1) konstantea.

4.1.6 Elementu bat listan badago: *contains(T elem)*

public boolean contains(T elem)
// Aurrebaldintza: Elementu bat jasotzen du.
// Postbaldintza: Elementua badago true else null.

- Proba-Kasuak:

Zerrenda	Elem	Emaitza
()	a	false
(a)	a	true
(a, b, c)	b	true
(a, b, c)	d	false

- Algoritmoa:

```
1   uneko apuntadore bat sortu last punteroan;
2               aurkitu ez duen bitartean eta unekoaren hurrengoa != last {
3                       elementua aurkitzen bada
4                       {
5                           bueltatu true;
6                       }
7                       else
8                       {
9                           bueltatu false;
10                      }
11              }
```

- Kostua: o($\frac{n}{2}$).
  Non n = listako elemetu kopurua.

4.1.7 Elementu bat bilatu: *find(T elem)*

public T find(T elem)
// Aurrebaldintza: Elementu bat jasotzen du.
// Postbaldintza: Listan bilatzen ari den elementua bueltatuko du, listan badago. Bestela null.

- Proba-Kasuak:

Zerrenda	Elem	Emaitza
()	a	null
(a)	a	a
(a)	b	null
(a, b, c)	b	b
(a, b, c)	h	null

- Algoritmoa:

```
1   baldin eta elementua hori listan badago {
2                       bueltatu elem;
3               }bestela {
4                       bueltatu null;
5               }
```

- Kostua: o($\frac{n}{2}$).
  Non n = listako elementu kopurua.

4.2 UnorderedCircularLinkedList

4.2.1 Elementu bat hasieran txertatu: *addToFront(T elem)*

public void addToFront(T elem)
// Aurrebaldintza:Elementu bat jasotzen du.
// Postbaldintza: Elementua (elem) hasieran gehitzen du.

- Proba-Kasuak:

Zerrenda	Elem	Emaitza
()	a	(a)
(a)	b	(b, a)
(a, b, c)	h	(h, a, b, c)

- Algoritmoa:

```
1       baldin eta lista hutsa {
2           sortu lista eta txertatu;
3       }bestela
4       {
5           txertatu hasieran;
6       }
7       kontadorea eguneratu;
```

- Kostua: O(1) konstantea.

4.2.2 Azkenengo posizioan txertatu: *addToRear(T elem)*

public void addToRear(T elem)
// Aurrebaldintza: Elementu bat jasotzen du.
// Postbaldintza: Listan elementua txertatu da azkenengo posizioan.

- Proba-Kasuak:

Zerrenda	Elem	Emaitza
()	a	(a)
(a)	b	(a, b)
(a, b, c)	d	(a, b, c, d)

- Algoritmoa:

```
1       baldin eta lista hutsa {
2           sortu lista eta txertatu;
3       }bestela
4       {
5           txertatu amaieran;
6       }
7       kontadorea eguneratu;
```

- Kostua: O(1) konstantea.

4.2.3 Txertatu elementua target eta gero: *addAfter(T elem, T target)*

public void addAfter(T elem, T target)
// Aurrebaldintza: Elementu bat eta "Target"bat jasotzen da.
// Postbaldintza: Elem, "Target"elementuaren atzean txertatuta egongo da.

- Proba-Kasuak:

Zerrenda	Elem	Target	Emaitza
()	a	b	()
(a)	b	a	(a, b)
(a, b, c)	d	c	(a, b, c, d)
(a, b, c)	d	b	(a, b, d, c)

- Algoritmoa:

```
1        berria nodoa sortu (elem) elemetuarekin.
2                unekoa = last;
3                egin{
4                        baldin eta unekoa target-ren berdina {
5                                baldin eta unekoa == last {
6                                        addToRear(elem);
7                                }bestela {
8                                        txertatu unekoa eta gero;
9                                }
10                       }
11                       bestela {
12                       aurreratu unekoa;
13                       }
14               }bitartean(unekoaren hurrengoa!=last);
15               eguneratu kontagailua;
```

- Kostua: O($\frac{n}{2}$).
  Non n = Listako elementu kopurua.

4.3 OrderedCircularLinkedList

4.3.1 Elementua bere posizioan gehitzen du: *add(T elem)*

public void add(T elem)
// Aurrebaldintza: Elementu bat jasotzen da.
// Postbaldintza: Elementua sartu da bere posizioan.

- Proba-Kasuak:

Zerrenda	Elem	Emaitza
()	a	(a)
(a)	b	(a, b)
(b)	a	(a, b)
(a, b)	c	(a, b, c)
(a, c)	b	(a, b, c)

- Algoritmoa:

```
1
2               txertatuta boolearra false ipini;
3               nodo berria sortu;
4               baldin eta last == null {
5                       berria berari apuntatu;
6                       last berriari apuntatu;
7                       eguneratu kontadorea;
8               }bestela{
9                       unekoa last.next apuntatu;
10                      aurrekoa null apuntatu;
11                      egin aurrekoa!=last && txertatuta ez dagoen bitartean {
12                              baldin eta(elem txikiagoa unekoa apuntatzen duen
                                    ↪ elem) {
13                                  baldin eta unekoa==last.next {
14                                          txertatu hasieran;
15                                          eguneratu last;
16                                  }bestela {
17                                          txertatu erdian;
18                                  }
19                                  txertatuta boolearra true ipini;
20                                  eguneratu kontadorea;
21                              }bestela {
22                                      aurrekoa unekoa apuntatu;
23                                      unekoa unekoa.next apuntatu;
24                              }
25                      }
26                      baldin ez dago txertatua {
27                              berria.next last.next apuntatu;
28                              last.next berria apuntatu;
29                              last berria apuntatu;
30                      }
31              }
```

- Kostua: O($\frac{n}{2}$).
  Non n = Listako elementu kopurua.

4.3.2 Bi lista ordenatu bildu: *merge(OrderedCircularLinkedList<T> z)*

public void merge(OrderedCircularLinkedList<T> z)
// Aurrebaldintza: Jasotzen da OrderedCircularLinkedList motatako lista bat.
// Postbaldintza: Bi listak (jasotakoa eta gurea) OrderedCircularLinkedList batean bilduta.

- Proba-Kasuak:

Zerrenda1	Zerrenda2	Emaitza
()	()	()
(a)	()	(a)
()	(a)	(a)
(a, c)	(b)	(a, b, c)
(a, b, c)	(d, e, f)	(a, b, c, d, e, f)

- Algoritmoa:

```
1
2  baldin eta z.last == null {
3          last apuntatuko du z.last;
4          }bestela {
5                  baldin eta last == null {
6                          last apuntatuko du z.last;
7                  }bestela {
8                          l1 nodoa last.next apuntatu;
9                          l1aur nodoa last apuntatu;
10                         l2 nodoa z.last.next apuntatu;
11                         s gurelistaren luzeera;
12                         egin( gure listaren luzeera desberdin gure listaren luzeeera
                               ↪  + jasotakoaren listaren luzeera) bitartean{
13                                 berria nodoa sortu l2.elem sartuz;
14                                 if(l2.elem txikiagoa l1.elem) {
15                                         berria.next apuntatzen du l1;
16                                         l1aur.next apuntatzen du berria;
17                                         l2 aurreratu;
18                                         l1 apuntatzen du berria;
19                                         count kontadorea eguneratu;
20                                 }bestela {
21                                         baldin (l1==last) {
22                                             //lista amaitu da eta l2 l1 baino
                                                   ↪ handiagoa da
23                                             berria.next apuntatzen du last.next;
24                                             l1.next apuntatzen du berria;
25                                                 last apuntatzen du berria;
26                                                 count kontadorea eguneratu;
27                                                 }
28                                                 l1aur apuntatzen du l1;
29                                                 l1 apuntatzen du l1.next;
30                                         }
31                                 }
32
33                         }
34                 }
```

- Kostua: O(n).

  Non n = Listako elementu kopurua. (Lista luzeena)

5 Kodea

5.1 Interfazeak

5.1.1 ListADT.java

```java
package packlEnuntziatu2;

import java.util.Iterator;

public interface ListADT<T> {
        public void setDeskr(String nom);
        public String getDeskr();
        public T removeFirst();
        public T removeLast();
        public T remove(T elem);
        public T first();
        public T last();
        public boolean contains(T elem);
        public T find(T elem);
        public boolean isEmpty();
        public int size();
        public Iterator<T> iterator();
}
```

5.1.2 OrderedListADT.java

```java
package packlEnuntziatu2;

public interface OrderedListADT<T extends Comparable<T>> extends ListADT<T> {
        public void add (T elem);
        public void merge (OrderedCircularLinkedList<T> z);
}
```

5.1.3 UnorderedListADT.java

```java
package packlEnuntziatu2;

public interface UnorderedListADT<T> extends ListADT<T>{
        public void addToFront(T elem);
        public void addToRear(T elem);
        public void addAfter(T elem, T target);
}
```

5.2 Bigarren eginkizuneko inplementazioak

5.2.1 CircularLinkedList.java

```java
package packlEnuntziatu2;

import java.util.Iterator;

public class CircularLinkedList<T> implements ListADT<T>{
        protected Node<T> last;
        protected String deskr;
        protected int count;



        // eraikitzailea
        public CircularLinkedList(String pDeskr)
        {
                this.last = null;
                this.deskr = pDeskr;
                this.count = 0;
        }


        // metodoak

        public void setDeskr(String nom)
        {
                deskr=nom;
        }
        public String getDeskr()
        {
                return deskr;
        }
        public T removeFirst()
        {
                if(last==null) {
                        return null;
                }
                Node<T> unekoa= last.next;
                last.next=unekoa.next;
                count--;
                return unekoa.elem;
        }
        public T removeLast() {
                Node<T> unekoa=last;
                if(unekoa==null) {
                        return null;
                }
                while(unekoa.next!=last) {
                        unekoa=unekoa.next;
                }
                last=unekoa;
                unekoa=unekoa.next;
                last.next=unekoa.next;
```

```
52                  count--;
53                  return unekoa.elem;
54
55          }
56          public T remove(T elem) {
57                  boolean aurk=false;
58                  Node<T> unekoa=last.next;
59                  Node<T> aurrekoa=null;
60                  if(unekoa==null) {
61                          return null;
62                  }
63                  while(!aurk&& aurrekoa!=last) {
64                          if(unekoa.elem.equals(elem)) {
65                                  if(unekoa==last) {
66                                          removeLast();
67                                          aurk=true;
68                                  }else {
69                                          aurrekoa.next=unekoa.next;
70                                          aurk=true;
71                                  }
72                          } else {
73                                  aurrekoa=unekoa;
74                                  unekoa=unekoa.next;
75                          }
76                  }
77                  if(!aurk) {
78                          return null;
79                  }else {
80                          count--;
81                          return unekoa.elem;
82                  }
83          }
84          public T first() {
85                  Node<T> unekoa=last;
86                  if(unekoa==null) {
87                          return null;
88                  }
89                  return unekoa.next.elem;
90          }
91          public T last() {
92                  if(last==null) {
93                          return null;
94                  }
95                  return last.elem;
96          }
97          public boolean contains(T elem) {
98                  boolean aurk=false;
99                  Node<T> unekoa=last;
100                 while(unekoa.next!=last && !aurk) {
101                         if(unekoa.elem.equals(elem)) {
102                                 aurk=true;
103                         }else {
104                                 unekoa=unekoa.next;
105                         }
106                 }
```

```
107                 return aurk;
108         }
109         public T find(T elem) {
110                 if(contains(elem)) {
111                         return elem;
112                 }else {
113                         return null;
114                 }
115         }
116         public boolean isEmpty() {
117                 return(last==null);
118         }
119         public int size() {
120                 return count;
121         }
122         public Iterator<T> iterator(){
123                 return new ListIterator();
124         }
125         private class ListIterator implements Iterator<T>{
126                 private Node<T> aux=last;
127                 private boolean lehena=false;
128                 public boolean hasNext() {
129                         if(isEmpty()) {
130                                 return false;
131                         }
132                         else if (aux.equals(last) && lehena ){
133                                 return false;
134                         }
135                         else {
136                                 lehena=true;
137                                 return true;
138                         }
139                 }
140
141                 public T next() {
142                         return aux.elem;
143                 }
144
145         }
146
147
148 }
```

5.2.2 UnorderedCircularLinkedList.java

```
1  package packlEnuntziatu2;
2  public class UnorderedCircularLinkedList<T> extends CircularLinkedList<T> implements
       ↪ UnorderedListADT<T> {
3
4          public UnorderedCircularLinkedList(String pDeskr){
5                  super(pDeskr);
6          }
7
8          public void addToFront(T elem) {
9                  Node<T> berria= new Node<T>(elem);
10                 if(isEmpty()) {
11                         last=berria;
12                         berria.next=berria;
13                 }else {
14                         berria.next=last.next;
15                         last.next=berria;
16                 }
17                 count++;
18         }
19
20         public void addToRear(T elem) {
21                 Node<T> berria= new Node<T>(elem);
22                 if(isEmpty()) {
23                         last=berria;
24                         berria.next=berria;
25                 }else {
26                         berria.next=last.next;
27                         last.next=berria;
28                         last=berria;
29                 }
30                 count++;
31         }
32
33         public void addAfter(T elem, T target) {
34                 Node<T> berria= new Node<T>(elem);
35                 Node<T> unekoa= last;
36                 do{
37                         if(unekoa.elem.equals(target)) {
38                                 if(unekoa==last) {
39                                         addToRear(elem);
40                                 }else {
41                                         berria.next=unekoa.next;
42                                         unekoa.next=berria;
43                                 }
44                         }
45                         else {
46                         unekoa=unekoa.next;
47                         }
48                 }while(unekoa.next!=last);
49                 count++;
50         }
51 }
```

5.2.3 OrderedCircularLinkedList.java

```
1  package packlEnuntziatu2;
2
3  public class OrderedCircularLinkedList<T extends Comparable<T>> extends
       ↪ CircularLinkedList<T> implements OrderedListADT<T> {
4
5
6          public OrderedCircularLinkedList(String pDeskr)
7          {
8                  super(pDeskr);
9
10         }
11
12         public void add(T elem) {
13                 Comparable <T> generiko = (Comparable <T>) elem;
14                 boolean txertatuta=false;
15                 Node<T> berria= new Node<T>((T) generiko);
16                 if(last==null) {
17                         berria.next=berria;
18                         last=berria;
19                         count ++;
20                 }else {
21                         Node<T> unekoa= last.next;
22                         Node<T> aurrekoa= null;
23                         while(aurrekoa!=last && !txertatuta) {
24                                 if(elem.compareTo((T) unekoa.elem)<=0) {
25                                         if(unekoa==last.next) {
26                                                 last.next=berria;
27                                                 berria.next=unekoa;
28                                         }else {
29                                                 berria.next=unekoa;
30                                                 aurrekoa.next=berria;
31                                         }
32                                         txertatuta=true;
33                                         count ++;
34                                 }else {
35                                         aurrekoa=unekoa;
36                                         unekoa=unekoa.next;
37                                 }
38                         }
39                         if(!txertatuta) {
40                                 berria.next=last.next;
41                                 last.next=berria;
42                                 last=berria;
43                         }
44                 }
45         }
46
47         public void merge(OrderedCircularLinkedList<T> z) {
48                 if(z.last==null) {
49
50                 }else {
51                         if(last==null) {
52                                 last=z.last;
```

```
53                      }else {
54                              Node<T> l1= last.next;
55                              Node<T> l1aur= last;
56                              Node<T> l2= z.last.next;
57                              int s= this.size();
58                              int konp;
59                              while(this.size()!=s+z.size()){
60                                      Node<T> berria= new Node<T>(l2.elem);
61                                      konp= l2.elem.compareTo((T) l1.elem);
62                                      if(konp<=0) {
63                                              berria.next=l1;
64                                              l1aur.next=berria;
65                                              l2=l2.next;
66                                              l1=berria;
67                                              count ++;
68                                      }else {
69                                              if(l1==last) {                  //lista
                                                    ↪ amaitu da eta l2 l1 baino
                                                    ↪ handiagoa da
70                                                      berria.next=last.next;
71                                                      l1.next=berria;
72                                                      last=berria;
73                                                      count ++;
74                                              }
75                                              l1aur=l1;
76                                              l1=l1.next;
77                                      }
78                              }
79
80                      }
81              }
82      }
83
84  }
```

5.2.4 Node.java

```
1   package packlEnuntziatu2;
2
3   public class Node<T>{
4           public T elem;
5           public Node<T> next;
6
7
8           public Node (T elem)
9           {
10                  this.elem = elem;
11                  this.next = null;
12          }
13
14
15  }
```

5.3 Lehen eginkizuneko kode berria

5.3.1 Hitza.java

```java
package packlEnuntziatu2;
import packlEnuntziatu1.Web;
import java.util.Iterator;

public class Hitza {
	// atributuak
	private String katea;
	private UnorderedCircularLinkedList<Web> wLista;

	// eraikitzailea
	public Hitza (String pKatea){
		this.katea = pKatea;
		this.wLista = new UnorderedCircularLinkedList<Web>("Weben lista");
	}

	// getters
	public UnorderedCircularLinkedList<Web> getWebLista(){
		return this.wLista;
	}

	public String getKatea(){
		return this.katea;
	}

	// metodoak
	private Iterator<Web> getIteradorea(){
		return this.wLista.iterator();
	}


	public void gehituWebBat(Web pWeb){
		this.wLista.addToRear(pWeb);
	}

	public void gakoWebInprimatu() {
		System.out.println(" ");
		Iterator<Web> itr= this.getIteradorea();
		Web w=null;
		while(itr.hasNext()) {
			w=itr.next();
			w.webInprimatu();
		}
	}
}
```

6 Ondorioak

Laborategi honetan lista estekatuak landu ditugu eta hauek *ArrayList*-en ordez erabiltzen ikasi dugu. Azken finean, laborategi honen helburua *ArrayList*-ak lista estekatuekin ordezkatzea zen.

Lista estekatuak *ArrayList*-ak baino kode lerro gehiago behar dituzte elementuak bilatze ko edota txertatxeko, hala ere, denbora hau oso txikia da eta kostua konstantea izaten jarraitzen du.

Bukatzeko, lista estekatuak erabiltzea, *CircularLinkedList*, oso ondo datorkigu hasieran elementu bat txertatu nahi dugunean kostua konstantea delako eta *ArrayList*-ean kostua lineala baita

7 Erreferentziak

- **Mendialdua, Inigo. Datu-Egiturak eta Algoritmoak:**
  Egitura estekatuen laborategia.
  URL
  `https://egela.ehu.eus/pluginfile.php/4272370/mod_resource/content/1/lab-egitura-estekatuak%20%281%29.pdf`