Pilak eta ilarak

Aurkibidea

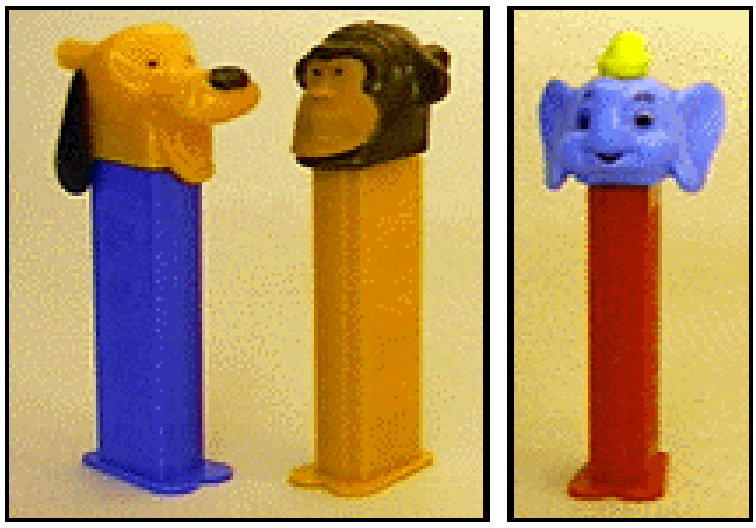
- *Pilak*
  - Zer dira?
  - Ezaugarriak
  - Metodo nagusiak
  - Adibideak
- *Inplementazio adibideak*
  - Array-etan oinarrituta
  - Zerrendetan oinarrituta
- *Ilarak*
  - Zer dira?
  - Ezaugarriak
  - Metodo nagusiak
  - Adibideak
- *Inplementazio adibideak*
  - Array-etan oinarrituta
  - Zerrendetan oinarrituta

Pilak

Pilak
Zer dira?. Ezaugarriak

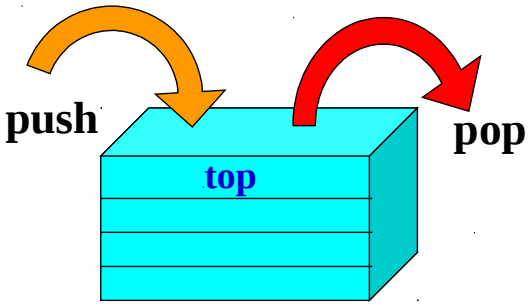
- Datu egitura **lineala** non:
  - Txertatu den elementu berriena atzitu daiteke soilik (**top** edo gailur)
  - Objektuak **mutur batetik** txertatzen eta ezabatzen dira **LIFO** (**L**ast **I**n **F**irst **O**ut) irizpidea jarraituz.
- Egokiak dira:
  - **Azken** elementua atzitu behar dugunean soilik.
  - Zerrenda bat **alderantzikatu** nahi dugunean.

Pilak
Adibidea



Pilak
Metodo nagusiak

- Mutur batetik txertatu (**pilaratu**): **`push(x)`**
- Mutur berdinetik atera (**despilatu**): **`pop()`**



Eskuragarri dagoen elementu bakarra azken txertatua da (gailur): **top**

Pilak
Metodo nagusiak eta metodo laguntzaileak

Metodo nagusiak	*Esanahia*
void **push**(T elem)	Pilaren gailurrean elementu bat **txertatzen** da
T **pop**()	Pilaren gailurrean dagoen elementua **ezabatzen** da posizio horretan zegoen elementua itzuliz

Metodo laguntzaileak	*Esanahia*
int **size**()	Pilaren **tamaina** itzultzen du
boolean **isEmpty**()	True pila **hutsa** bada eta false kontrako kasuan
T **peek()**	**Gailurrean** dagoen elementua itzultzen du gailurretik ezabatu gabe.

Pilak
Adibideak

- Pilak konpiladoreetan:
  - **Sinboloen oreka** konprobatzeko
  - **Metodoen deialdiak gordetzeko:**
    Errekurtsioa
    - Deialdi bakoitzean gordetzen da:
      - Metodoen aldagai lokalak (kontextu)
      - Deialdia egin den lerroa

Pilak: metodoen deialdiak gordetzeko

```
1  public class Faktoriala{
2   public static void main(String args[]){
3    int  param = Integer.parseInt(args[0]);
4    long emaitza = fakt(param);
5    System.out.println( args[0] +"! = " + emaitza);
6   }// main amaiera
7   public static long fakt(int n){
8     if(n<=1)
9       return 1;
10    else
11      return n*fakt(n-1);
12  }// fakt amaiera
13 }// Faktoriala amaiera
```

Pilak

Adibide 1: Parentesi konprobaketa

- Ondo:
  - –
  - ()
  - ((()()))
- Gaizki:
  -)(
  - (()
  - ())

- Simboloen oreka konprobatzeko erregelak:
- Oinarrizkoa: ()
- Sekuentzia: ()()
- Habiratuak: (())

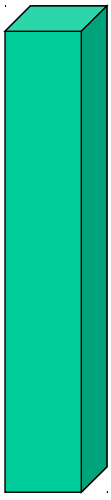
Pilak

Adibide 1: Parentesi konprobaketa

- **Erregelak**:
  - **(** aurkitzen dugun bakoitzean, pilan txertatzen dugu.
  - **)** aurkitzen dugun bakoitzean, gailurrean dagoen **(** ateratzen dugu
  - Parentesi katea **zuzena** da, kate guztia korritu dugunean **pila hutsik badago**.

Pilak

Adibide: Parentesi konprobaketa (()(()())())

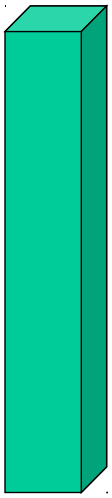


Erregelak:
Parentesi **irekia**: **pilaratu**
Parentesi **itxia**: **despilatu**

(( ) (( ) ( )) ( ))

Pilak

Adibide 1: Parentesi konprobaketa (()(()())())

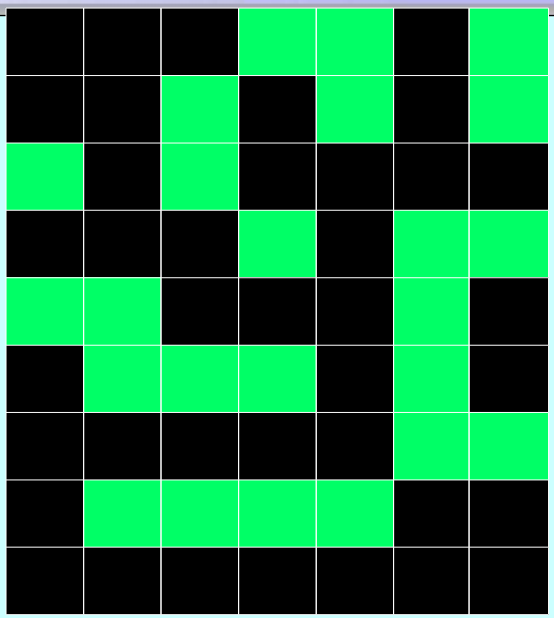


Zuzena: katea korritu dugu eta **pila hutsik dago**

(()(()(()())())

Labirintoa zeharkatzen

```
private int [ ] [ ] labirinto =
{ {1,1,1,0,0,1,0},
  {1,1,0,1,0,1,0},
  {0,1,0,1,1,1,1},
  {1,1,1,0,1,0,0},
  {0,0,1,1,1,0,1},
  {1,0,0,0,1,0,1},
  {1,1,1,1,1,0,0},
  {1,0,0,0,0,1,1}
  {1,1,1,1,1,1,1} };
```



labirinto [errenkada] [zutabe] == 1 “bidea da”

labirinto [errenkada] [zutabe] == 0 “horma da”

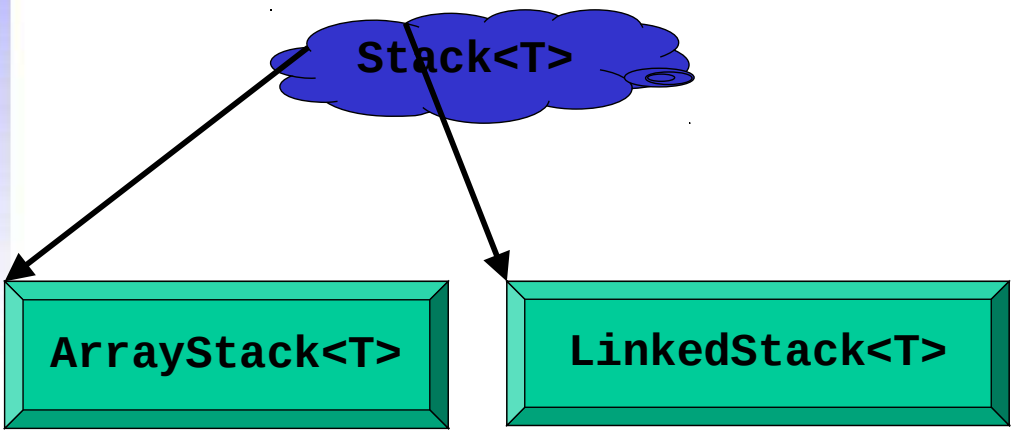
Labirintoa zeharkatzen

```
public boolean traverse (){
  Poner posición inicial en la cima de la pila
  Marcar esa posición como visitada
  while “la posición no sea la salida” y
                  “tenga esperanza de poder llegar a la salida” {
    posición = extraer la cima de la pila
    if “la posición es la salida” terminar el while
    else {
           para cada casilla posible de avance desde posición:
                 apilar esa casilla /* puede verse como
                     una tarea que queda por hacer: intentar
                     el camino desde esa casilla*/
                 Marcar esa posición como visitada
    }
  }//end while
  if “la posición es la salida” return true
  else return false
}
```

Pila interfazea

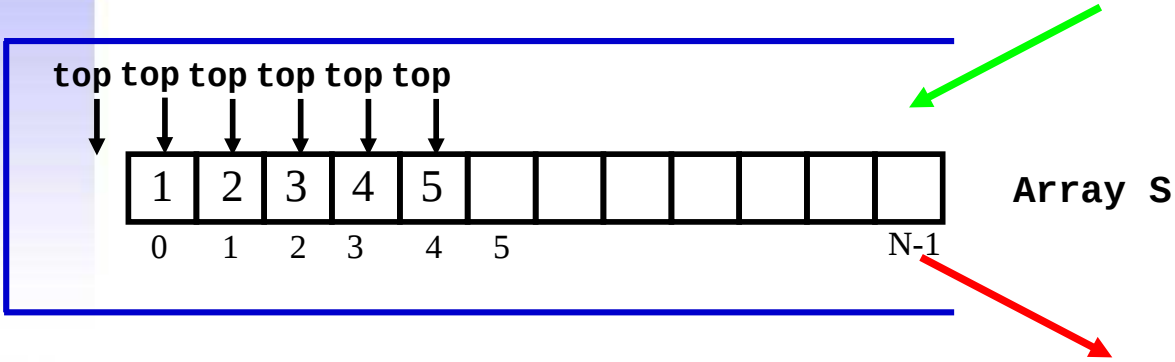
```
public interface Stack<T> {
  public int size();
  public boolean isEmpty();
  public void push(T o);
  public T pop();
  public T peek();
}
```

Pilak:
Inplementazio desberdinak interfaze batentzat



Pilak:
Array-etan oinarritutako inplementazioa

S array baten bidez inplementatutako **pila** batean **o** objektu bat txertatu nahi dugu




```
top = top +1; //top etiketa posizio bat mugitu
s[top]= o;    //top posizio berrian elementua txertatzen da
```

Pilak:
Array-etan oinarritutako inplementazioa

```
public class ArrayStack<T> implements Stack<T> {
  public static final int CAP = 1000;
  private int maxSize;        // size of stack array
  private T[] stackArray;
  private int top;            // top of stack

  public ArrayStack() {this(CAP);}  // constructor
  public ArrayStack(int alturaMax) {
    maxSize = alturaMax;  // set array size
    stackArray = (T[])(new Object[maxSize]);
                      // create array
    top = -1;         // no items yet
  }
```


Pilak:
Array-etan oinarritutako inplementazioa

```
public void push(T elemento) {
     // put item on top of stack
 stackArray[++top] = elemento;
     // increment top, insert item
}

public T pop() {
     // take item from top of stack
    return stackArray[top--];
         // access item, decrement top
}
```

Pilak:
Array-etan oinarritutako inplementazioa

```
    public T peek()  { // peek at top of stack
        return stackArray[top];
    }

    public boolean isEmpty() { // true if empty
        return (top == -1);
    }

    public int size() {
        return (1+top);
    }
}
```

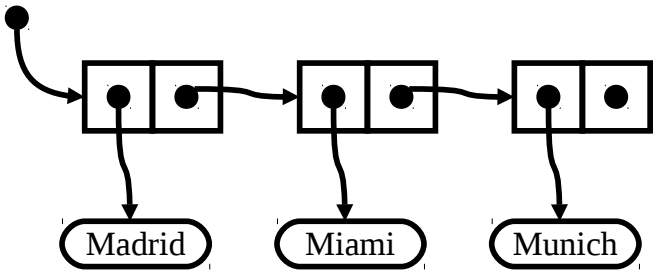

Pilak:
iteradore bat nahi dugu?

```
    public Iterator<T> iterator() {
          return new ReverseArrayIterator(); }
```

Nested class

```
    public class ReverseArrayIterator implements Iterator<T> {

      private int i =  top;

      public boolean hasNext() {
        return i >= 0; }

      public Item next() {
        return stackArray[i--]; }

      public void remove() {
        throw new UnsupportedOperationException(); }
    } // ReverseArrayIterator
} // ArrayStack<T>
```

Pilak:
Zerrenda estekatuetan oinarritutako inplementazioa



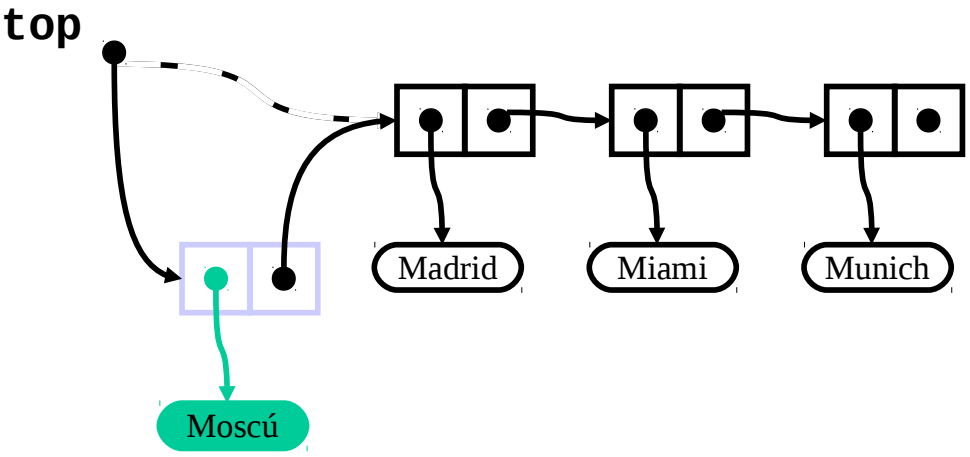
Pilak:
Zerrenda estekatuetan oinarritutako inplementazioa

```
public class LinkedStack<T> implements Stack<T>{
   private LinearNode<T> top;
   private int count;
   public LinkedStack(){ // eraikitzailea
     top = null;
     count = 0;
   }
   public boolean isEmpty( ){
     return (top == null);
   }
   public int size(){ return count; }
```

Pilak:

Zerrenda estekatuetan oinarritutako inplementazioa txertaketa: push()

```
public void push(T e) {
                                                                ;
}
```



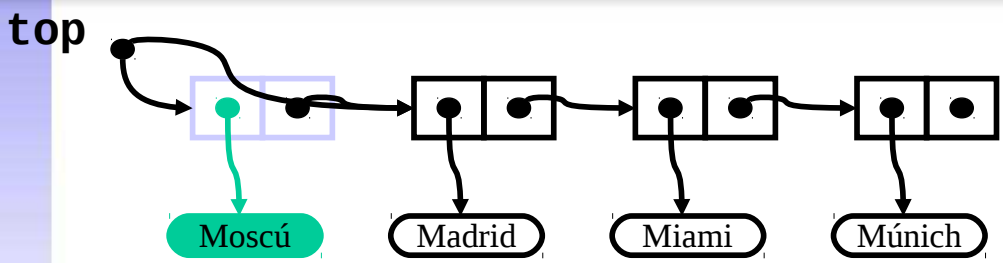
Pilak:

Zerrenda estekatuetan oinarritutako inplementazioa atzipena: peek()

```
public T peek() {
  // Precondition: not empty

}
```

Pilak:

Zerrenda estekatuetan oinarritutako inplementazioa ezabaketa: pop()




```
public T pop(){
   // Precondition: not empty

}
} // LinkedStack
```


Datu egiturak

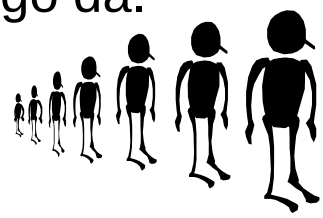
Ilarak

Ilarak
Zer dira? Ezaugarriak

- Datu egitura **lineala** non:
  - Atzipena, hasieran txertatutako elementuari murriztua dago (**elementu zaharrena**)
  - Elementuak mutur batetik txertatzen dira eta beste muturretik ateratzen dira **FIFO** (**F**irst **I**n **F**irst **O**ut) irizpidea jarraituz.

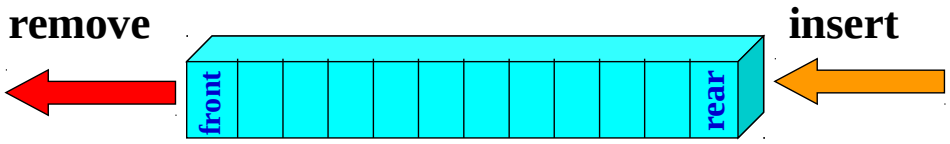
Ilarak:
Adibideak

- **Autobuseko ilara**
  - Lehenengoa sartzen da, lehenengoa etorri dena.
  - Ilaran denbora gehiago daramana
- **Inprimaketa-ilara**
  - Azken bidalitako fitxategia, azkenekoa ateratzen dena izango da.



Ilarak
Metodo nagusiak

- Mutur batetik txertatu (**ilaratu**): **`insert(x)`**
- Beste muturretik atera (**desilaratu**): **`remove()`**



Eskuragarri dago, Ilararen **lehen** elementua (front).

Ilarak
Metodo nagusiak eta metodo laguntzaileak

Metodo nagusiak	*Esanahia*
void **insert**(T elem)	Elementu bat **txertatzen** du ilararen bukaeran.
T **remove**()	Ilararen hasierako elementua **ezabatzen** du eta bere edukia itzultzen du . Hutsa bada, null bueltatuko du.

Metodo laguntzaileak	*Esanahia*
int **size**()	Ilararen **tamaina** itzultzen du
boolean **isEmpty**()	true, ilara **hutsa** bada eta false kontrako kasuan
T **front()**	Ilararen hasieran dagoen elementua itzultzen du ilaratik atera gabe

Mezu baten kodeketa

- Mezuko letra bakoitza aldatuko da, bere ordez, letra horri “dagokion” gakoak emandako alfabetoko distantzian kokatuta dagoen letra jarriz
- Gakoa zenbakien sekuentzia bat da
  - Adibidez, gakoa {3, 15, -4, 7, -9, 5} bada, orduan lehenengo letra 3ko distantzian dagoen letragatik aldatuko da, bigarrena 15eko distantzian eta horrela letra guztiekin. 5a erabili eta gero, lehenengo gaiarekin jarraituko da, mezu oso kodetu arte
  -
- Ikus 5.2 atala [Lewis, Chase] liburuan

Erroaren bidezko ordenazioa (radix sort)

- Zenbakien sekuentzia bat ordenatu behar da (digitu-kopuru maximoa ezagututa)
- Ikus *radix sort* Edozein liburutan (wikipedia)
- Ikus 7.4 atala [Lewis, Chase] liburuan

Erroaren bidezko ordenazioa. Adibidea

- Gehienez 3 digituko zenbaki hamartarrak
- Sekuentzia ez-ordenatua:
  - 170, 45, 75, 90, 2, 24, 802, 66.
  - Edo beste era batera:
  - 170, 045, 075, 090, 002, 024, 802, 066

Lehen fasea

- 170, 045, 075, 090, 002, 024, 802, 066
- Ilaretan sartuko ditugu pisu gutxieneko digituagatik (num%10)
- Berriro bilduko ditugu ilara batean, 0 ilaratik hasita, 9 ilarara iritsi arte:
  - 170, 090, 002, 802, 024, 045, 075, 066

0	17<u>0</u>, 09<u>0</u>
1	
2	00<u>2</u>, 80<u>2</u>
3	
4	02<u>4</u>
5	04<u>5</u>, 07<u>5</u>
6	06<u>6</u>
7	
8	
9	

Bigarren fasea

- 170, 090, 002, 802, 024, 045, 075, 066

  Ilaretan sartuko ditugu pisu gutxieneko bigarren digituagatik ((num/10)%10)

Berriro bilduko ditugu ilara batean, 0 ilaratik hasita, 9 ilarara iritsi arte:

- 002, 802, 024, 045, 066, 170, 075, 090

0	002, 802
1	
2	024
3	
4	045
5	
6	066
7	170, 075
8	
9	090

Hirugarren fasea (eta azkena)

- 002, 802, 024, 045, 066, 170, 075, 090
- Ilaretan sartuko ditugu pisu gutxieneko hirugarren digituagatik ((num/100)%10)

0	002, 024, 045, 066, 075, 090
1	170
2	
3	
4	
5	
6	
7	
8	802
9	

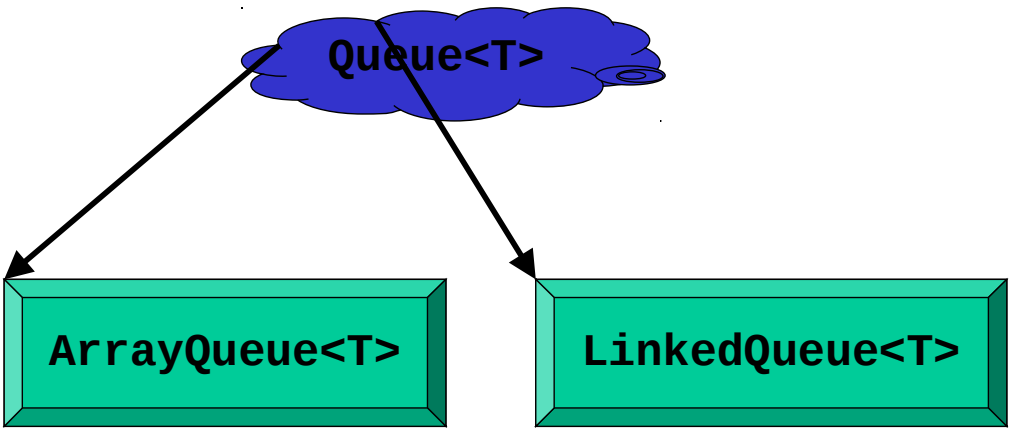
- Berriro bilduko ditugu ilara batean, 0 ilaratik hasita, 9 ilarara iritsi arte:

002, 024, 045, 066, 075, 090, 170, 802

Ilarak
Ilararen interfazea

```
public interface Queue<T>  {
  public int size();
  public boolean isEmpty();
  public void insert(T o)  ;
  public T remove()  ;
  public T front()  ;
}
```


Ilarak:
Inplementazio desberdinak interfaze batentzat



Ilarak:

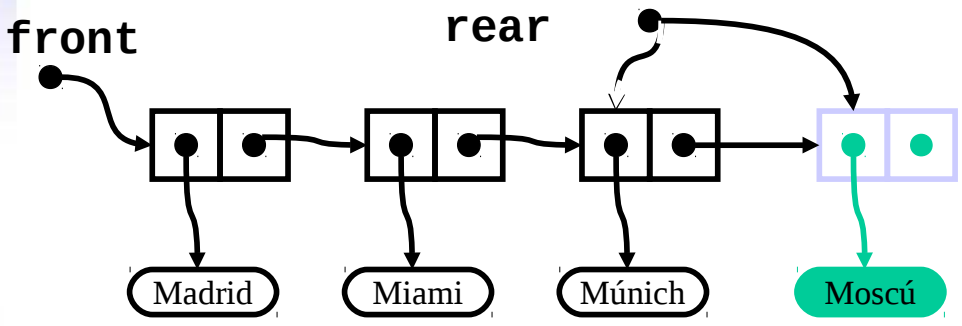
Zerrenda estekatuetan oinarritutako inplementazioa

```
public class LinkedQueue<T> implements Queue<T>{
   private LinearNode<T> front;
   private LinearNode<T> rear;
   private int count;
    public LinkedQueue(){
      front = null;
      rear = null;
      count = 0;
    }
  public boolean isEmpty(){ return front==null;}
  public int size(){ return count;}
```


Ilarak:

Zerrenda estekatuetan oinarritutako inplementazioa
Txertaketa: insert

```
  public void insert(T e) {

  }
```



Ilarak:

Zerrenda estekatuetan oinarritutako inplementazioa
Ezabaketa: remove

```
public T remove(){
  // Precondition: not empty
```

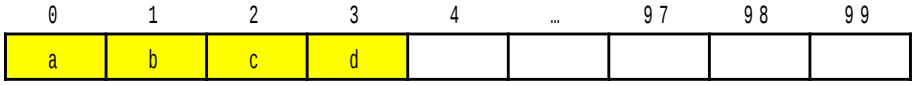
```
}
```


Ilarak:

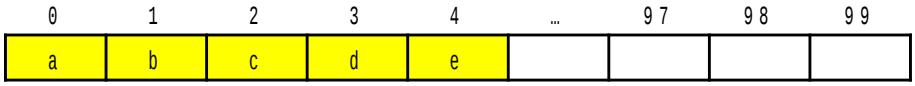
Zerrenda estekatuetan oinarritutako inplementazioa
Hasierako elementua: first

```
public T front(){
  // Precondition: not empty
  return (front.getElement());
}
```

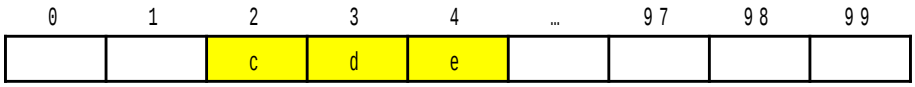
Ilara: arrayen bidezko inplementazioa



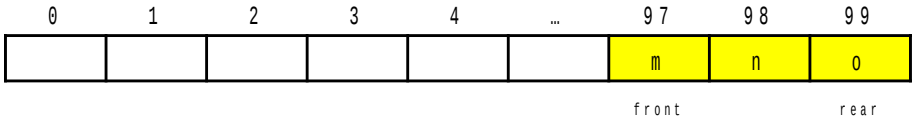
i.insert('e');



i.remove(); i.remove(); // 2 elementu atera

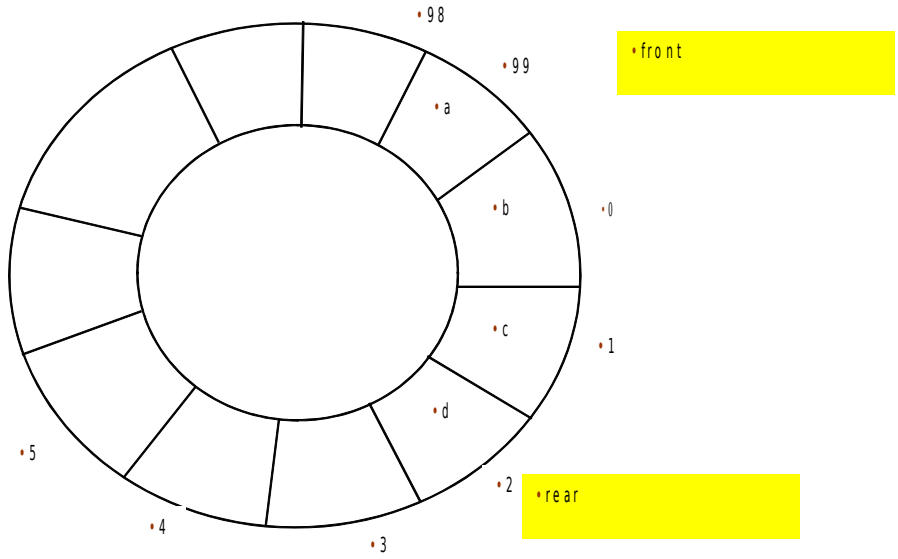


insert O(1) alda? Ezin dira elementu gehiago sartu baina tokia badago!



Ilara: arrayen bidezko inplementazioa

- Lehen soluzioa: elementuak mugitu tokia egiteko => insert O(n) da!!!
- Bigarren soluzioa: "ikusi" arraya zirkularra balitz bezala



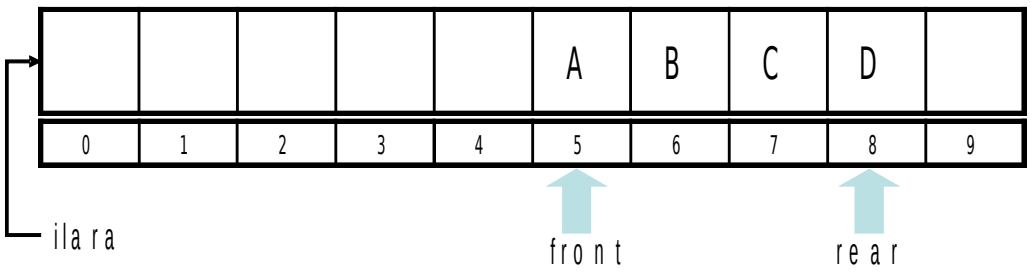
Ilara: arrayen bidezko inplementazioa

- Arazoa: zelan mugitu aurrekoa?

Elementuaren indizea(N)	Hurrengoa
5	6
6	7
...	
98	99
99	0
0	1

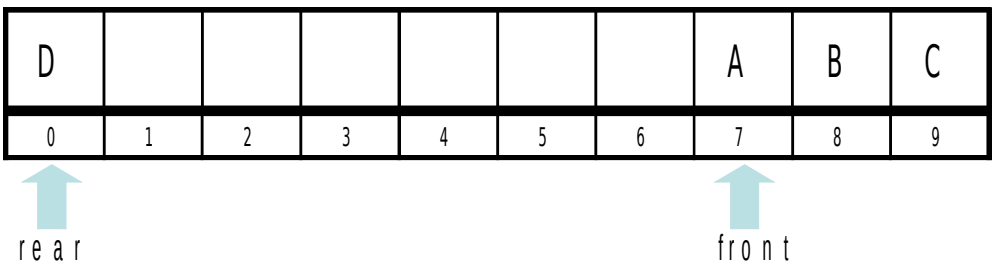
Hurrengoa(N) = (N + 1) % 100

Ilara: arrayen bidezko inplementazio "zirkularra" (0)



nItems = 4

maxSize = 10



Ilara: arrayen bidezko inplementazio “zirkularra” (1)

```
public class ArrayQueue<T> implements Queue<T> {
   private int maxSize;
   private T[] taula;
   private int front;
   private int rear;
   private int nItems;

   public ArrayQueue(int s) {
            maxSize = s;
            taula = (T[])(new Object[maxSize]);
            front = 0;
            rear = -1;
            nItems = 0;
   }
   public void insert(T j)  {
         if(rear == maxSize-1)
            rear = -1;
         taula[++rear] = j;
         nItems++;
   }
```


Ilara: arrayen bidezko inplementazio “zirkularra” (2)

```
  public T remove()  {
   T temp = taula[front++];
   if(front == maxSize)
      front = 0;
   nItems--;
   return temp;
  }

  public T front() { // Precondition: not empty
   return taula[front];
  }

  public boolean isEmpty() { return (nItems==0); }
  public boolean isFull() { return (nItems==maxSize); }
  public int size() { return nItems; }

} // ArrayQueue
```

Interfaze bat
Inplementazio desberdin asko

Egitura Zer? / Nola? Inplementazioa	push pop **Pila**	**Ilara** remove insert
Array		
Vector		
LinkedList null	null	null
DoubleLinkedList null null	null null	null null
Beste egitura linealak		