Representación de datos

Estructuras de Datos: Pilas y Colas

Índice

- ***Pilas***
  - ¿Qué son?
  - Características
  - Métodos principales
  - Ejemplos
- ***Ejemplos de implementación***
  - Basada en arrays
  - Basada en vectores
  - Basada en Listas
- ***Colas***
  - ¿Qué son?
  - Características
  - Métodos principales
  - Ejemplos
- ***Ejemplos de implementación***
  - Basada en arrays
  - Basada en vectores
  - Basada en Listas

Representación de datos

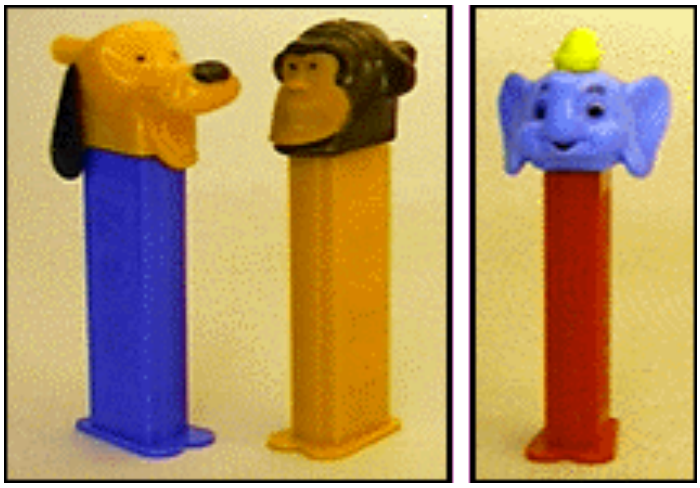
Pilas

Pilas

¿Qué son?. Características

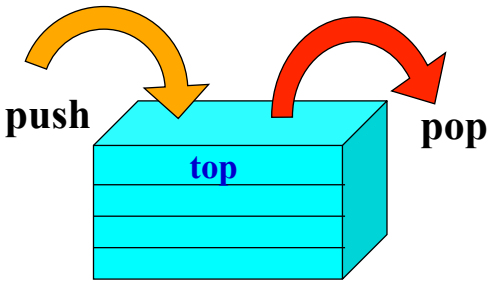
- Estructura de datos **lineal** en la que:
  - Sólo está permitido el acceso al elemento insertado más recientemente (**top** o cima)
  - Los objetos se insertan y eliminan **por un solo extremo** de acuerdo al principio **LIFO** (**L**ast **I**n **F**irst **O**ut)
- Son apropiadas cuando:
  - Sólo necesitamos acceder al **último** elemento
  - Queremos **dar la vuelta** a una lista de elementos

Pilas
Ejemplo



Pilas
Métodos principales

- Meter por un extremo (**apilar**): `push(x)`
- Sacar por el mismo extremo (**desapilar**): `pop()`



El único elemento accesible es el último insertado (**cima**): top

Pilas

Métodos principales y métodos auxiliares

Métodos principales	*Significado*
void **push**(T elem)	**Inserta** un elemento en la cima de la pila
T **pop**()	**Elimina** el elemento en la cima de la pila y devuelve el objeto que contiene

Métodos auxiliares	*Significado*
int **size**()	Devuelve el **tamaño** de la pila
boolean **isEmpty**()	Devuelve true si la pila está **vacía** y false en caso contrario
T **top()**	Devuelve el elemento que hay en la **cima** de la pila sin eliminarlo de la misma.

Iterator<T> **iterator**();	Devuelve un iterador para recorrer todos los elementos de la pila, empezando por la cima

Ilustración/ejemplo

Operación	Contenido Pila	posición TOP
1. S=new Pila	<empty>	0
2. S.push(‘a’)	a	1
3. S.push(‘b’)	a b	2
4. S.push(‘c’)	a b c	3
5. S.pop()	a b	2
6. S.push(‘d’)	a b d	3
7. S.push(‘e’)	a b d e	4
8. S.pop()	a b d	3
9. S.pop()	a b	2
10. S.pop()	a	1

Ejercicios

- Cual será el estado de la pila después de las siguientes operaciones:

  create stack
  push A onto stack
  push F onto stack
  pop item from stack
  push B onto stack
  pop item from stack
  pop item from stack

- Mostrar el estado de la pila y el valor de cada variable después de ejecutar cada una de las siguientes sentencias:

  A = 5 B = 3 C = 7

(a)
create stack
push A onto stack
*push C*C onto stack*
pop item from stack and store in B
push B+A onto stack
pop item from stack and store in A
pop item from stack and store in B

(b)
create stack
push B onto stack
push C onto stack
push A onto stack
*A=B*C*
push A+C onto stack
pop item from stack and store in A
pop item from stack and store in B
pop item from stack and store in C

Pilas
Aplicaciones

- Pilas en los compiladores:
  - Comprobar **equilibrado de símbolos**
- Pilas en los interpretes
  - **Llamadas a métodos**
    - Se guardan:
      - Las variables locales al método
      - La línea donde se hizo la llamada
    - Ej: Recursión
- Pilas para expresiones aritmeticas
  - Conversión entre distintas notaciones
  - Evaluación de expresiones

Pilas
Ejemplo 1: Comprobar paréntesis

- Bien:
  - –
  - – ()
  - – ((()(())))

- Mal:
  - –)(
  - – (()
  - – ())

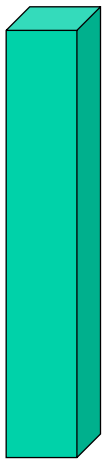
- Reglas para comprobar símbolos equilibrados:
  - Básico: ()
  - Secuenciación: ()()
  - Anidamiento: (())

Pilas
Ejemplo 1: Comprobar paréntesis

- **Reglas**:
  - – Cada vez que nos encontremos (lo metemos en la pila.
  - – Cada vez que nos encontremos) sacamos el (superior de la pila.
  - – La cadena de paréntesis es **correcta**, si la **pila está vacía** al acabar de recorrer toda la cadena.

Pilas

Ejemplo1: Comprobar paréntesis (()(()())())



Reglas:

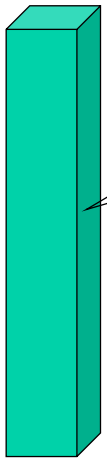
-Paréntesis **abierto**: **apilar**

-Paréntesis **cerrado**: **desapilar**

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

Pilas

Ejemplo1: Comprobar paréntesis (()(()())())



Correcto: Hemos recorrido toda la cadena y la **pila está vacía**

Pilas

Ejemplo2: Recursión (implementación)

```
1  public class PruebaFactorial{
2   public static void main(String args[]){
3    int  param = Integer.parseInt(args[0]);
4    long resultado = fac(param);
5    System.out.println("El factorial de " +
     args[0] +" = " + resultado);
6   }//fin del main
7   public static long fac(int n){
8     if(n<=1)
9       return 1;
10    else
11      return n*fac(n-1);
12  }//fin del fac
13  }//fin de la clase
```

Pila de llamadas a métodos

fact(1)	n = 1	Línea 9
fact(2)	n = 2	Línea 11
fact(3)	n = 3	Línea 11
fact(4)	n = 4	Línea 11
main(param)	param=4	Línea 4

- Cada vez que llamamos a un método guardamos:
  - Las variables locales al método
  - La línea donde se hizo la llamada
- Cuando termina su ejecución sacamos el método de la pila

Pilas

Ejemplo2: Recursión (implementación)

1

fact(1)	n = 1	Línea 9

```
9   return 1;
```

fac(1)=1

fact(2)	n = 2	Línea 11
fact(3)	n = 3	Línea 11
fact(4)	n = 4	Línea 11
main(param)	param=4	Línea 4

- Cada vez que llamamos a un método guardamos:
  - Las variables locales al método
  - La línea donde se hizo la llamada
- Cuando termina su ejecución sacamos el método de la pila

Pilas

Ejemplo2: Recursión (implementación)

2

fact(2)	n = 2	Línea 11

```
11    return n*fac(n-1);
12  }//fin del fac
```

2*fac(1)

fac(2) =2*fac(1)=2

fac(1)=1

fact(3)	n = 3	Línea 11
fact(4)	n = 4	Línea 11
main(param)	param=4	Línea 4

•Cada vez que llamamos a un método guardamos:
  –Las variables locales al método
  –La línea donde se hizo la llamada
•Cuando termina su ejecución sacamos el método de la pila

Pilas

Ejemplo2: Recursión (implementación)

6

fact(3)	n = 3	Línea 11

```
11    return n*fac(n-1);
12  }//fin del fac
```

3*fac(2)

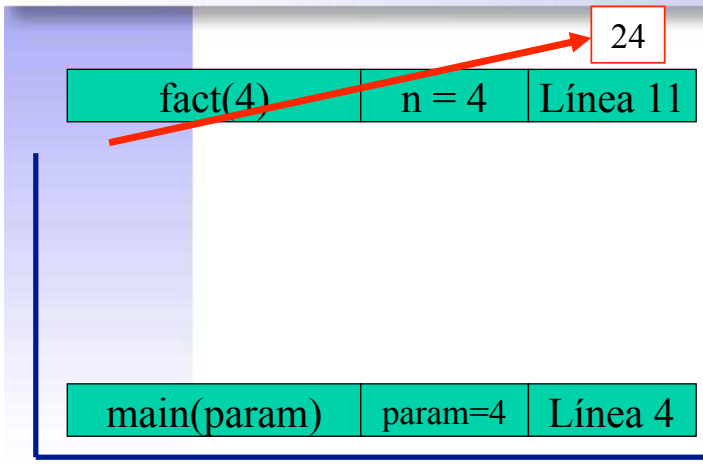
fac(3) = 3*fac(2)=6

fac(2)=2
fac(1)=1

fact(4)	n = 4	Línea 11
main(param)	param=4	Línea 4

•Cada vez que llamamos a un método guardamos:
  –Las variables locales al método
  –La línea donde se hizo la llamada
•Cuando termina su ejecución sacamos el método de la pila

Pilas
Ejemplo2: Recursión (implementación)




```
11    return n*fac(n-1);
12  }//fin del fac
```

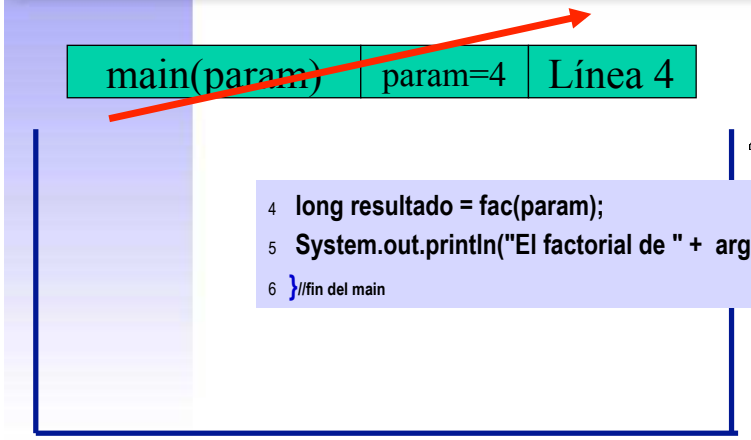
fac(4) = 4*fac(3)=24

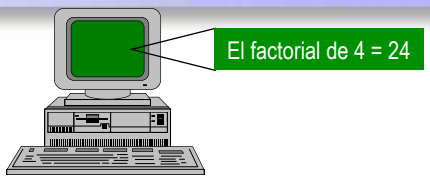
4*fac(3)

fac(3)=6
fac(2)=2
fac(1)=1

- Cada vez que llamamos a un método guardamos:
  - Las variables locales al método
  - La línea donde se hizo la llamada
- Cuando termina su ejecución sacamos el método de la pila

Pilas
Ejemplo2: Recursión (implementación)






```
4  long resultado = fac(param);
5  System.out.println("El factorial de " +  args[0] +" = " + resultado);
6  }//fin del main
```

resultado = 24

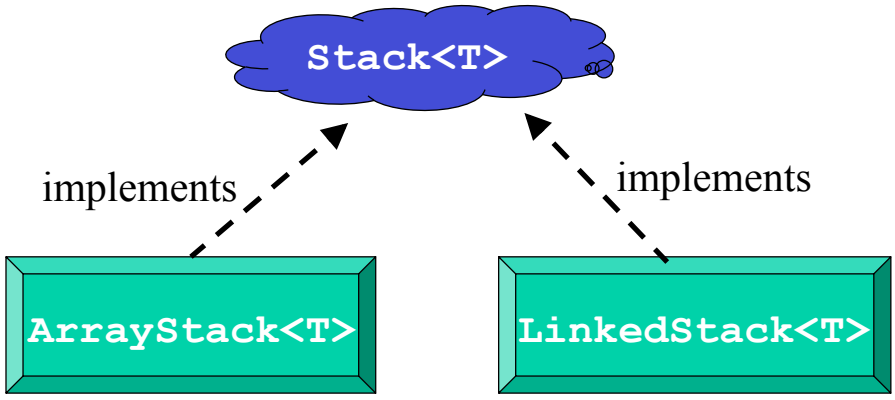
fac(4)=24
fac(3)=6
fac(2)=2
fac(1)=1

- Cada vez que llamamos a un método guardamos:
  - Las variables locales al método
  - La línea donde se hizo la llamada
- Cuando termina su ejecución sacamos el método de la pila

Interfaz para pilas

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


Pilas:
Una interfaz y varias implementaciones

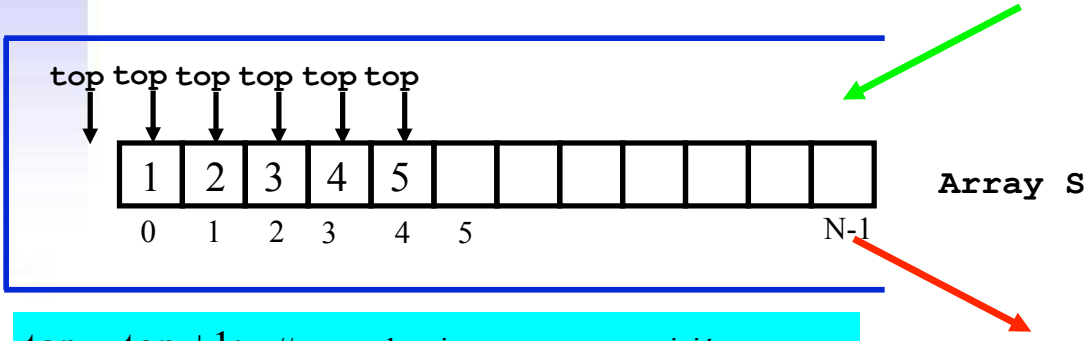


OJO!: No corresponden con Java, java.util.Stack es una clase

Pilas:
Implementación basada en Arrays

```
Stack<Integer> pila = new ArrayStack<Integer>();
for (int i = 1; i < 5; i++)
        pila.push(new Integer(i));
```

Queremos insertar un objeto **obj** en la **pila** implementada por el array S




```
top = top +1;   //muevo la etiqueta top una posición
s[top]= obj;    //inserto el elemento nuevo en la posición top
```


Pilas:
Implementación basada en Arrays

```
public class ArrayStack<T> implements Stack<T> {
  public static final int CAP=1000;
  private int capacity;
  private T S[];
  private int top = -1;

  public ArrayStack(){this(CAP);}

  public ArrayStack(int cap){
     capacity = cap;
    S = new T[capacity];
  }
```

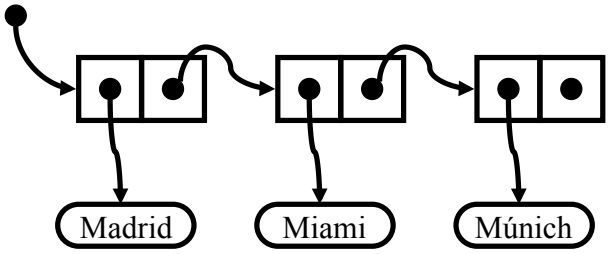
Pilas:
Implementación basada en Arrays

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

public boolean isEmpty() {
   return (top < 0);
}

public T top() throws StackEmptyException {
   if (isEmpty())
     throw new StackEmptyException("vacio"};
   return S[top];
}
```


Pilas:
Implementación basada en Arrays

```
public void push(T obj) throws StackFullException {
  if (size() == capacity)
      throw new StackFullException("lleno");
   S[++top] = obj; //primero incremento y luego asignación
}

public T pop() throws StackEmptyException {
   T obj;
   if (isEmpty())
      throw new StackEmptyException("vacio"};
   obj = S[top];
   S[top--] = null; //primero asignación y luego decremento
   return obj;
}
}//fin de la clase
```

Pilas:
Implementación basada en Listas enlazadas

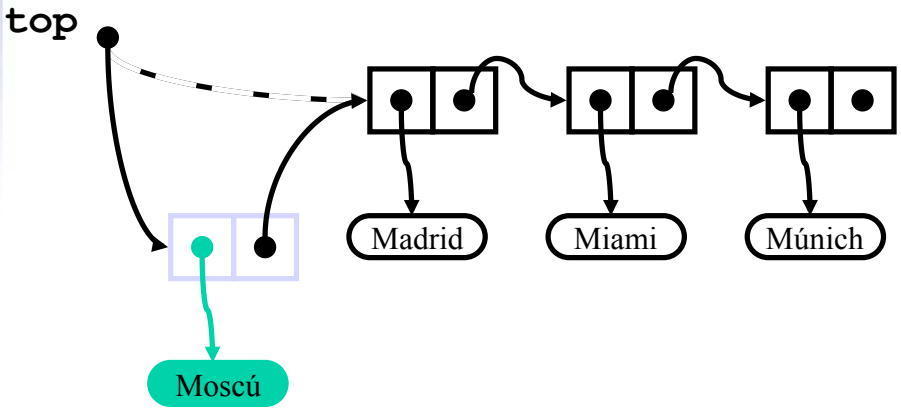


Pilas:
Implementación basada en Listas enlazadas

```
public class LinkedStack<T> implements Stack<T> {
  private LinkedList<T> linkedList;

  public LinkedStack() {
     linkedList = new LinkedList<T>();
  }

  public int size() {
      return linkedList.size()
  }

  public boolean isEmpty() {
      return(size()==0);
  }
```

Pilas:
Implementación basada en Listas enlazadas
Inserción: push()

```
public void push(T e) {
  linkedList.insertFirst(e);
}
```

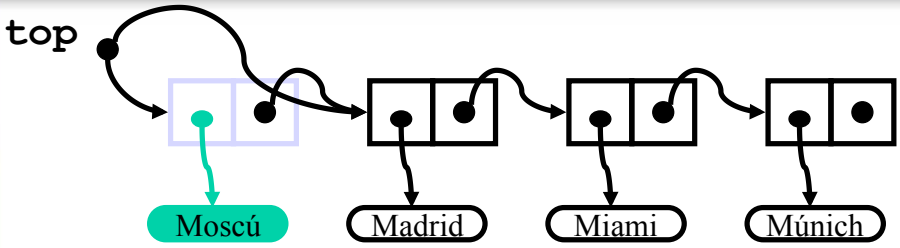


Pilas:
Implementación basada en Listas enlazadas

```
public T top() throws StackEmptyException{
 if (isEmpty())
   throw new StackEmptyException("vacia");
 else {
   LinkeListIterator<T> itrList =
                          linkedList.first();
   return itrList.getCurrent();
  }
}
```

Pilas:

Implementación basada en Listas enlazadas.
Eliminación: pop




```
public T pop() throws StackEmptyException{
  if (isEmpty())
    throw new StackEmptyException("vacia");
  else {
    LinkeListIterator<T> itrList =
                              linkedList.first();
    T temp = itList.getCurrent();
    itrList.remove();
    return temp;
}
```

Pilas: Aplicación expresiones aritméticas

- Notaciones:
  - Infix (infija): (A+B) – (C*D)
  - Postfix (postfija): A B + C D * -
  - Prefix (prefija): - + A B * C D
- Problemas:
  - Conversiones entre notaciones
  - Evaluaciones de expresiones
- IMPORTANTE: sobre los operadores:
  - 1º ejecutar los de menor **prioridad**
    - En la expresión: paréntesis, ^, * ó /, + ó -,
    - En la pila: ^, * ó /, + ó -, paréntesis

    OJO! Es distinta según el lugar dónde aparece
  - y orden de **precedencia** (lo mantiene la pila)

Pilas: Aplicación
expresiones aritméticas (algoritmos)

Converting between notations:

- INFIX to PREFIX : (A+B) – (C*D)
  - Do the first brace: (A+B), the PREFIX is +AB
  - Do the second brace: (C*D), the PREFIX is *CD
  - The end is operator -: +AB - *CD, the PREFIX is -+AB*CD
- INFIX to POSTFIX : (A+B) – (C*D)
  - Do the first brace: (A+B), the POSTFIX is AB+
  - Do the second brace: (C*D), the POSTFIX is CD*
  - The end is operator -: AB+ - CD*, the PREFIX is AB+CD*-
- PREFIX to INFIX : +/*A B C D
  - Find the first operator: *, take 2 operands after the operator (**A** and **B**), the INFIX is (A*B)
  - Find the second operator: /, take 2 operands after the operator (**A*B** and **C**), the INFIX is ((A*B)/C)
  - Find the third operator: +, take 2 operands after the operator (**((A*B)/C)** and **D**), the INFIX is ((A*B)/C)+D

Converting between notations(2)

- PREFIX to POSTFIX : +/*A B C D
  - Find the first operator: *, take 2 operands after the operator (**A** and **B**), the POSTFIX is AB*
  - Find the second operator: /, take 2 operands after the operator (**AB*** and **C**), the POSTFIX is AB*C/
  - Find the third operator: +, take 2 operands after the operator (**AB*C/** and **D**), the POSTFIX is AB*C/D+
- POSTFIX to INFIX : ABCD*/-
  - Find the first operator: *, take 2 operands before the operator (**C** and **D**), the INFIX is (C*D)
  - Find the second operator: /, take 2 operands before the operator (**(C*D)** and **B**), the INFIX is (B/(C*D)
  - Find the third operator: -, take 2 operands before the operator (**(B/(C*D)** and **A**), the INFIX is A – (B/(C*D)
- POSTFIX to PREFIX : ABCD*/-
  - Find the first operator: *, take 2 operands before the operator (**C** and **D**), the PREFIX is *CD
  - Find the second operator: /, take 2 operands before the operator (***CD** and **B**), the PREFIX is /B*CD
  - Find the third operator: -, take 2 operands before the operator (**/B*CD** and **A**), the PREFIX is -A /B*CD

Exercise: Converting

1. Convert these INFIX to PREFIX and POSTFIX :
   a) A / B – C / D
   b) (A + B) ^ 3 – C * D
   c) A ^ (B + C)

2. Convert these PREFIX to INFIX and POSTFIX :
   a) + – / A B C ^ D E
   b) – + D E / X Y
   c) ^ + 2 3 – C D

3. Convert these POSTFIX to INFIX and PREFIX :
   a) A B C + –
   b) G H + I J / *
   c) A B ^ C D + –

Representación de datos

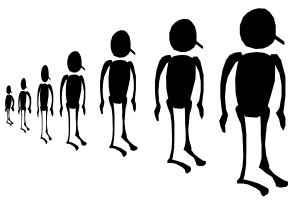
Colas

Colas
¿Qué son?. Características

- Estructura de datos **lineal** en la que:
  - El acceso está restringido al elemento insertado al principio (**el más antiguo**)
  - Los objetos se insertan por un extremo y se eliminan por otro de acuerdo al principio **FIFO** (**F**irst **I**n **F**irst **O**ut)

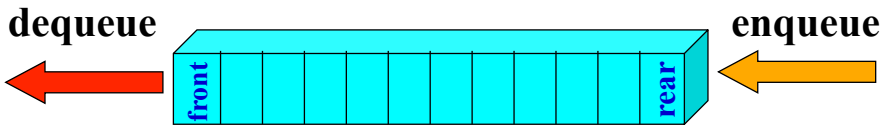
Colas:
Ejemplos

- La **cola del autobús**
  - El primero en entrar al autobús será el primero que llegó
  - El que lleva más tiempo en la cola
- La **cola de la impresora**
  - El último trabajo enviado a la impresora será el último en salir.



Colas
Métodos principales

- Meter por un extremo (**encolar**): `enqueue(x)`
- Sacar por el extremo contrario (**desencolar**): `dequeue()`



Son accesibles tanto el **primer** elemento de la cola (**front**) como el **último** (**rear**)

Colas
Métodos principales y métodos auxiliares

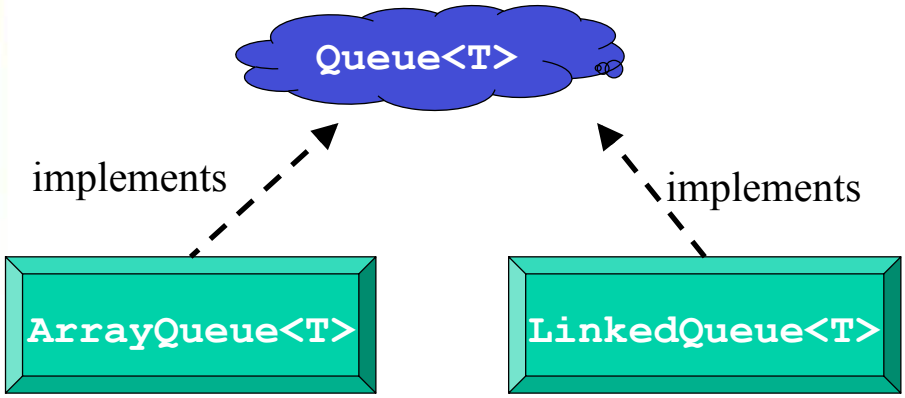
Métodos principales	*Significado*
void **enqueue**(T elem)	**Inserta** un elemento al final de la cola
T **dequeue**()	**Elimina** un elemento al principio de la cola y devuelve su contenido

Métodos auxiliares	*Significado*
int **size**()	Devuelve el **tamaño** de la cola
boolean **isEmpty**()	Devuelve true si la cola está **vacía** y false en caso contrario
T **front()**	Devuelve el elemento que hay al principio de la cola sin eliminarlo de la misma.
void **clear()**	Elimina toda la información

Colas
Interfaz para colas

```
public interface Queue<T>  {
  public int size();
  public boolean isEmpty();
  public void enqueue(T o);
  public T dequeue() throws QueueEmptyException;
  public T front()  throws QueueEmptyException;
}
```


Colas:
Una interfaz y varias implementaciones

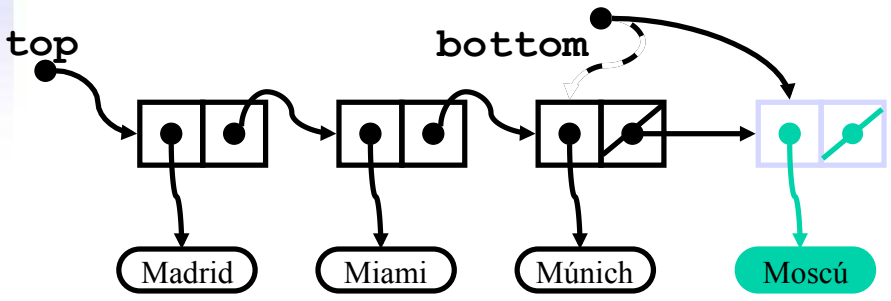


Colas:
Implementación basada en Listas enlazadas

```
public class LinkedQueue<T>
             implements Queue<T>{
 private LinkedList<T> linkedList;
 public LinkedQueue() {
    linkedList = new LinkedList<T>();
 }
 public int size() {
      return linkedList.size()
 }
 public boolean isEmpty() {
  return(size() == 0);
 }
```


Colas:
Implementación basada en Listas enlazadas.
Inserción: enqueue

```
public void enqueue(T e) {
  linkedList.insertLast(e);
}
```



Colas:

Implementación basada en Listas enlazadas.
Eliminación: dequeue

```
public T dequeue() throws QueueEmptyException{
 T temp;
 if (isEmpty())
  throw new QueueEmptyException("vacia");
 else {
    LinkeListIterator<T> itrList =
                linkedList.first();
    temp = itList.getCurrent();
    itrList.remove();
    return temp;
  }
}
```


Una interfaz
Varias implementaciones

Estructura / ¿Qué? — ¿Cómo? Implementación	Pila (push, pop)	Cola (dequeue, enqueue)
Array		
ArrayList		
LinkedList (null)		
DoublyLinkedList (null)		
Otras estructuras de datos lineales		

Java

- La clase `LinkedList` es la que implementa el interfaz `Queue`

```
public interface Queue<T> extends
  Collection<T> {
    T element();
    boolean offer(T o);
    T peek();
    T poll();
    T remove();
}
```


Method Summary	
T	element() Retrieves, but does not remove, the head of this queue, it throws an exception if this queue is empty..
boolean	offer(T o) Inserts the specified element into this queue, if possible.
T	peek() Retrieves, but does not remove, the head of this queue, returning null if this queue is empty.
T	poll() Retrieves and removes the head of this queue, or null if this queue is empty.
T	remove() Retrieves and removes the head of this queue.

Otras estructuras de datos

- `ColaDoble` (BiCola): es parecido a una cola, con la diferencia que permite añadir y eliminar datos desde los dos extremos
- Cola con prioridad (`PriorityQueue`): el primer dato que se recupera es el de mayor prioridad (el entero que menor valor tiene)