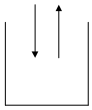
Problema.
Leer las líneas de un archivo y escribirlas al revés (1º la última, 2º la penúltima, ..., última la 1ª).

Indicación.
- Use el TDA (Tipo de Dato Abstracto) Stack (Pila)
- Stack: contenedor que opera con el principio LIFO (Last In First Out), es decir, el último elemento que ingresa es el primero que sale.



TDA: tipo de dato con implementación oculta (representación privada y operaciones públicas)

TDA Stack

Operación	significado	explicación
new Stack()		crear stack vacío
void push(Object x) throws StackFull	x 	Poner x en stack. Si está lleno produce la excepción StackFull
Object pop( ) throws StackEmpty		Sacar un valor de stack. Si está vacío produce la excepción StackEmpty
void reset( )		vaciar stack
boolean empty( )	¿  ?	true si stack está vacío
boolean full( )	¿  ?	true si stack está lleno

```
static public void main(String[]args)
throws Exception
{
//crear stack para mantener líneas
Stack s=new Stack();

//leer líneas y ponerlas en stack
BR A=new BR(new FR(args[0]));
String linea;
while((linea=A.readLine())!=null){
  if(s.full()) U.abortar("demasiadas lineas");
  s.push(linea);
}

//sacar y escribir líneas de stack
while( !s.empty() )
  U.println((String)s.pop());//Object a String
}
```

Solución 2: atrapando las excepciones ("el golpe avisa")

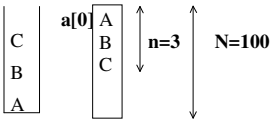
```
Stack s=new Stack();
try{
  BR A=new BR(new FR(args[0]));
  String linea;
   while((linea=A.readLine())!=null)
     s.push(linea);
}catch(StackFull x){
   U.abortar("demasiadas líneas");
}
try{
   while(true)
    U.println((String)s.pop());
}catch(StackEmpty x){}//no hacer nada
```

Sintaxis: `try{…}catch(Nombre x){…}`

"catcher" (atrapador de excepción)
- se ejecuta sólo si en el bloque try ocurrió la excepción nombrada
- cuando termina se sigue con las instrucciones sgtes al catcher.

TDA Stack: implementación con un arreglo

Ejemplo:




```
class Stack
{
protected Object[]a;
protected int n;
protected static final int N=100;
//static: dato global
//(para todos los objetos de la clase)
```

```
  public Stack(){
     a = new Object[N];
     n = 0;
  }
  public boolean empty(){
     return n == 0;
  }
  public boolean full(){
     return n >= N;
  }
  public void reset(){
     a = new Object[N];
     n = 0;
  }
```

```
public void push(Object x)throws StackFull
{
   if( n >= N ) throw new StackFull();
   a[n++] = x; //a[n]=x; ++n;
}
public Object pop()throws StackEmpty
{
   if( n == 0 ) throw new StackEmpty();
   return a[--n];//--n; return a[n];
}
}//fin class Stack
```

throws Nombre: indica que si en el método se produce la excepción indicada, entonces no se atrapará y se trasmitirá/propagará al método que lo invocó (que a su vez la podrá atrapar o propagar)

throw new Nombre(): lanza/produce/genera la excepción indicada

Definición de excepciones

```
class StackFull extends Exception{}
class StackEmpty extends Exception{}
```

Exception

- clase predefinida
- clase madre de todas las excepciones
- excepciones de Input/Output:
  class IOException extends Exception{…}
  class FileNoutFoundException extends IOException{…}
  etc
- excepciones al ejecutar programas:
  class RunTimeException extends Exception{…}
  class NullPointerException extens RunTimeException{…}
  class ArrayIndexOutOfBoundsException extends RunTimeException{…}
  etc

Alternativa2: atrapando excepciones predefinidas

```
public void push(Object x)throws StackFull{
try{
   a[n++] = x;
}catch(ArrayIndexOutOfBoundsException x){
   throw new StackFull();
}
}
public Object pop()throws StackEmpty{
try{
   return a[--n];
}catch(ArrayIndexOutOfBoundsException x){
   n=0;  //estaba con -1
   throw new StackEmpty();
}
}
}//fin class Stack
```

Problema. Simular una cola de atención de clientes de acuerdo al siguiente diálogo:
Eventos:
1=llegada de cliente
2=atención de cliente
0=fin de eventos

evento? 1		
cliente? A		A
evento? 1		
cliente? B		A B
evento? 2		
se atiende A	A ←	B
evento? 1		
cliente? C		B C
…		
evento? 0		

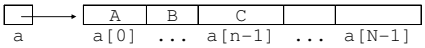
Indicación. Use una Queue (Cola) que es un contenedor que opera con el principio FIFO (First In First Out), es decir, el primer elemento que ingresa es el primero que sale.

Operación	significado
new Queue()	
void enque(Object x) throws QueueFull	← x
Object deque( ) throws QueueEmpty	←
void reset( )	
boolean empty( )	¿ ?
boolean full( )	¿ ?

```
Queue q=new Queue();
char evento;
while((evento=U.readChar("evento?"))!='0'){
  if(evento=='1')
    try{
      q.enque( U.readLine("cliente?") );
    }catch(QueueFull x){
      U.println("rechazado");
    }
  else if(evento=='2')
    try{
      U.println("se atiende "+(String)q.deque());
    }catch(QueueEmpty x){
      U.println("no hay clientes");
    }
  else
    U.println("evento incorrecto");
}
```

TDA Queue: implementación con un arreglo

Ejemplo: ← A B C ←




```
class Queue{
protected Object[]a;
protected int n;
protected static final int N=100;

public Queue(){reset();}
public void reset(){a=new Object[N];n=0;}
public boolean empty(){return n==0;}
public boolean full(){return n>=N;}
```

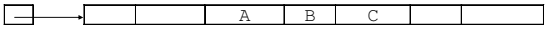
```
public void enque(Object x)throws QueueFull{
  if(n>=N) throw new QueueFull();
  a[n++] = x;
}
public Object deque()throws QueueEmpty{
   if(n==0) throw new QueueEmpty();
   Object aux=a[0];
  for(int i=0;i<n-1;++i)
     a[i]=a[i+1];
   --n;
  return aux;
}
}//fin class Queue
```

deque: O(n), es decir, depende del tamaño de la cola

Implementación TDA Queue: cola "flotante" en un arreglo

Ejemplo: ← A B C ←

enque y deque O(1)




```
class Queue
{
protected Object[]a;
protected int n; //largo de la cola
protected int ip, iu;//indices de 1° y último
protected static final int N=100;
```

```
public Queue(){
   reset();
}
public boolean empty(){
   return n == 0;
}
public boolean full(){
   return n >= N;
}
public void reset(){
   a=new Object[N];
   n=0;
   ip=0; //índice de primer objeto que llegue
   iu=-1;//para que primer enque le sume 1
}
```

```
public void enque(Object x)throws QueueFull{
   if( n >= N ) throw new QueueFull();
  iu=(iu+1) % N;//++iu; if(iu==N) iu=0;
   a[iu] = x;
   ++n;
}
public Object deque()throws QueueEmpty{
   if( n == 0 ) throw new QueueEmpty();
   Object aux = a[ip];
   ip = (ip + 1) % N;
   --n;
   return aux;
}
}//fin class Queue
class QueueFull extends Exception{}
class QueueEmpty extends Exception{}
```

Propuesto: implementar usando el arreglo y sólo dos enteros

Implementar TDA Diccionario: operaciones con lista de palabras y significados, sin repetición de palabras

Operación	Significado
Object buscar (Comparable x)	devolver significado de palabra x (null si no existe)
boolean agregar (Comparable x,Object y) throws DiccLleno	agregar palabra x con significado y (false si ya existe). Excepción DiccLleno si no hay espacio
boolean borrar (Comparable x)	borrar palabra x (devolver false si no existe)
boolean cambiar (Comparable x,Object y)	cambiar significado de palabra x por y (false si no existe)