# Soluciones Ejercicios Tema 7

Germán Moltó Martínez
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia



## Cambio de Signo Pila (1/3)

```
public class EjerciciosPila {
 ….
public static void cambiaSignoPila(Pila<Integer> p){
  if (!p.esVacia()) {
     int dato = p.desapilar().intValue();
     cambiaSignoPila(p);
     p.apilar(new Integer(-dato));
 }
…
}
```



## Cambio de Signo Pila (2/3)

- Si queremos que la operación se ejecute sobre una instancia de Pila, debemos añadir la operación en el modelo y luego implementarla.

```
package modelos;
public interface Pila<E> {
        void apilar(E x);
        E desapilar();
        E tope();
        boolean esVacia();
        void cambiaSigno();
    }
```



## Cambio de Signo Pila (3/3)

```
public class ArrayPilaInteger  implements Pila<Integer>{
…
public void cambiaSigno(){
  if (!esVacia()) {
     int dato = this.desapilar().intValue();
     cambiaSigno();
     this.apilar(new Integer(-dato));
  }
…
}
```

## Pila Sombrero

```
public interface Pila<E> {
        void apilar(E x);
        E desapilar();
        E tope();
        boolean esVacia();
        boolean esSombrero(Pila<E> p);
    }
```

- En la clase que implemente la interfaz Pila añadiremos:

```
public boolean esSombrero(Pila <E> p){
    if (p.esVacia()) return true;
    else {
      E pDato = p.desapilar();
      if (!pDato.equals(this.desapilar())) return false;
      else return esSombrero(p);
     }}}
```

¿Qué problema tiene esta implementación?



## Pila Sombrero (Versión No Destructiva)

```
public boolean esSombrero(Pila<E> p){
    if (p.esVacia()) return true;
    else {
      E Dato = p.desapilar();
      E thisDato = this.desapilar();
      if (!pDato.equals(thisDato)) {
        this.apilar(thisDato);
          p.apilar(pDato);
        return false;
       }
      else {
        boolean loEs = esSombrero(p);
        this.apilar(thisDato);
          p.apilar(pDato);
        return loEs;
     }}}
```



## Inversión de una Cola

```
public static <E> void invierteCola (Cola<E> c){
  if (!c.esVacia()){
     E primero = c.desencolar();
     invierteCola(c);
     c.encolar(primero);
   }
}
```



## Pila2Cola (1/2)

```
package modelos;
public interface Pila<E> {
        void apilar(E x);
        E desapilar();
        E tope();
        boolean esVacia();
        Cola<E> toCola();
    }
```

- En la implementación de la Pila:

```
public class ArrayPila<E> implements Pila<E>{
  …
 public Cola<E> toCola(){
     ….
  }}
```

## Pila2Cola (2/2)

```
private void toCola(Cola<E> c){
      if (!this.esVacia()) {
        E prim  = this.desapilar();
        toCola(c);
        c.encolar(prim);
        apilar(prim);
    }

    public Cola<E> toCola(){
       Cola<E> c = new ArrayCola<E>();
       toCola(c);
       return c;
     }
```



## Inversión de Pila 1/2

```
public static <E> Pila<E> invierteePila(Pila<E> p){
  Pila<E> nuevaPila = new ArrayPila<E>();
  while (!p.esVacia()){
    nuevaPila.apilar(p.desapilar());
  }
  return nuevaPila;
}
```

- Fíjate en la utilización de la genericidad que se ha hecho en el método.



## Inversión de Pila 2/2

```
public class TestEjerciciosPila {
  public static void main (String args[]) {
     Pila<Integer> p = new ArrayPila<Integer>();
     for (int i = 0 ; i < 10 ; i++) p.apilar(new Integer(i));
     System.out.println("Pila Original:" + p);
     Pila<Integer> p1 = EjerciciosPila.invierteePila(p);
     System.out.println("Pila Invertida:" + p1);
  }
```



## Cálculo de Talla de Pila

- Cálculo de la talla de una Pila. Añadimos la operación: int talla() a la interfaz Pila.
- Implementamos el método en la clase ArrayPila.

```
public int talla(){
      int nelem = 0;
      if (!esVacia()) {
        E x = desapilar();
        nelem = talla() + 1;
        apilar(x);
       }
      return nelem;
  }
```

## Ampliando Funcionalidad de ListaConPI

```
public void localiza(E x){
      boolean enc = false; inicio();
      while (!esFin() && !enc){
        E dato = recuperar();
        if (dato.equals(x)) enc = true;
        else siguiente();
}
```

```
public void vaciar(){
    inicio();
    while (!esFin()) eliminar();
   }
```

```
public int talla(){
      int nelem = 0;
      inicio();
      while (!esFin()) {nelem++; siguiente();}
      return nelem;
    }
```



## Lista con Punto de Interés: Anterior (1/2)

- Añadimos los siguientes métodos:
  - void **anterior**(); → Situa el PI sobre el elemento anterior al que apunta actualmente. Esta operación solo puede ejecutarse si !esInicio()).
  - boolean **esInicio**(); → Devuelve true si el PI está situado antes del inicio de la lista, es decir, antes de su primer elemento. Sirve como condición de parada al realizar la iteración descendente.



## Lista con Punto de Interés: Anterior (2/2)

```
public String toString(){
    String res="";
    this.fin();
    this.anterior();
     while (!this.esInicio()){
          E elem = this.recuperar();
          this.anterior();
          res += elem.toString() + " ";
      }
 return res;
}
```



## Lista con Punto de Interés: eliminarTodos

```
public void eliminarTodos(E x) throws ElementoNoEncontrado {
   boolean estaX = false;
   lpi.inicio();
   while (!lpi.esFin() ) {
         if ( lpi.recuperar().equals(x) ) {
               lpi.eliminar();
               estaX = true;
         } else lpi.siguiente();
    }
   if ( !estaX ) throw new ElementoNoEncontrado("Al
   eliminarTodos:        el Dato "+x+" no está");
}
```

• Únicamente hay que desplazar el PI si NO hemos realizado el borrado.

## Lista con Punto de Interés: Lista Ordenada

```
public void insertarOrdenado(E x){
   lpi.inicio();
   while (!lpi.esfin() && lpi.recuperar().compareTo(x) < 0 ) {
         lpi.siguiente();
   }
   lpi.insertar(x);
}
```



## Desapilar Todos Rango (1/2)

1. Implementar la interfaz PilaExtendida

```
package modelos;
public interface PilaExtendida<E extends Comparable<E>>
    extends Pila<E> {
    void desapilarTodosRango(E x, E y);
}
```



## Desapilar Todos Rango (2/2)

2. Implementar la clase *ArrayPilaExtendida* utilizando única y exclusivamente los métodos de la interfaz *Pila*

```
package lineales;
import modelos.*;
public class ArrayPilaExtendida<E extends Comparable<E>> extends
    ArrayPila<E> implements PilaExtendida<E> {
public void desapilarTodosRango(E x, E y){
     if (!esVacia()){
       E dato = desapilar();
       desapilarTodosRango(x,y);
       if ( (dato.compareTo(x) < 0) || (dato.compareTo(y) > 0) ){
         apilar(dato);
        }
     }}}
```



## Navegador de Internet (1/2)

```
public static int pagWebVisita(Pila<PaginaWeb> p, PaginaWeb w) {
int resultado;
if (p.esvacia()) resultado=-1;
  else if(p.tope().equals(w)) resultado=0;
        else {
          PaginaWeb aux=p.desapilar();
          resultado = pagWebVisita(p,w);
          if (resultado!=-1) resultado++;
          p.apilar(aux);
        }
return resultado;
}
```

## Navegador de Internet (2/2)

- Talla del Problema:
  - Número de páginas web en la Pila.
- Instancias Significativas:
  - Caso Mejor: La página web buscada es la más recientemente visitada, es decir, que está situada en el tope de la pila.
  - Caso Peor: La página web buscada nunca fue visitada y, por lo tanto, no está en la Pila.
- Ecuaciones de Recurrencia (se omiten los casos base):
  - $T^M_{pagWebVisitada}(talla>0) = k_1$
  - $T^P_{pagWebVisitada}(talla>0) = T^P_{pagWebVisitada}(talla - 1) + k$
- Cotas de Complejidad
  - $T^M_{pagWebVisitada}(talla) \in \Theta(1)$
  - $T^P_{pagWebVisitada}(talla) \in \Theta(talla)$
  - $T_{pagWebVisitada}(talla) \in \Omega(1), T_{pagWebVisitada}(talla) \in O(talla)$



## Implementación de Pila usando ListaConPI (1/2)

```
public class PilaListaConPI<E> implements Pila<E> {
  private ListaConPI<E> lcpi;

  public PilaListaConPI(){
    lcpi = new LEGListaConPI<E>();
  }
  public void apilar (E x){
    lcpi.insertar(x);
    lcpi.inicio();
  }
  public E desapilar (){
    E tope = lcpi.recuperar();
    lcpi.eliminar();
    return tope;
    }
```



## Implementación de Pila usando ListaConPI (2/2)

```
public E tope(){
      return lcpi.recuperar();
   }
public boolean esVacia(){
      return lcpi.esVacia();
   }
   public String toString(){
     String info="";
     while (!lcpi.esFin()){
        info += lcpi.recuperar() + " ";
        lcpi.siguiente();
     }
     return info;
    }
}
```



## Cancelar en Cola

```
public void cancelar(E x) {
  boolean fin = false;
  this.encolar(null);
  while ( !fin ) {
    E  aux = this.desencolar();
    if ( aux == null ) fin = true;
    else if ( !aux.equals(x) ) this.encolar(aux);
  }
}
```

## Solución Borrar Iguales en Pila

```
public int  eliminarIguales(E d) {
   int cont = 0;
   if ( !this.esVacia() ) {
      E datoPila = this.desapilar();
      cont += this.eliminarIguales(d);
      if (!datoPila.equals(d) ) this.apilar(d);
      else cont ++;
   }
   return cont;
}
```



## Pasar Mayores de Pila a Cola (I)

```
public static <E extends Comparable<E>> Cola<E>
   pasarMayoresQue(E x, Pila<E> p){

      Cola<E> q = new ArrayCola<E>();
      pasarMayoresQue(x,p,q);
      return q;
   }
```



## Pasar Mayores de Pila a Cola (II)

```
public static <E extends Comparable<E>> void
   pasarMayoresQue(E x, Pila<E> p, Cola<E> q){
       if (!p.esVacia()){
          E aux = p.desapilar();
          pasarMayoresQue(x, p,q);
          if (aux.compareTo(x) > 0) q.encolar(aux);
          else p.apilar(aux);
       }
    }
```