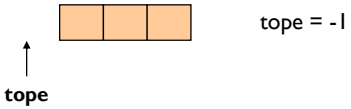
Soluciones Ejercicios Tema 8

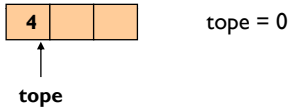
Germán Moltó Martínez
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Traza Pila (1/2)

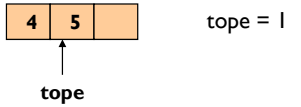
- Pila<Integer> p = new ArrayPila<Integer>();



- p.apilar(new Integer(4));

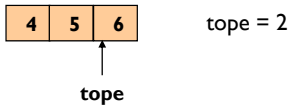


- p.apilar(new Integer(5));

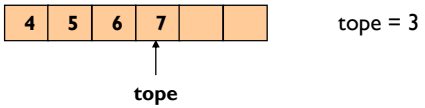


Traza Pila (2/2)

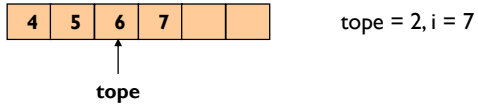
- p.apilar(new Integer(6));



- p.apilar(new Integer(7));



- Integer i = p.desapilar();



- p.toString(); → " 6 5 4 "

borraBase y topeBase en Pila

- Añadimos los nuevos métodos a la especificación de Pila y realizamos su implementación en la clase ArrayPila

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

```
public void topeBase(){
    E tmp;
    tmp = elArray[0];
    elArray[0] = elArray[tope];
    elArray[tope] = tmp;
  }
```

borraBase y topeBase en Pila

```
public class ArrayPila<E> implements Pila<E> {
  …
 public void topeBase(){ …}

 public E borraBase() {
       E base = elArray[0];
       for ( int i = 1 ; i <= tope ; i++){
         elArray[i-1] = elArray[i];
        }
        tope--;
        return base;
     }
}// Fin de la clase ArrayPila
```


Reemplazar Ocurrencias en Pila (1/3)

- Enriquecemos el modelo de pila

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


Reemplazar Ocurrencias en Pila (2/3)

- Implementación del método utilizando únicamente la especificación de Pila

```
package librerias.estructurasDeDatos.lineales;
import librerias.estructurasDeDatos.modelos.*;
public class ArrayPila<E> implements Pila<E>{
…
  public int reemplaza(E x, E y){
    int nreemp = 0;
     if (!esVacia()){
         E aux = desapilar();
         nreemp += reemplaza(x, y);
         if (aux.equals(x)) {apilar(y); nreemp++;}
         else apilar(aux);
    }
    return nreemp;   }}
```

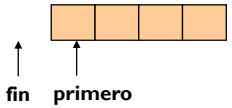

Reemplazar Ocurrencias en Pila (3/3)

- Implementación del método teniendo acceso a la implementación

```
package librerias.estructurasDeDatos.lineales;
import librerias.estructurasDeDatos.modelos.*;
public class ArrayPila<E> implements Pila<E> {
…
  public int reemplaza(E x, E y){
      int nreemp = 0;
      for (int i = tope; i >= 0 ; i--) {
        if (elArray[i].equals(x)) {
            elArray[i] = y;
               nreemp++;
          }
       }
       return nreemp;
   }}
```

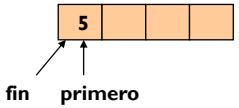
Traza Cola (1/4)

Cola<Integer> q = new ArrayCola<Integer>()



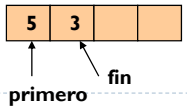
primero = 0, fin = -1, tallaActual = 0

q.encolar(new Integer(5));



primero = 0, fin = 0, tallaActual = 1

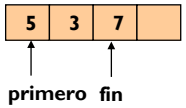
q.encolar(new Integer(3));



primero = 0, fin = 1, tallaActual = 2

Traza Cola (2/4)

q.encolar(new Integer(7));

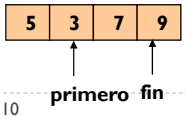


primero = 0, fin = 2, tallaActual = 3

q.encolar(new Integer(9));

primero = 0, fin = 3, tallaActual = 4

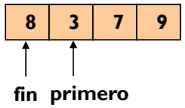
Integer y = q.desencolar();



primero = 1, fin = 3, tallaActual =3, y = 5

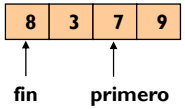
Traza Cola (3/4)

q.encolar(new Integer(8));



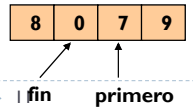
primero = 1, fin = 0, tallaActual = 4

Integer y = q.desencolar();



primero = 2, fin = 0, tallaActual = 3, y = 3

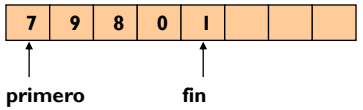
q.encolar(new Integer(0))



primero = 2, fin = 1, tallaActual = 4

Traza Cola (4/4)

q.encolar(new Integer(1))



primero = 0, fin = 4, tallaActual = 5

Tas ejecutar q.toString(): 7 9 8 0 1

Localiza en LEGListaConPI (1/2)

- Implementación utilizando únicamente los métodos de la interfaz ListaConPI.

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


Localiza en LEGListaConPI (2/2)

- Implementación utilizando los atributos:

```
public class LEGListaConPI<E> implements ListaConPI<E>{
…
    public void localiza(E x){
      boolean enc = false;
      ant = pri;
      while (ant.siguiente != null && !enc){
        if (ant.siguiente.dato.equals(x)) enc = true;
        else ant = ant.siguiente;
      }}}
```


Método anterior en ListaConPI (1/2)

1. NO es posible implementar el método anterior con coste constante con esas clases.
   - Sin embargo es posible implementarlo con coste lineal con la talla del problema (número de elementos de la lista).

```
public void anterior(){
     NodoLEG<E> aux = pri;
     while (aux.siguiente != ant) aux = aux.siguiente;
     ant = aux;
   }
```

2. Se debe utilizar como representación interna una Lista Doblemente Enlazada (objetos NodoLDEG) para poder tener una referencia de un nodo a su nodo anterior.

Método anterior en ListaConPI (2/2)

2. (Continuacion). Se deben modificar las operaciones de inserción y borrado para actualizar la referencia al nodo anterior al involucrado.
   La implementación del método anterior:

```
public void anterior(){
     ant = ant.anterior;
}
```

Borrar en LEGListaConPI

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


Contar Apariciones de Cola (1/2)

1. Implementar la interfaz *ColaExtendida*

```
package modelos;
public interface ColaExtendida<E> extends Cola<E> {
    int contarApariciones(E x);
}
```


Contar Apariciones de Cola (2/2)

1. Implementar la clase *ArrayColaExtendida* suponiendo que se conoce y se tiene accesible la implementación interna de ArrayCola

```
package librerias.estructurasDeDatos.lineales;
import librerias.estructurasDeDatos.modelos.*;
public class ArrayColaExtendida<E> extends ArrayCola<E> implements
    ColaExtendida<E> {
    public int contarApariciones(E x) {
        int indice= primero;
        int contador=1;
        int apariciones=0;
        while (contador<=tallaActual) {
                if (elArray[indice].equals(x)) apariciones++;
                indice= incrementa(indice);  contador++;
        }
        return apariciones;}}
```


Ampliando funcionalidad de ListaConPI (1/3)

- Implementar la interfaz ListaConPIPlus

```
package librerias.estructurasDeDatos.modelos;
public interface ListaConPIPlus<E> extends ListaConPI<E>
{
    void vaciar();
    int talla();
}
```

Ampliando funcionalidad de ListaConPI (2/3)

- Implementar la clase LEGListaConPIPlus suponiendo que se conoce y se tiene accesible la implementación interna de LEGListaConPI

```
package librerias.estructurasDeDatos.lineales;
import librerias.estructurasDeDatos.modelos.*;
public class LEGListaConPIPlus<E> extends LEGListaConPI<E>
      implements ListaConPIPlus<E>{

  public void vaciar(){
     pri.siguiente = null;  ant = pri; ult = pri;
   }
```

- El mecanismo de recogida de basura provocará la eliminación en cascada de todos los nodos.

Ampliando funcionalidad de ListaConPI (3/3)

- Implementar la clase *LEGListaConPIPlus* suponiendo que se tiene accesible LEGListaConPI

```
package librerias.estructurasDeDatos.lineales;
import librerias.estructurasDeDatos.modelos.*;
public class LEGListaConPIPlus<E> extends LEGListaConPI <E>
     implements ListaConPIPlus <E>{
 public int talla(){
    NodoLEG<E> aux = pri.siguiente;   int contador = 0;
    while (aux != null) {
      contador++;  aux = aux.siguiente;
    }
    return contador;
 }
```


Implementación de Cola con Lista Enlazada (1/3)

```
public class LEGCola<E> implements Cola<E>{
  NodoLEG<E> primero, fin;

  public LEGCola(){ primero = fin = new NodoLEG<E>(null); }

  public void encolar(E x){
      NodoLEG<E> nuevo = new NodoLEG<E>(x);
      fin.siguiente = nuevo;
      fin = nuevo;
  }
  public E desencolar(){
     E elPrimero = primero.siguiente.dato;
     primero.siguiente = primero.siguiente.siguiente;
     if ( primero.siguiente == fin) fin = primero;
     return elPrimero;   }
```


Implementación de Cola con Lista Enlazada (2/3)

```
  public E primero(){
     return primero.siguiente.dato;
  }

  public boolean esVacia(){ return primero == fin; }

  public String toString(){
      NodoLEG<E> aux = primero.siguiente;
      String cadena = "";
      while (aux != null) {
          cadena += aux.dato + " ";
          aux = aux.siguiente;
        }
      return cadena;
    }
} /* Fin de LEGCola */
```

Implementación de Cola con Lista Enlazada (3/3)

- Coste Temporal Asintótico (Para todos los métodos de Cola)
  - Talla del problema: Número de elementos de la Cola.
  - Instancias Significativas: No hay
  - Cotas de Complejidad: $\Theta(1)$.
- El coste de los métodos con esta implementación es constante, al igual que con la implementación en array.
- Pero el coste espacial es mayor, debido a la memoria necesaria por los nodos y por los enlaces.

Implementación de Pila con LEG (1/2)

```
public class LEGPila<E>  implements Pila<E>{
    NodoLEG<E> elTope;
    public LEGPila () {
        elTope=null;
    }
    public void apilar(E x) {
        elTope=new NodoLEG<E>(x,elTope);
    }
    public E tope() {
        return elTope.dato;
    }
```


Implementación de Pila con LEG (2/2)

```
public E desapilar() {
    E dato = elTope.dato;
    elTope = elTope.siguiente;
    return dato;
}
public boolean esVacia() { return (elTope==null); }
public String toString() {
   String res = "";
   for (NodoLEG<E> aux = elTope; aux!=null; aux=aux.siguiente)
           res += aux.dato +"\n";
    return res;
    }}
```


El Modelo Secuencia (ArraySecuencia) (1/5)

```
import librerias.excepciones.*;
public class ArraySecuencia<E> implements Secuencia<E>{
  protected int tope, talla;
  protected E elArray[];
  protected final static int CAPACIDAD_POR_DEFECTO = 200;

 public ArraySecuencia(){
    elArray = (E[]) new Object[CAPACIDAD_POR_DEFECTO];
    tope = -1;
    talla = 0;
  }
```

El Modelo Secuencia (ArraySecuencia) (2/5)

```
public void insertar(E x){
    if ( tope + 1 == elArray.length) duplicarVector();
    tope++;
    elArray[tope] = x;
    talla++;
 }
 private void duplicarVector() {
     E nuevoVector[] = (E[]) new Object[elArray.length*2];
     for ( int i = 0; i <= tope; i++ ) nuevoVector[i] = elArray[i];
     elArray = nuevoVector;
  }
```


El Modelo Secuencia (ArraySecuencia) (3/5)

```
public int indiceDe(E x){
    int i = 0;
    boolean enc = false;
    while (i < talla && (!enc)){
     if (elArray[i].equals(x)) enc = true;
     else i++;
    }
    if (enc) return i;
    else return -1;
    }
```


El Modelo Secuencia (ArraySecuencia) (4/5)

```
public E borrar(E x) throws ElementoNoEncontrado{
    E elBorrado = null;
    int pos = indiceDe(x);
    if (pos == -1) throw new ElementoNoEncontrado("El elemento " +
   x + " no esta");
    else {
      elBorrado = elArray[pos];
      for ( int i = pos ; i < talla-1; i++) elArray[i] = elArray[i+1];
   }
    talla--;
    return elBorrado;
    }
```


El Modelo Secuencia (ArraySecuencia) (5/5)

```
public E recuperar(int indice){ return elArray[indice]; }

public boolean esVacia(){ return talla == 0;}

public int talla(){ return talla;}

public String toString(){
    String cadena = "";
    for (int i = 0;  i<talla; i++) {
       cadena += elArray[i] + " ";
     }
     return cadena;
   }

} /* Fin de la clase ArraySecuencia */
```

El Modelo Secuencia (LEGSecuencia) (1/5)

```
public class LEGSecuencia<E> implements Secuencia<E>
{
  protected NodoLEG<E> primero, ultimo;
  protected int talla;

public LEGSecuencia(){
    primero = ultimo = new NodoLEG<E>(null);
    talla= 0;
  }

public void insertar(E x){
    NodoLEG<E> nuevo = new NodoLEG<E>(x);
    ultimo.siguiente = nuevo;
    ultimo = nuevo;
    talla++;
  }
```


El Modelo Secuencia (LEGSecuencia) (2/5)

```
public E borrar(E x) throws ElementoNoEncontrado{
   NodoLEG<E> aux = primero;
   boolean enc = false;    E elBorrado = null;
   while (aux.siguiente != null && !enc) {
     if (aux.siguiente.dato.equals(x)) enc = true;
     else aux = aux.siguiente;
    }
    if (!enc) throw new ElementoNoEncontrado("No esta " + x);
    else {
        elBorrado = aux.siguiente.dato;
        if ( aux.siguiente == ultimo ) ultimo = aux;
        aux.siguiente = aux.siguiente.siguiente;
    }
    talla--;
    return elBorrado;
  }
```


El Modelo Secuencia (LEGSecuencia) (3/5)

```
public int indiceDe(E x) {
   NodoLEG<E> aux = primero.siguiente;
   int contador = 0;
   boolean enc = false;

   while (aux != null && !enc){
     if (aux.dato.equals(x)) enc = true;
     else {
        contador++;
        aux = aux.siguiente;
       }
   }
   if (!enc) return -1;
   else return contador;

  }
```


El Modelo Secuencia (LEGSecuencia) (4/5)

```
public E recuperar(int indice){
    NodoLEG<E> aux = primero.siguiente;
    int i = 0;
    while ( i < indice) {aux = aux.siguiente; i++;}
    return aux.dato;
   }

   public boolean esVacia(){
    return primero == ultimo;
   }

  public int talla(){
    return talla;
   }
```

El Modelo Secuencia (LEGSecuencia) (5/5)

```
public String toString(){
    String cadena = "";
    NodoLEG<E> aux = primero.siguiente;
    while (aux != null) {
      cadena += aux.dato + " ";
      aux = aux.siguiente;
     }
   return cadena;
 }
} /* Fin de la clase LEGSecuencia */
```


El Modelo Secuencia (Costes)

- ArraySecuencia:
  - insertar, recuperar, esVacia, talla: Θ(1)
  - Borrar: Θ(N) (Desplazamiento de datos)
  - indiceDe: Ω(1), O(N)
- LEISecuencia
  - insertar, esVacia, talla: Θ(1)
  - borrar: Ω(1), O(N)
  - indiceDe: Ω(1), O(N)
  - recuperar: Θ(N)

Invierte Pila (1/2)

```
package librerias.estructurasDeDatos.modelos;
public interface Pila<E> {
    ...
     public void invertir();
}
```

```
package lineales;
public class  ArrayPila<E> implements Pila<E>{
    ...
      public void invertir(){
        for (int i = 0; i <= tope/2 ; i++){
            swap(elArray, i, tope-i);
         }
       }
...
```


Invierte Pila (2/2)

```
private void swap(E[] v, int pos1, int pos2){
        E tmp;
        tmp = v[pos1];
        v[pos1] = v[pos2];
      v[pos2] = tmp;
   }
}
```

- NOTA: Cuando hay un nº impar de elementos en la Pila el intercambio central no es necesario.
- El coste del algoritmo es lineal con el número de elementos de la pila, sin instancias significativas.

Cambia Signo en Pila

```
public class LEGPilaInteger extends LEGPila<Integer>
   implements Pila<Integer>{

public void cambiarSigno() {
   for (NodoLEG<Integer> aux=this.tope; aux!=null;
   aux=aux.siguiente){
      aux.dato = new Integer(-(aux.dato.intValue()));
   }
 }
```


Método anterior en LEGListaConPI (1/2)

```
public void anterior() throws ElementoNoEncontrado{
   // Sólo se ejecuta cuando hay un Elemento anterior al que ocupa el PI actual
   if ( ant == pri ) throw new ElementoNoEncontrado("Error: no hay anterior al
    del PI actual");
   // ant.siguiente señala el PI actual. Recorrido hasta ant con un Enlace auxiliar
   NodoLEG<E> anteriorPI = pri;
   while ( anteriorPI.siguiente != ant ) anteriorPI = anteriorPI.siguiente;
   /* Resolución del Recorrido: al terminar el bucle anteriorPI.siguiente == ant, por
    lo que basta con la  instrucción ant = anteriorPI para que el PI de la Lista se
    sitúe sobre el anterior al actualmente apuntado. Nótese que si esFin()==true, el
    anterior es el último de la Lista */
   ant = anteriorPI;
}
```


Método anterior en LEGListaConPI (2/2)

```
public void anterior() throws ElementoNoEncontrado{
   if ( ant == pri ) throw new ElementoNoEncontrado("Error: no hay anterior
    al del PI actual");
   ant = ant.anterior;
}
//Coste independiente de la talla
```