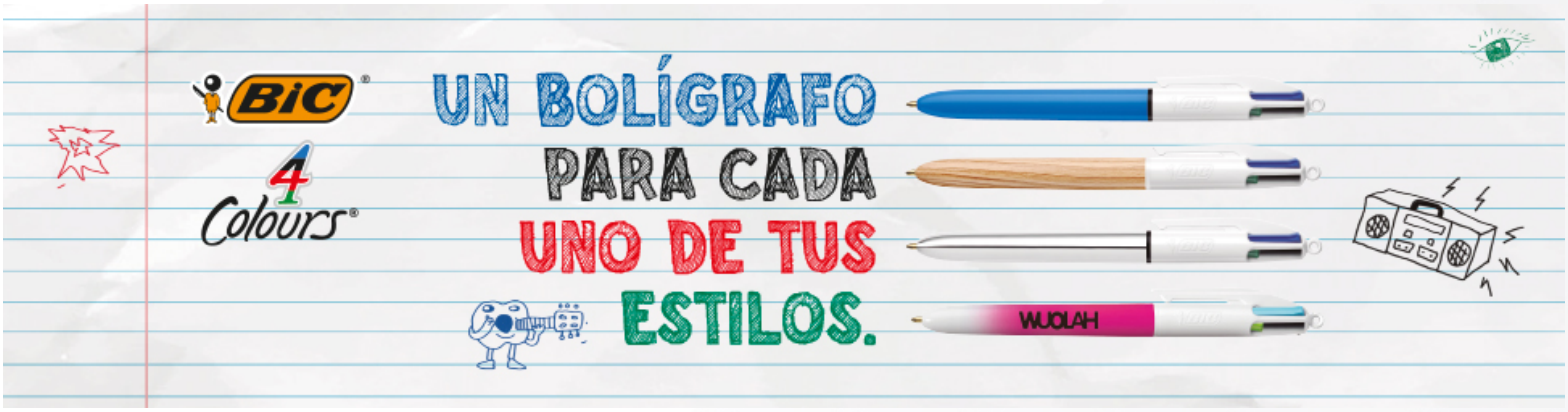
ejerciciosT1soluciones.pdf

onepiece_03

Estructuras de Datos y Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia



1. Aplicación de los criterios de diseño de las clases jerárquicas Java de una EDA

Ejercicio 1.1: Implementa las clases **ArrayPila** y **LEGPila** implementaciones del modelo **Pila** mediante un array y una lista enlazada genérica, respectivamente.

SOLUCIÓN:

```
package librerias.estructurasDeDatos.lineales;

import  librerias.estructurasDeDatos.modelos.*;

public class ArrayPila<E> implements Pila<E> {
      // Atributos
       protected E elArray[];
       protected int tope;
       protected static final int CAPACIDAD_POR_DEFECTO = 50;

      // Constructor vacío
      @SuppressWarnings("unchecked")
      public ArrayPila(){
        elArray = (E []) new Object[CAPACIDAD_POR_DEFECTO];
        tope = -1;
      }

      /** inserta el elemento e en una pila, o lo coloca en su parte superior
**/
      public void apilar(E e) {
        if ( tope + 1 == elArray.length) duplicarArray();
        tope++;
        elArray[tope] = e;
      }
      // duplica el tamaño actual de un array
      @SuppressWarnings("unchecked")
      protected void duplicarArray() {
          E nuevoArray[] = (E []) new Object[elArray.length*2];
          System.arraycopy(elArray, 0, nuevoArray, 0, tope);
          elArray = nuevoArray;
      }

      /** IFF !esVacia(): recupera y elimina de una pila el elemento situado
en su parte superior **/
      public E desapilar(){
        E res = elArray[tope];
        tope--;
        return res;
      }

      /** IFF !esVacia(): obtiene el elemento situado en la parte superior de
una pila **/
      public E tope(){
        return elArray[tope];
      }

      /** comprueba si la pila es vacía **/
      public boolean esVacia(){
         return ( tope == -1 );
      }
```

```
    /** recupera un String con los elementos de la pila en orden LIFO
     *  e.g. para una pila de enteros de 1 a 4,
     *  en órden LIFO, toString imprimirá [4, 3, 2, 1];
     *  si es vacía, []
     */
    public String toString() {
        StringBuilder res = new StringBuilder();
        res.append("[");
        for ( int i = tope; i >= 0; i-- ) res.append(elArray[i]);
        res.append("]");
        return res.toString();

        //String res = "";
        //for ( int i = tope; i >= 0; i-- ) res +=elArray[i].toString()+"\n";
        //return res;

    }
}
```

SOLUCIÓN:

```
package librerias.estructurasDeDatos.lineales;

import  librerias.estructurasDeDatos.modelos.*;

public class LEGPila<E> implements Pila<E> {

       // Atributos
      protected NodoLEG<E> tope;
      protected int talla;

      // Constructor vacío
      public LEGPila(){
          tope = null;
          talla = 0;
      }

       /** inserta el elemento e en una pila, o lo coloca en su parte
superior **/
      public void apilar(E e){
          tope = new NodoLEG<E>(e, tope);
          talla++;
      }

      /** IFF !esVacia(): recupera y elimina de una pila el elemento situado
en su parte superior **/
      public E desapilar(){
           E datoPI = tope.dato;
           tope = tope.siguiente;
           talla--;
           return datoPI;
      }

      /** IFF !esVacia(): obtiene el elemento situado en la parte superior de
una pila **/
       public E tope(){
           return tope.dato;
      }
```

Lorem

PIRATA BEACH FEST

12 - 13- 14 Y 15 DE JULIO

GANDIA · 5º ANIVERSARIO

SKA- P ⋋ NATOS Y WAOR ⋋ SFDK ⋋ DORIAN ⋋ AYAX Y PROK ⋋ DESAKATO
SIDECARS ⋋ MALA RODRÍGUEZ ⋋ TALCO ⋋ MORAD ⋋ FERNANDOCOSTA
STAY HOMAS ⋋ TOTEKING ⋋ POLE ⋋ LOS CHIKOS DEL MAÍZ ⋋ LA FÚMIGA
LENDAKARIS MUERTOS ⋋ CELTAS CORTOS ⋋ SIDONIE ⋋ KAZE ⋋ NARCO
JUANCHO MARQUÉS ⋋ ELS CATARRES ⋋ BOIKOT ⋋ LÁGRIMAS DE SANGRE
ANDY Y LUCAS ⋋ CIUDAD JARA ⋋ HOKE & LOUIS AMOEBA ⋋ NATALIA LACUNZA
EL CANIJO DE JEREZ ⋋ VARRY BRAVA ⋋ MOJINOS ESCOZÍOS ⋋ PIGNOISE
SHARIF ⋋ LOS DE MARRAS ⋋ CUPIDO ⋋ AUXILI ⋋ REINCIDENTES ⋋ BUHOS
FYAHBWOY ⋋ RAPSUSKLEI ⋋ SUU ⋋ SONS OF AGUIRRE & SCILA ⋋ DON PATRICIO
SÔBER ⋋ MR. KILOMBO ⋋ NATIVA ⋋ EL ÚLTIMO KE ZIERRE ⋋ ENVIDIA KOTXINA
ZETAK ⋋ SMOKING SOULS ⋋ KING ÁFRICA ⋋ ITACA BAND ⋋ LEY DJ ⋋ CHATA FLORES
THE TYETS ⋋ PAULA KOOPS ⋋ FUNKIWIS ⋋ GIGATRON ⋋ TRIBADE ⋋ PUPIL-LES
GRISON ⋋ EMLAN ⋋ AYA ⋋ KULTO KULTIBO ⋋ DJ PLAN B ⋋ MALUKS
HOLISTIKS ⋋ CHILL ADDICTS ⋋ EL SONIDO DE LA CIGARRA

A LA VENTA EN PIRATAFESTIVAL.COM

4 DÍAS DE FESTIVAL · ACAMPADA · RAVE · BUS A LA PLAYA · 5 ESCENARIOS

```
      /** comprueba si la pila es vacía **/
      public boolean esVacia(){
           return ( tope == null );
      }

      /** recupera un String con los elementos de la pila en orden LIFO **/
      public String toString(){
           String res = "";
           NodoLEG<E> aux;
           for ( aux = tope ; aux != null; aux = aux.siguiente)
                res += aux.dato.toString()+"|";
           return res;
      }
}
```

Ejercicio 1.2: Implementa la clase **LEGCola**, implementación del modelo de colas mediante una lista enlazada genérica.

SOLUCIÓN:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.*;

public class LEGCola<E> implements Cola<E> {
   // Atributos
    protected NodoLEG<E> principioC, finalC;
    protected int talla;

    /** Constructor vacío **/
    public LEGCola(){
        principioC = finalC = null;
        talla = 0;
    }

    /** inserta el elemento e en una cola, o lo coloca en su parte superior
**/
    public void encolar(E e){
        NodoLEG<E> nuevo = new NodoLEG<E>(e);
        // IF this.esVacia():
        // actualiza principioC
        if ( finalC == null ) principioC = nuevo;
        else finalC.siguiente = nuevo;
        finalC = nuevo;
        talla++;
    }

    /** IFF !esVacia(): recupera y elimina de una cola el elemento que se
encuentra al principio**/
    public E desencolar(){
        E datoPI = principioC.dato;
        principioC = principioC.siguiente;
        // actualiza finalC después de recuperar el último elemento de la
cola
        if ( principioC == null ) finalC = null;
        talla--;
        return datoPI;
    }
```

```
    /** IFF !esVacia(): recupera el elemento en la parte superior de la cola,
     * el primero en orden de inserción **/
    public E primero(){
        return principioC.dato;
    }

     /** comprueba si la cola es vacía **/
    public boolean esVacia(){
        return ( principioC == null );
    }

    /** recupera un String con los elementos de la cola en orden de inserción
    public String toString(){
        String res = "";
        for ( NodoLEG<E> aux = principioC ; aux != null; aux = aux.siguiente
)
                res += aux.dato.toString()+"|";
        return res;
    }
}
```

2. Uso de la jerarquía Java de una EDA

Ejercicio 2.1: Ampliar la funcionalidad de la EDA **Pila** vía herencia (**LEGPilaExt**) para añadir un nuevo método que devuelva el elemento en la base de la pila. Implementa este método

a) Accediendo a los atributos de **LEGPila**.
b) Utilizando únicamente los métodos del modelo.

SOLUCIÓN a:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.PilaExt;

public class LEGPilaExt<E> extends LEGPila<E> implements PilaExt<E> {

    /** devuelve el elemento en la base de la pila accediendo a los
    * atributos de LEGPila **/
    public E base() {
        NodoLEG<E> aux = tope;
        while (aux.siguiente != null) aux = aux.siguiente;
        return aux.dato;
    }
}
```

SOLUCIÓN b:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.PilaExt;

public class LEGPilaExt<E> extends LEGPila<E> implements PilaExt<E> {

    /** devuelve el elemento en la base de la pila utilizando únicamente
    * los métodos del modelo **/
```

```
    public E base() {
         E res, aux = desapilar();
         if (esVacia()) res = aux;
         else res = base();
         apilar(aux);
         return res;
    }
}
```

Ejercicio 2.2: Ampliar la funcionalidad de la EDA **Cola** mediante herencia (**ColaExt** y **ArrayColaExt**) para añadir un nuevo método que cambie el orden de los elementos en la cola. Implementa este método

a) Accediendo a los atributos de **ArrayCola**.
b) Utilizando únicamente los métodos del modelo.

SOLUCIÓN:

```
package librerias.estructurasDeDatos.modelos;

public interface ColaExt<E> extends Cola<E> {
    void invertir();
}
```

SOLUCIÓN a:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.ColaExt;

public class ArrayColaExt<E> extends ArrayCola<E> implements ColaExt<E> {

    /** cambia el orden de los elementos en la cola accediendo a los
    * atributos de ArrayCola     */
    public void invertir() {
        int i = principioC, j = finalC;

        for (int cont = 0; cont < talla/2; cont++) {
            E aux = elArray[i];
            elArray[i] = elArray[j];
            elArray[j] = aux;
            if (++i == elArray.length) i = 0;
            if (--j == -1) j = elArray.length - 1;
        }
    }
}
```

SOLUCIÓN b:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.ColaExt;

public class ArrayColaExt<E> extends ArrayCola<E> implements ColaExt<E> {
```

```
    /** cambia el orden de los elementos en la cola utilizando únicamente
    * los métodos del modelo     */
    public void invertir() {
        if (!esVacia()) {
            E tmp = desencolar();
            invertir();
            encolar(tmp);
        }
    }
}
```

Ejercicio 2.3: Corregir errores de compilación en la clase **TestEdaCola** (en ejemplos/tema1), sin modificar la interfaz **Cola<E>** ni la clase **ArrayCola<E>** que implementa la interfaz.

SOLUCIÓN:

```
package ejemplos.tema1;

// modificación 1: incluir importaciones
import librerias.estructurasDeDatos.modelos.*;
import librerias.estructurasDeDatos.lineales.*;

public class TestEDACola {
    public static void main(String[] args) {
        // modificación 2: instanciar la cola a números enteros
        Cola<Integer> q = new ArrayCola<Integer>();

        // modificaciones relacionadas con el tamaño
        // declarar una variable local, tallaQ, con la misma funcionalidad
        // ya que, en este ejercicio
        // no se permiten modificaciones en Cola ni en ArrayCola.

        int tallaQ = 0; // modificación 3
        System.out.println("Creada una Cola con " + /*q.talla()*/tallaQ
            + " Integer, q = " + q.toString());
        q.encolar(new Integer(10));
        tallaQ++; // modificación 4
        q.encolar(new Integer(20));
        tallaQ++; // modificación 5
        q.encolar(new Integer(30));
        tallaQ++; // modificación 6
        System.out.println("La Cola de Integer actual es q = " +
q.toString());
        System.out.println("Usando otros metodos para mostrar sus Datos el
resultado es ...");
        String datosQ = "";
        while (!q.esVacia()) {
            Integer primero = q.primero();
            if (primero.equals(q.desencolar())) datosQ += primero + " ";
            else datosQ += "ERROR ";
            tallaQ--; // modificación 7
        }
        System.out.println(" el mismo, " + datosQ
            + ", PERO q se vacia, q = " + q.toString());
    }
}
```

3. Jerarquía Colección

Ejercicio 3.1: Utilizando la clase de la jerarquía java Collection **ArrayDeque** vía herencia, diseña la clase **ArrayDequeCola** que implemente la interfaz **Cola**.

SOLUCIÓN:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.*;
import java.util.ArrayDeque;
import java.util.Iterator;

public class ArrayDeQueCola<E> extends ArrayDeque<E> implements Cola<E> {
    protected ArrayDeque elArray;

    /** constructor **/
    @SuppressWarnings("unchecked")
    public ArrayDeQueCola() {
        elArray = new ArrayDeque();
    }

    /** inserta e al final de la cola **/
    // SII la inserción supera el tamaño de la matriz, ArrayDeque
    // duplica automáticamente su tamaño hasta 64 e incrementa size()/2
    // a partir de tamaños mayores.
    public void encolar(E e) {
        elArray.add(e);
    }

    /** SII !esVacia ():
     *  retira y borra el elemento al principio de la cola **/
    public E desencolar() {
        return (E) elArray.poll();
    }

    /** SII !isEmpty():
     ** retira pero no borra el elemento al principio de la cola, en orden
     * de inserción **/
    public E primero() { return (E) elArray.peekFirst(); }

    /** comprueba si la cola está vacía **/
    public boolean esVacia() { return elArray.size() == 0; }

   /** obtiene un String con los Elementos de una Cola en orden FIFO,
    * u orden de inserción, y en el formato utilizado en el estándar Java.
    * Así, por ejemplo, si tienes una Cola con Enteros 1 a 4,
    * en orden FIFO, toString devuelve [1, 2, 3, 4];
    * si la cola está vacía, devuelve [].*/

    public String toString() {
        // NOTA: Se utiliza la clase StringBuilder en lugar de String,
        // por razones de eficiencia
        StringBuilder res = new StringBuilder();
        res.append("[");
```

```
        //Iterator i = elArray.iterator();
        //while (i.hasNext())
        //      res.append(i.toString() + ", ");

        for(Integer j = 0; j < elArray.size();j++ )
            res.append(elArray.toArray()[j].toString());

        res.append("]");
        return res.toString();
    }
}
```

Ejercicio 3.2: Utilizando vía Herencia la clase **ArrayDequeCola** anterior, escribe la clase **ArrayDequeColaPlus** que implemente la interfaz **ColaPlus**.

SOLUCIÓN:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.*;

public class ArrayDeQueColaPlus<E> extends ArrayDeQueCola<E> implements
ColaPlus<E> {

    public int talla() {
        return elArray.size();
    }
}
```

Ejercicio 3.3: Utilizando la jerarquía **Deque** escribe un nuevo diseño de la clase **TestEdaCola**, llamado **TestEDAColaVDeque**, equivalente a **TestEDACola**.

SOLUCIÓN:

```
package ejemplos.tema1;

import librerias.estructurasDeDatos.modelos.*;
import librerias.estructurasDeDatos.lineales.*;

public class TestEDAColaVDeque {
    public static void main(String[] args) {
        ColaPlus<Integer> q = new ArrayDeQueColaPlus<Integer>();

        System.out.println("Creada una Cola con " + q.talla()
            + " Integer, q = " + q.toString());
        q.encolar(new Integer(10));
        q.encolar(new Integer(20));
        q.encolar(new Integer(30));
        System.out.println("La Cola de Integer actual es q = " +
q.toString());
        System.out.println("Usando otros metodos para mostrar sus Datos el
resultado es ...");
        String datosQ = "";
        while (!q.esVacia()) {
            Integer primero = q.primero();
            if (primero.equals(q.desencolar())) datosQ += primero + " ";
```

```
            else datosQ += "ERROR ";
        }
        System.out.println(" el mismo, " + datosQ
            + ", PERO q se vacia, q = " + q.toString());
    }
}
```

4. Lista con Punto de Interés

Ejercicio 4.1: Crea la clase **LEGListaConPIPlus**, que será una implementación enlazada del (sub)Modelo **ListaConPIPlus**.

IMPORTANTE: para desarrollar esta clase debes descargar la plantilla de la interfaz ListaConPiPlus.java de PoliformaT T1/Plantillas. No utilices la proporcionada en eda.zip.

Crea la clase e implementa los siguientes métodos:

- boolean **contiene**(E)
- boolean **eliminarPrimero**(E)
- boolean **eliminarUltimo**(E)
- boolean **eliminarTodos**(E e)
- void **concatenar**(ListaConPI<E>)
- void **vaciar**()
- void **buscar**()
- String **toString**()

SOLUCIÓN:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.*;

public class LEGListaConPiPlus<E> extends LEGListaConPI<E> implements
ListaConPIPlus<E> {

    /** comprueba si Elemento e está en Lista Con PI **/
    public boolean contiene(E e){
        inicio();
        while (!esFin() && !recuperar().equals(e)) siguiente();
        if (esFin()) return false;
        return true;
    }

    /** elimina la primera aparición del elemento e en una lista con PI
    * y devuelve true, o devuelve false si e no está en la Lista.**/

    public boolean eliminarPrimero(E e){
        inicio();
        while (!esFin()) {
            if (recuperar().equals(e)) {
                eliminar();
                return true;
            }
            siguiente();
```

```
        }
        return false;
    }

    /** elimina la última aparición del Elemento e en una Lista Con PI
     * y devuelve true, o devuelve false si e no está en la Lista.**/

    public boolean eliminarUltimo(E e){
        NodoLEG<E> ultimo = null;
        inicio();
        while (!esFin()) {
            if (recuperar().equals(e)) ultimo = ant;
            siguiente();
        }
        if (ultimo == null) return false;
        else{
            //if (ultimo.siguiente == ult) ant = ult = ultimo;
            //ultimo.siguiente = ultimo.siguiente.siguiente;
            //talla--;
            //return true;

            ant = ultimo;
            eliminar();
            return true;
        }
    }

    /** elimina todas las apariciones del elemento e en una lista con PI
     * y devuelve true, o devuelve false si e no está en la Lista
     */
    public boolean eliminarTodos(E e){
        boolean b = false;
        inicio();
        while (!esFin()) {
            if (recuperar().equals(e)) {
                eliminar();
                b = true;
            }
            else siguiente();
        }
        return b;
    }

    /** concatena una Lista Con PI con otra **/
    public void concatenar(ListaConPI<E> otra){
        this.fin();
        otra.inicio();
        while (!otra.esFin()) {
            E dato = otra.recuperar();
            this.insertar(dato);
            otra.siguiente();
        }
    }

    /** elimina todos los elementos de una lista con PI **/
    public void vaciar(){
        inicio();
        while (!esVacia()) eliminar();
    }
```

```
    /** invierte el orden de los elementos de la lista **/
    public void invertir(){
        if (!esVacia()) {
            inicio();
            E dato = recuperar();
            eliminar();
            invertir();
            insertar(dato);
        }
    }

    /** coloca el PI en x. Si no se encuentra el punto de referencia,
    * el PI se colocará al final de la lista **/
    public void buscar(E x){
        inicio();
        while (!esFin() && !recuperar().equals(x)) siguiente();
    }

    /** devuelve una cadena con la descripción de los elementos de
    * la lista **/
    public String toString(){
        String res = "";
        for (inicio(); !esFin(); siguiente())
            res += recuperar().toString() + "\n";
        return res;
    }
}
```

Ejercicio 4.2: Crea un método en **LEGListaConPIPlus** que desplace todos los elementos de la lista una posición hacia la izquierda, de forma que el primer elemento pase a ser el último.

SOLUCIÓN:

```
    public void moverAIzquierda() {
        if (talla <= 1) { return; }
        ult.siguiente = pri.siguiente;
        pri.siguiente = pri.siguiente.siguiente;
        ult = ult.siguiente;
        ult.siguiente = null;
    }
```

Ejercicio 4.3: Amplía la funcionalidad de la EDA Lista con PI mediante herencia con los siguientes métodos:

- void **buscar**(E x): coloca el PI en x. Si no se encuentra el dato, el PI se colocará al final de la lista.
- void **vaciar**(): vacía la lista.
- void **invertir**(): invierte el orden de los elementos de la lista.
- void **eliminar**(E x): elimina todos los elementos iguales a x de la lista.

Para ello, sólo utiliza los métodos existentes en el modelo **Lista con PI**.

SOLUCIÓN:

```
public interface ListaConPIExt<E> extends ListaConPI<E> {
	void buscar(E x);
	void vaciar();
	void invertir();
	void eliminar(E x);
}

public class LEGListaConPIExt<E> extends LEGListaConPI<E> implements
ListaConPIExt<E> {

	public void buscar(E x) {
	    inicio();
	    while (!esFin() && !recuperar().equals(x)) siguiente();
	}

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

	public void invertir(){
	    if (!esVacia()) {
	       inicio();
	       E dato = recuperar();
	       eliminar();
	       invertir();
	       insertar(dato);
	    }
	}

	public void eliminar(E x) {
	    inicio();
	    while (!esFin())
	    if (recuperar().equals(x)) eliminar();
	    else siguiente();
	}
}
```

Ejercicio 4.4 [EXAMEN 2021]: Diseñar un método estático tal que, dadas dos **ListasConPI** genéricas, ambas sin elementos repetidos y ordenados ascendentemente, elimine de dichas listas todos los elementos que tengan en común y los devuelva almacenados en una Pila.

Por ejemplo, si la primera lista es [1, 3, 5, 7, 9, 10] y la segunda es [1, 2, 4, 9, 11], tras la llamada al método las listas contendrán [3, 5, 7, 10] y [2, 4, 11], respectivamente, y la pila resultado contendrá los elementos eliminados [1, 9].

SOLUCIÓN:

```
public static < E extends Comparable<E>> Pila<E> metodo1(
	ListaConPI<E> l1, ListaConPI<E> l2) {
	Pila<E> p = new ArrayPila<E>();
	l1.inicio();
	l2.inicio();
```

```
	while (!l1.esFin() && !l2.esFin()) {
		E e1 = l1.recuperar();
		E e2 = l2.recuperar();
		int cmp = e1.compareTo(e2);
		if (cmp == 0) {
			p.apilar(e1);
			l1.eliminar();
			l2.eliminar();
		} else if (cmp < 0) {
			l1.siguiente();
		} else {
			l2.siguiente();
		}
	}
	return p;
}
```

Ejercicio 4.5 [EXAMEN 2021]: Diseñar un método estático tal que, dadas dos **ListaConPI** genéricas, compruebe si ambas listas contienen los mismos elementos pero en orden inverso.

Por ejemplo, el método devolverá true si una lista es [1, 3, 5, 7, 9] y la otra es [9, 7, 5, 3, 1]. Este método no debe modificar el contenido de las listas.

SOLUCIÓN:

```
public static <E> boolean inversas(ListaConPI <E> l1, ListaConPI <E> l2) {
	if (l1. talla() != l2. talla()) return false;
	l1.inicio();
	l2.inicio();
	return inversasRec(l1, l2);
}

private static<E> boolean inversasRec(ListaConPI <E> l1,  ListaConPI <E> l2)
{
	if (l1.esFin()) return true;
	E e1 = l1.recuperar();
	l1.siguiente();
	if (!inversasRec(l1, l2)) return false;
	E e2 = l2.recuperar();
	l2.siguiente();
	return e1.equals(e2);
}
```

5. Comparación Genérica

Ejercicio 5.1: Ampliar la funcionalidad de la EDA **Pila** mediante herencia (**PilaExt**, **LEGPilaExt**) para añadir un nuevo método que devuelva el elemento más pequeño de la pila. Implementa este método:

a) Accediendo a los atributos de **LEGPila**.
b) Utilizando únicamente los métodos del modelo.

SOLUCIÓN:

```
public interface PilaExt<E extends Comparable<E>> extends Pila<E>
{
    // IFF !esVacia()
    E minimo();
}
```

SOLUCIÓN a:

```
public class LEGPilaExt<E extends Comparable<E>> extends LEGPila<E>
implements PilaExt<E> {
/** devuelve el elemento más pequeño de la pila
* accediendo a los atributos de LEGPila **/
    public E minimo() {
        NodoLEG<E> aux = tope;
        E min = null;
        while (aux != null) {
            if (min == null || aux.dato.compareTo(min) < 0) min = aux.dato;
            aux = aux.siguiente;
        }
        return min;
    }
}
```

SOLUCIÓN b:

```
public class LEGPilaExt<E extends Comparable<E>> extends LEGPila<E>
implements PilaExt<E> {
    /** devuelve el elemento más pequeño de la pila
    * utilizando sólo los métodos del modelo **/
    public E minimo() {
        if (esVacia()) return null;
        E dato = desapilar();
        E minResto = minimo();
        apilar(dato);
        if (minResto == null || dato.compareTo(minResto) < 0) return dato;
        return minResto;
    }
}
```

Ejercicio 5.2: En **LEGListaConPi** crea un método estático tal que, dadas dos **ListaConPI** genéricas, ambas sin elementos repetidos y ordenadas de forma ascendente, elimine de dichas listas todos los elementos que tengan en común y los devuelva almacenados en una Cola.

SOLUCIÓN:

```
public static <E extends Comparable<E>> Cola<E>
encolarRepetidos(ListaConPI<E> l1, ListaConPI<E> l2) {
         Cola<E> c = new ArrayCola<E>();
         l1.inicio();
         l2.inicio();
```

```
        while (!l1.esFin() && !l2.esFin()) {
            E e1 = l1.recuperar();
            E e2 = l2.recuperar();
            int cmp = e1.compareTo(e2);
            if (cmp == 0) {
                c.encolar(e1);
                l1.eliminar();
                l2.eliminar();
               }
            else if (cmp < 0) { l1.siguiente(); }
            else { l2.siguiente(); }
           }
           return c;
       }
   }
```