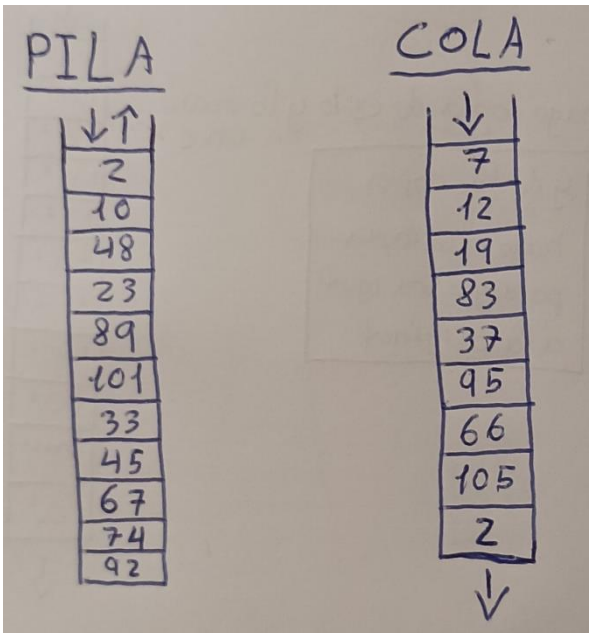
MEMORIA DE LA PRÁCTICA 2: TAD Pila y TAD Cola 2023



29 ABRIL

UNIVERSIDAD DE VALLADOLID
Creado por: Raúl Colindres de Lucas y Alfredo Sobrados González
Nº de grupo de las prácticas: 1
Turno de laboratorio: Miércoles de 18:00h a 20:00h
Fecha de entrega: Martes 2 de mayo de 2023, 22h

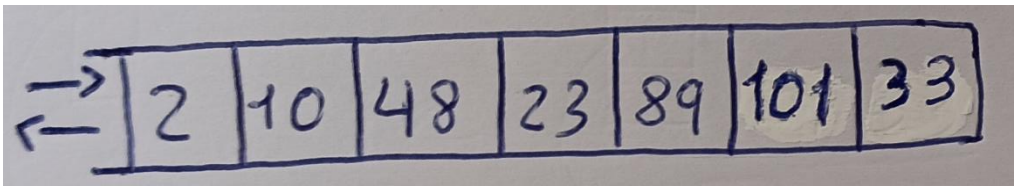
CAMPUS PÚBLICO
MARÍA ZAMBRANO
SEGOVIA

Especificación lógica completa

Especificación lógica completa de los TAD diseñados

Especificación Lógica del TAD Pila

1.-Definición: el TAD a diseñar es una pila, las pilas son contenedores de información los cuales funcionan de esta forma: vas apilando información y cuando quieres desapilar tienes que desapilar siempre justamente el último dato que haya ingresado en la estructura de datos el que esta en la cima o tope de la pila.



2.-Elementos: los elementos que almacena este TAD son enteros y carácteres principalmente.

3.-Tipo de organización: es una estructura de datos lineal.

4.-Dominio de los elementos del TAD: el dominio está compuesto por todos los valores enteros y caracteres que se pueden representar en Java que es lenguaje utilizado para implementar el TAD.

5.-Operaciones Básicas:
Nombre: apilar
Descripción: apila elementos en la pila de manera que el último elemento es la cima
Datos de entrada: el elemento a apilar
Datos de Salida: no devuelve nada.
Precondiciones: que la pila este creada antes de realizar esta operación.
Postcondiciones:

Nombre: desapilar
Descripción: desapila elementos en la pila de manera que el último elemento apilado o la cima de la pila es lo que se desapila.
Datos de entrada: ninguno
Datos de Salida: devuelve el elemento desapilado.

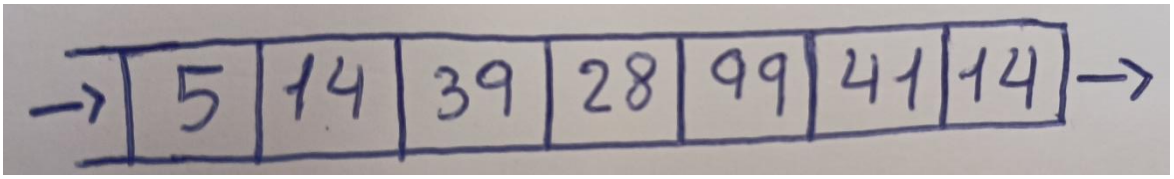
Precondiciones: que la pila no este vacía.
Postcondiciones:

Nombre: tope
Descripción: devuelve el tope o cima de la pila en cuestión y si está vacía devuelve una excepción.
Datos de entrada: ninguno
Datos de Salida: devuelve el tope o cima de la pila.
Precondiciones: que la pila no este vacía.
Postcondiciones: ninguna.

Nombre: esVacia
Descripción: comprueba si la pila está vacía o no
Datos de entrada: ninguno
Datos de Salida: devuelve true si la pila está vacía y devuelve false si tiene contenido.
Precondiciones: ninguna.
Postcondiciones: ninguna.

Especificación Lógica del TAD Cola

1.-Definición: el TAD a diseñar es una pila, las pilas son contenedores de información los cuales funcionan de esta forma: vas apilando información y cuando quieres desapilar tienes que desapilar siempre justamente el último dato que haya ingresado en la estructura de datos el que esta en la cima o tope de la pila.



2.-Elementos: los elementos que almacena este TAD son enteros y carácteres principalmente.

3.-Tipo de organización: es una estructura de datos lineal.

4.-Dominio de los elementos del TAD: el dominio está compuesto por todos los valores enteros y caracteres que se pueden representar en Java que es lenguaje utilizado para implementar el TAD.

5.-Operaciones Básicas:
Nombre: encolar
Descripción: encola elementos en la cola de manera que el primero encolado está en el frente de la cola y el último encolado está en el final de la cola.
Datos de entrada: el elemento a encolar.
Datos de Salida: no devuelve nada.
Precondiciones: que la cola este creada.
Postcondiciones: ninguna.

Nombre: desencolar
Descripción: desencola elementos en la pila de manera que el último elemento es la cima.
Datos de entrada: ninguno.
Datos de Salida: devuelve el elemento desencolado.
Precondiciones: que la cola no este vacía.
Postcondiciones: ninguna.

Nombre: primero
Descripción: esta operación te devuelve el primer elemento encolado en la cola
Datos de entrada: ninguno.
Datos de Salida: devuelve el elemento que esta primero en la cola.
Precondiciones: que la cola no este vacía.
Postcondiciones: ninguna.

Nombre: esVacia
Descripción: comprueba si la cola está vacía o no
Datos de entrada: ninguno
Datos de Salida: devuelve true si la cola está vacía y devuelve false si tiene contenido.
Precondiciones: ninguna.
Postcondiciones: ninguna.

Breve descripción de los procesos

Breve descripción de los procesos llevados a cabo en los distintos métodos que implementan cada una de las opciones de los menús/submenús

El menú principal tiene tres opciones que son el submenú de pilas y el submenú de colas y la opción de comprobar si una frase es palíndroma. Empezaremos describiendo los procesos llevados a cabo en los métodos del submenú de pilas, después haremos lo mismo con el submenú de colas y por último describiremos como se lleva a cabo el proceso de comprobar si una frase es palíndroma o no.

Submenú de pilas:

1. Método crearPilaEnteros(): este método crea la pila de enteros vacía y si tiene contenido entonces la vacía, con el método privado vaciarPila().
2. Método apilarElemPos(): este método apila elementos en la pila creada o vaciada en el anterior método, usa una pila auxiliar en el proceso.
3. Método duplicarPila(): este método duplica la pila original de manera recursiva usando una pila auxiliar copia para realizar este proceso.
4. Método invertirPilaPC(): este método invierte la pila original de manera iterativa usando una pila y una cola auxiliares en el proceso.
5. Método comprobarBasePila(): este método comprueba de manera recursiva si la base de la pila es igual al número de elementos de la pila usando tres pilas auxiliares.

Submenú de colas:

1. Método crearColaEnteros(): este método crea la cola de enteros vacía y si tiene contenido entonces la vacía, con el método privado vaciarCola().
2. Método encolarElemPos(): este método encola elementos en la cola creada o vaciada en el anterior método, usa una cola auxiliar en el proceso.
3. Método copiarCola(): este método copia la cola original de manera recursiva usando una cola auxiliar para realizar este proceso.
4. Método repetirNesimo(): este método solicita un número al usuario e invoca el método recursivo repetirNesimo() encuentra el número “n” en la cola y lo replica las veces que indique el propio elemento.

Clase/Método comprobarFrasePalindroma(): este método/clase lo que hace es comprobar si una frase es palíndroma o no, es decir si se puede leer igual de un lado que de otro, de manera recursiva, con la ayuda de una pila y una cola de caracteres auxiliares para realizar la comprobación

Código fuente

Listado de todo el código fuente desarrollado

Clase Menus:

```
package Menus;

/**
 *
 * @author Usuario
 */
public abstract class Menus {


    public void ejecutar(){
        String respuesta="";

        do{
            respuesta = ejecutarOpciones();
        }while(respuesta.equals("s"));
    }

    public abstract String ejecutarOpciones();
}
```

Clase MenuPrincipal:

```
package Menus;

import java.util.ArrayList;
import EntradaSalida.MyInput;

/**
 *
 * @author Usuario
 */
public class MenuPrincipal extends Menus{
    private ArrayList<Menus> menus = new ArrayList<Menus>();

    public MenuPrincipal(){
        super();
        menus.add(new SubMenuPilaEnteros());
```

```
        menus.add(new SubMenuColaEnteros());
        menus.add(new comprobarFrasePalindroma());
    }

    @Override
    public void ejecutar(){
        String respuesta="";

        do{
            respuesta = ejecutarOpciones();
        }while(respuesta.equals("s"));
    }

    @Override
    public String ejecutarOpciones(){
        System.out.println("");
            System.out.println("Menú Principal");
            System.out.println("seleccione una opción:");
            System.out.println("0. Salir");
            System.out.println("1. Submenú Pila de Enteros");
            System.out.println("2. Submenú Cola de Enteros");
            System.out.println("3. Comprobar frase palíndroma");

        String s=MyInput.readString();

        int i=0;
        try{
               i= Integer.parseInt(s);
        }catch(NumberFormatException ex){
                  System.out.println("La entrada no tiene formato de número.
Inténtelo de nuevo");
                  return "s";
               }
        if((i>0)&&(i<=menus.size())){
                menus.get(i-1).ejecutar();
                return "s";}
        else if ((i<0)||(i>menus.size())){
                System.out.println("opción no válida. Inténtelo de nuevo");
                return "s";}
        else
                System.out.println("Gracias por utilizar nuestros TADs Pila
y Cola");
                return "n";
    }
}
```

Clase SubMenuPilaEnteros:

```
package Menus;

import EntradaSalida.MyInput;
import java.util.Scanner;
import librerias.estructurasDeDatos.lineales.*;

/**
 *
 * @author Usuario
 */
public class SubMenuPilaEnteros extends Menus {
    private LEPila<Integer> pila;

    public SubMenuPilaEnteros(){
        pila=new LEPila<Integer>();
    }

    @Override
    public String ejecutarOpciones() {
       System.out.println("");
        System.out.println("SubMenú Pila de Enteros");
        System.out.println("Seleccione una opción:");
        System.out.println("0. Volver");
        System.out.println("");
        System.out.println("1. Crear Pila de enteros");
        System.out.println("2. Introducir elementos en la Pila");
        System.out.println("3. Duplicar Pila");
        System.out.println("4. Invertir Pila");
        System.out.println("5. Comprobar Base de la Pila");
        String s=MyInput.readString();
        switch(s){
                case "0": {return "n";}
                case "1": {crearPilaEnteros();return "s";}
                case "2": {apilarElemPos();return "s";}
                case "3": {duplicarPila();return "s";}
                case "4": {invertirPilaPC();return "s";}
                case "5": {comprobarBasePila();return"s";}
                default: {System.out.println("Opción no válida. Vuelva a
intentarlo."); return "s";}
            }
    }

    public void crearPilaEnteros(){
        if(!pila.esVacia()){
            System.out.println("Pila de enteros existe!!");
            vaciarPila();
            System.out.println("Pila de enteros vaciada con exito.");
        }
```

```
        else{
            pila = new LEPila<Integer>();
            System.out.println("Pila vacía creada.");
            System.out.println("Pulse <Intro> para continuar...");
            Scanner scanner = new Scanner(System.in);
            scanner.nextLine();
        }
    }

    private void vaciarPila(){
        while(!pila.esVacia()){
            pila.desapilar();
        }
    }

    public void apilarElemPos(){
        int num=-1;

        System.out.println("Introduzca el entero que quiere almacenar en la
Pila (para finalizar pulse 0):");
        while(num!=0){
            num = MyInput.readInt();
            if(num>0){
                pila.apilar(num);
            }
        }

        LEPila<Integer> aux = new LEPila<Integer>();

        System.out.println("PILA:");
        while(!pila.esVacia()) {
            int numero = pila.desapilar();
            System.out.println(numero);
            aux.apilar(numero);
        }
        while(!aux.esVacia()){
            int numero = aux.desapilar();
            pila.apilar(numero);
        }
    }

    public void duplicarPila() {
        LEPila<Integer> copia = new LEPila<Integer>();
        duplicarPilaAux(copia);
        System.out.println("Pila Original:");
        while (!pila.esVacia()){
            System.out.println(pila.desapilar());
        }

        System.out.println("Pila Copiada:");
```

```
        System.out.println("NUEVA_PILA");

        while (!copia.esVacia()){
            System.out.println(copia.desapilar());
        }
    }


    private void duplicarPilaAux( LEPila<Integer> copia) {
        if (pila.esVacia()) return;
        Integer elemento = pila.desapilar();
        duplicarPilaAux(copia);
        copia.apilar(elemento);
        pila.apilar(elemento);
    }


    public void invertirPilaPC() {
        ArrayCola<Integer> cola = new ArrayCola<Integer>();

        System.out.println("Pila Original");
        System.out.println("PILA");
        // Mover los elementos de la pila original a la cola
        while (!pila.esVacia()) {
            int elem = pila.tope();
            cola.encolar(pila.desapilar());
            System.out.println(elem);
        }

        // Mover los elementos de la cola a la pila
        while (!cola.esVacia()) {
            pila.apilar(cola.primero());
            cola.desencolar();
        }

        // Imprimir la pila invertida
        System.out.println("La pila invertida queda...");
        System.out.println("PILA");
        while (!pila.esVacia()) {
            System.out.println(pila.tope());
            pila.desapilar();
        }
    }

    public void comprobarBasePila(){
        int num,valor;

        LEPila<Integer> copia = new LEPila<Integer>();
        LEPila<Integer> aux = new LEPila<Integer>();
        LEPila<Integer> cp2 = new LEPila<Integer>();
```

```
        while(!pila.esVacia()){
            aux.apilar(pila.desapilar());
        }

        while(!aux.esVacia()){
            int elemento = aux.desapilar();
            pila.apilar(elemento);
            copia.apilar(elemento);
        }

        while(!copia.esVacia()){
            cp2.apilar(copia.desapilar());
        }

        num = comprobarBasePila(cp2);

        if(num==pila.tamaño()){
            valor=0;
        }
        else if(num > pila.tamaño()){
            valor=1;
        }
        else if(num < pila.tamaño() && num!=-1){
            valor=2;
        }
        else{
            valor=-1;
        }

        System.out.println("PILA");
        while (!pila.esVacia()) {
            System.out.println(pila.desapilar());
        }

        switch (valor) {
            case 0: {System.out.println("El elemento de la base de la pila
es IGUAL que el número de elementos de la misma");}
                break;
            case 1: {System.out.println("El elemento de la base de la pila
es MAYOR que el número de elementos de la misma");}
                break;
            case 2: {System.out.println("El elemento de la base de la pila
es MENOR que el número de elementos de la misma");}
                break;
            default: {System.out.println("ERROR!!");}
        }
    }

    private static int comprobarBasePila(LEPila<Integer> p){
```

```
        int elemento=-1;
        if(!p.esVacia() && p.tamaño()!=1){
            elemento = p.desapilar();
            comprobarBasePila(p);
        }
        return elemento;
    }
}
```

Clase SubMenuColaEnteros:

```
package Menus;

import EntradaSalida.MyInput;
import java.util.Scanner;
import librerias.estructurasDeDatos.lineales.ArrayCola;

/**
 *
 * @author Usuario
 */
public class SubMenuColaEnteros extends Menus {
    private ArrayCola<Integer> nuevaCola;

    public SubMenuColaEnteros() {
        nuevaCola=new ArrayCola<Integer>();
    }

    @Override
    public String ejecutarOpciones() {
       System.out.println("");
        System.out.println("SubMenú Cola de Enteros");
        System.out.println("Seleccione una opción:");
        System.out.println("0. Volver");
        System.out.println("");
        System.out.println("1. Crear Cola de enteros");
        System.out.println("2. Introducir elementos en la Cola");
        System.out.println("3. Duplicar Cola");
        System.out.println("4. Repetir n-ésimo");
        String s=MyInput.readString();
        switch(s){
            case "0": {return "n";}
            case "1": {crearColaEnteros();return "s";}
            case "2": {encolarElemPos();return "s";}
            case "3": {copiarCola();return "s";}
            case "4": {repetirNesimo();return "s";}
            default: {System.out.println("Opción no válida. Vuelva a
intentarlo."); return "s";}
        }
    }
```

```
    public void crearColaEnteros(){
        if(!nuevaCola.esVacia()){
            System.out.println("Cola de enteros existe!!");
            vaciarCola();
            System.out.println("Cola de enteros vaciada con exito.");
        }
        else{
            nuevaCola = new ArrayCola<Integer>();
            System.out.println("Cola vacía creada.");
            System.out.println("Pulse <Intro> para continuar...");
            Scanner scanner = new Scanner(System.in);
            scanner.nextLine();
        }
    }

    private void vaciarCola(){
        while(!nuevaCola.esVacia()){
            nuevaCola.desencolar();
        }
    }

    public void encolarElemPos(){
        int num=-1;

        System.out.println("Introduzca el entero que quiere almacenar en la
Cola (para finalizar pulse 0):");
        while(num!=0){
            num = MyInput.readInt();
            if(num>0){
                nuevaCola.encolar(num);
            }
        }

        ArrayCola<Integer> aux = new ArrayCola<Integer>();

        System.out.println("COLA:");
        while(!nuevaCola.esVacia()) {
            int numero = nuevaCola.desencolar();
            System.out.println(numero);
            aux.encolar(numero);
        }
        while(!aux.esVacia()){
            int numero = aux.desencolar();
            nuevaCola.encolar(numero);
        }
    }

    public void copiarCola() {
        ArrayCola<Integer> aux = new ArrayCola<Integer>();
        copiarColaAux(aux);
```

```
        System.out.println("Cola Original:");

        while(!nuevaCola.esVacia()){
            System.out.println(nuevaCola.desencolar());
        }

        System.out.println("Cola Copiada:");
        System.out.println("NUEVA_COLA");

        while(!aux.esVacia()){
            System.out.println(aux.desencolar());
        }
    }

    private void copiarColaAux(ArrayCola<Integer> aux) {
        if (nuevaCola.esVacia()) return;
        Integer elemento = nuevaCola.desencolar();
        copiarColaAux(aux);
        aux.encolar(elemento);
        nuevaCola.encolar(elemento);
    }

    public void repetirNesimo() {
        System.out.println("Introduce el valor de n:");
        int n = MyInput.readInt();

        if (n > 0 && n <= this.nuevaCola.tallaActual()) {
            int elemNesimo = repetirNesimoAux(n);
            int repeticiones = (int) elemNesimo;
            for (int i = 1; i <= repeticiones; i++) {
                this.nuevaCola.encolar(elemNesimo);
            }
            System.out.println("COLA REPLICADA");

            while(!this.nuevaCola.esVacia()){
                System.out.println(this.nuevaCola.desencolar());
            }
        } else {
            System.out.println("La posición n no es válida.");
        }
    }

    private int repetirNesimoAux(int n) {
        int elem;
        if (n == 1) {
            elem = this.nuevaCola.desencolar();
        } else {
            int elemAnterior = repetirNesimoAux(n-1);
            elem = this.nuevaCola.desencolar();
            this.nuevaCola.encolar(elemAnterior);
```

```
        }
        return elem;
    }
}
```

Clase comprobarFrasePalindroma:

```
package Menus;

import EntradaSalida.MyInput;
import librerias.estructurasDeDatos.lineales.ArrayCola;
import librerias.estructurasDeDatos.lineales.LEPila;

/**
 *
 * @author Usuario
 */
public class comprobarFrasePalindroma extends Menus {


    @Override
    public String ejecutarOpciones() {
        System.out.println("Opción 3: Comprobar Frase Palíndroma");
        System.out.println("seleccione una opción:");
        System.out.println("0. Salir de la opción 3 del Menú Principal");
        System.out.println("1. Comprobar Frase Palíndroma");

        String s= MyInput.readString();

        switch(s){
            case "0": {return "n";}
            case "1": {
                boolean cmp = comprobarFrasePalindroma();
                if(cmp){
                    System.out.println("La frase es palíndroma.");
                }
                else{
                    System.out.println("La frase no es palíndroma.");
                }

                return "s";
            }
            default: {System.out.println("Opción no válida. Vuelva a
intentarlo."); return "s";}
        }
    }

    public static boolean comprobarFrasePalindroma(){
        System.out.println("Introduzca la frase:");
        String frase = MyInput.readString();
        frase = frase.replaceAll("[^a-zA-Z0-9]", "");
```

```
        char[] ch = frase.toCharArray();

        LEPila<Character> p = new LEPila<Character>();
        ArrayCola<Character> c = new ArrayCola<Character>();

        for(int i = 0; i<frase.length(); i++){
            p.apilar(ch[i]);
            c.encolar(ch[i]);
        }

        while(!p.esVacia() && !c.esVacia()){
            char pilaTop = p.desapilar();
            char colaFront = c.desencolar();

            if(Character.toLowerCase(pilaTop) !=
Character.toLowerCase(colaFront)){
                return false;
            }
        }

        return true;
    }
}
```

Clase MyInput:

```
package EntradaSalida;

import java.io.*;
import java.util.ArrayList;
/**
 * Esta clase contiene los metodos y atributos de los procesos E/S que se
precisan en este programa.
 * @author Raul Colindres y Alfredo Sobrados
 */
public class MyInput {
    /**
     * Metodo que lee una cadena de caracteres desde el teclado.
     * @return string devuelve el string leído a traves de la pantalla
     */
public static String readString() {
    BufferedReader br = new BufferedReader(new
    InputStreamReader(System.in),1);
    String string="";
    try {
        string = br.readLine(); }
    catch (IOException ex) {
        System.out.println(ex); }
    return string; }

/**
```

```
 * Metodo que lee un dato tipo int desde el teclado.
 * @return Integer devuelve el entero leído a traves de la pantalla
 * @throws NumberFormatException arroja esta excepción cuando el formato del
número no es correcto
 * Postcondition: El valor que devuelve debe ser un número entero
 */
public static int readInt() throws NumberFormatException{
    return Integer.parseInt(readString());
}

/**
 * Metodo que lee un fichero que este en la carpeta raíz del proyecto.
 * @param nombreFichero es el string que contiene el nombre del fichero
 * @return v ArrayList de Strings que contiene la información del fichero
leído
 */
public static ArrayList <String> leeFichero(String nombreFichero){
    ArrayList <String> v = new ArrayList <String>();
    File fichero=null;
    FileReader fr=null;
    BufferedReader br=null;
    try{
        fichero=new File(nombreFichero);
        fr=new FileReader(fichero);
        br=new BufferedReader(fr);
        String linea;
        while ((linea=br.readLine())!=null){
        v.add(linea);}
    }
    catch (Exception e){
        e.printStackTrace();
    }
    finally {
    try {
        if (null!= fr){
        fr.close();
        br.close();}
        }
        catch (Exception e1){
        e1.printStackTrace();
        }
    }
    return v;
    }
/**
 * Metodo que realiza la serialización y deserialización.
 * @param <A> parametro descocido para mi
 * @param a parametro descocido para mi
```

```
 * @param nombreFichero nombre del fichero a guardar en en la carpeta raíz
del proyecto desde el programa o cargar desde la carpeta raíz hacía el
programa
 */
    public static <A> void serialize(A a, String nombreFichero) {
        System.out.println("Serializando...");
        try {
            FileOutputStream fos = new FileOutputStream(nombreFichero) ;
            ObjectOutputStream oos = new ObjectOutputStream(fos) ;
            oos.writeObject(a) ;
        } catch (Exception e) {
            System.err.println("Problem: "+e) ;
        }
    }
    public static <A> A deserialize(String nombreFichero) {
        System.out.println("DeSerializing...");
        try {
            FileInputStream fis = new FileInputStream(nombreFichero) ;
            ObjectInputStream iis = new ObjectInputStream(fis) ;
            return (A) iis.readObject() ;
        } catch (Exception e) {
            System.err.println("Problem: "+e) ;
        }
        return null ;
}

}
```

Clase ArrayCola:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.*;

/**
 *
 * @author Usuario
 * @param <E>
 */
public class ArrayCola<E> implements Cola<E>{
    protected E elArray[];
    protected int fin, primero, tallaActual;
    protected static final int CAPACIDAD_POR_DEFECTO = 200;

    public ArrayCola(){
        elArray = (E[]) new Object[CAPACIDAD_POR_DEFECTO];
        tallaActual=0;
        primero=0;
        fin = -1;
    }
```

```
@Override
public void encolar(E x) {
    if( tallaActual==elArray.length) duplicarArray();
    fin=incrementa(fin);
    elArray[fin]=x;
    tallaActual++;
}

@Override
public E desencolar() {
    E elPrimero = elArray[primero];
    primero = incrementa(primero);
    tallaActual--;
    return elPrimero;
}

@Override
public E primero() {
    return elArray[primero];
}

@Override
public boolean esVacia() {
    return(tallaActual==0);
}

private int incrementa(int indice) {
    if(++indice==elArray.length)indice=0;
    return indice;
}

@Override
public String toString(){
    String res="";
    int aux=primero;
    for(int i =0;i<tallaActual;i++, aux=incrementa(aux))
        res += elArray[aux]+"";
    return res;
}

private void duplicarArray() {
    E nuevo[]=(E[]) new Object[elArray.length*2];
    for(int i =0;i<tallaActual;i++, primero=incrementa(primero))
        nuevo[i]=elArray[primero];
    elArray=nuevo;
    primero=0;
    fin=tallaActual - 1;
}

public int tallaActual() {
```

```
        return tallaActual;
    }

}
```

Clase ArrayPila:

```
package librerias.estructurasDeDatos.lineales;

import librerias.estructurasDeDatos.modelos.*;

/**
 *
 * @author Usuario
 * @param <E>
 */
public class ArrayPila<E> implements Pila<E>{
    protected E elArray[];
    protected int tope;
    protected static final int CAPACIDAD_POR_DEFECTO = 200;

    public ArrayPila(){
        elArray= (E[]) new Object[CAPACIDAD_POR_DEFECTO];
        tope=-1;
    }

    @Override
    public void apilar(E x) {
        if(tope +1 == elArray.length) duplicarArray();
        tope++; elArray[tope]=x;
    }

    @Override
    public E desapilar() {
        E elUltimo = elArray[tope];
        tope--;
        return elUltimo;
    }

    @Override
    public E tope() {
        return elArray[tope];
    }

    @Override
    public boolean esVacia() {
        return ( tope==-1 );
    }

    @Override
    public String toString(){
```

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

    private void duplicarArray() {
        E nuevoArray[]=(E[]) new Object[elArray.length*2];
        for(int i=0;i<=tope;i++) nuevoArray[i]=elArray[i];
        elArray = nuevoArray;
    }

}
```

Clase LECola:

```
package librerias.estructurasDeDatos.lineales;

import java.util.NoSuchElementException;
import librerias.estructurasDeDatos.modelos.Cola;

/**
 *
 * @author Usuario
 * @param <E>
 */
public class LECola<E> implements Cola<E>{
    private NodoLEG<E> primero, ultimo; // puntero al nodo que está el
primero y el último de la cola
    private int tamaño; //tamaño de la cola

    public LECola(){
        primero = null; // la cola está vacía al inicio
        ultimo = null;
        tamaño = 0;
    }

    @Override
    public void encolar(E x) {
        NodoLEG<E> nuevo = new NodoLEG<E>(x);
        if(esVacia()){
            nuevo = primero;
            nuevo = ultimo;
        }
        ultimo.setSiguiente(nuevo);
        ultimo = nuevo;
        tamaño++;
    }

    @Override
    public E desencolar() {
        if(esVacia()){
```

```
            throw new NoSuchElementException("La cola está vacía"); //
lanzamos una excepción si la cola está vacía
        }
        E dato = primero.getDato(); //obtengo el dato que quiero desencolar
        primero = primero.getSiguiente(); //asigno la referencia del primero
al siguiente en la cola
        tamaño--; //disminuyo el tamaño de la cola
        return dato; //devuelvo el dato que he desencolado
    }

    @Override
    public E primero() {
        if (esVacia()) {
            throw new NoSuchElementException("La cola está vacía"); //
lanzamos una excepción si la cola está vacía
        }
        return primero.getDato(); // obtenemos el dato del nodo que está el
primero o en el frente de la cola
    }

    @Override
    public boolean esVacia() {
        return primero == null; // la cola está vacía si el primero es null
    }

    public int tamaño(){
        return tamaño;
    }

}
```

Clase LEG:

```
package librerias.estructurasDeDatos.lineales;

/**
 *
 * @author Usuario
 * @param <E>
 */
public class LEG <E> {
    protected NodoLEG<E> primero;

    public LEG(){
       primero=null;
    }

    public void insertar(E x){
        primero = new NodoLEG<E>(x, primero);
    }
```

```
    public void insertarEnFin(E x){
        NodoLEG<E> nl = new NodoLEG<E>(x);
        NodoLEG<E> aux = primero;
        if(aux==null)primero=nl;
        else{
            while(aux.siguiente!=null)aux = aux.siguiente;
            //aux referencia al último nodo de la lista
            aux.siguiente=nl;
        }
    }

    @Override
    public String toString(){
        String res="";
        for(NodoLEG<E> aux=primero; aux!=null;aux=aux.siguiente){
            res+=aux.dato.toString()+"\n";
        }
        return res;
    }

    public boolean eliminar(E x){
        NodoLEG<E> aux=primero; NodoLEG<E> ant=null; boolean res = false;
        while(aux!=null && !aux.dato.equals(x)){
            ant = aux; aux=aux.siguiente;
        }
        if(aux!=null){
            res=true;
            if(ant==null)primero=aux.siguiente;
            else ant.siguiente=aux.siguiente;
        }
        return res;
    }

    public E recuperar(int i){
        NodoLEG<E> aux; int j;
        for(aux=primero, j=0; j<i;aux=aux.siguiente, j++){;}
        return aux.dato;
    }

}
```

Clase LEPila:

```
package librerias.estructurasDeDatos.lineales;

import java.util.NoSuchElementException;
import librerias.estructurasDeDatos.modelos.Pila;

/**
 *
 * @author Usuario
```

```
 * @param <E>
 */
public class LEPila<E> implements Pila<E>{
    private NodoLEG<E> cima; // puntero al nodo que está en la cima de la
pila
    private int tamaño; // tamaño de la pila

    public LEPila() {
        cima = null; // la pila está vacía al inicio
        tamaño = 0;
    }

    @Override
    public void apilar(E x) {
        NodoLEG<E> nuevo = new NodoLEG<E>(x); // creamos un nuevo nodo con
el dato a apilar
        nuevo.setSiguiente(cima); // el siguiente del nuevo nodo es el nodo
que estaba en la cima
        cima = nuevo; // actualizamos la cima de la pila
        tamaño++; // aumentamos el tamaño de la pila
    }

    @Override
    public E desapilar() {
       if (esVacia()) {
            throw new NoSuchElementException("La pila está vacía"); //
lanzamos una excepción si la pila está vacía
        }
        E dato = cima.getDato(); // obtenemos el dato del nodo que está en
la cima
        cima = cima.getSiguiente(); // actualizamos la cima de la pila
        tamaño--; // disminuimos el tamaño de la pila
        return dato;
    }

    @Override
    public E tope() {
        if (esVacia()) {
            throw new NoSuchElementException("La pila está vacía"); //
lanzamos una excepción si la pila está vacía
        }
        return cima.getDato(); // obtenemos el dato del nodo que está en la
cima
    }

    public NodoLEG getCima(){
        return this.cima;
    }

    @Override
```

```
    public boolean esVacia() {
        return cima == null; // la pila está vacía si la cima es null
    }

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

Clase NodoLEG:

```
package librerias.estructurasDeDatos.lineales;

/**
 *
 * @author Usuario
 * @param <E>
 */
public class NodoLEG<E> extends LEG{
    protected E dato;
    protected NodoLEG<E> siguiente;
    /**
     * Método constructor parametrizado
     * @param dato
     */
    public NodoLEG(E dato){
        this(dato, null);
    }
    /**
     * Método constructor parametrizado
     * @param dato
     * @param siguiente
     */
    public NodoLEG(E dato, NodoLEG<E> siguiente){
        this.dato = dato;
        this.siguiente = siguiente;
    }

    public E getDato() {
        return dato;
    }

    public NodoLEG<E> getSiguiente() {
        return siguiente;
    }

     public void setSiguiente(NodoLEG<E> siguiente) {
        this.siguiente = siguiente;
    }
}
```

Interface Cola:

```
package librerias.estructurasDeDatos.modelos;

/**
 *
 * @author Usuario
 * @param <E>
 */
public interface Cola<E> {
    public void encolar(E x);
    public E desencolar();
    public E primero();
    public boolean esVacia();
}
```

Interface Pila:

```
package librerias.estructurasDeDatos.modelos;

/**
 *
 * @author Usuario
 * @param <E>
 */
public interface Pila<E> {
    public void apilar(E x);
    public E desapilar();
    public E tope();
    public boolean esVacia();
}
```

Clase Principal:

```
package tadspilaycola;

import Menus.MenuPrincipal;

/**
 *
 * @author Usuario
 */
public class TADsPilaYCola {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        MenuPrincipal mp = new MenuPrincipal();
        mp.ejecutar();
    }

}
```

Pruebas de ejecución

Pruebas de ejecución (pantallazos de ejecución de la aplicación)

Captura del menú principal:

```
Menú Principal
seleccione una opción:
0. Salir
1. Submenú Pila de Enteros
2. Submenú Cola de Enteros
3. Comprobar frase palíndroma
```

Captura de la opción 1 del menú principal: Submenú Pila de Enteros:

```
SubMenú Pila de Enteros
Seleccione una opción:
0. Volver

1. Crear Pila de enteros
2. Introducir elementos en la Pila
3. Duplicar Pila
4. Invertir Pila
5. Comprobar Base de la Pila
```

Captura de la opción 1 del Submenú Pila de enteros: Crear Pila de enteros:

```
Pila vacía creada.
Pulse <Intro> para continuar...
```

Captura de la opción 2 del Submenú Pila de enteros: Introducir elementos en la Pila:

```
Introduzca el entero que quiere almacenar en la Pila (para finalizar pulse 0):
4
18
47
87
23
0
PILA:
23
87
47
18
4
```

Captura de la opción 3 del Submenú Pila de enteros: Duplicar Pila:

```
Pila Original:
23
87
47
18
4
Pila Copiada:
NUEVA_PILA
23
87
47
18
4
```

Captura de la opción 4 del Submenú Pila de enteros: Invertir Pila:

```
Pila Original
PILA
23
87
47
18
4
La pila invertida queda...
PILA
4
18
47
87
23
```

Captura de la opción 5 del Submenú Pila de enteros: Comprobar Base de la Pila:

```
PILA
23
87
47
18
4
El elemento de la base de la pila es MENOR que el número de elementos de la misma
```

Captura de la opción 2 del menú principal: Submenú Cola de Enteros:

```
SubMenú Cola de Enteros
Seleccione una opción:
0. Volver

1. Crear Cola de enteros
2. Introducir elementos en la Cola
3. Duplicar Cola
4. Repetir n-ésimo
```

Captura de la opción 1 del Submenú Cola de enteros: Crear Cola de Enteros:

```
Cola vacía creada.
Pulse <Intro> para continuar...
```

Captura de la opción 2 del Submenú Cola de enteros: Introducir elementos en la Cola:

```
Introduzca el entero que quiere almacenar en la Cola (para finalizar pulse 0):
12
3
4
7
9
0
COLA:
12
3
4
7
9
```

Captura de la opción 3 del Submenú Cola de enteros: Duplicar Cola:

```
Cola Original:
5
4
3
2
1
Cola Copiada:
NUEVA_COLA
5
4
3
2
1
```

Captura de la opción 4 del Submenú Cola de enteros: Repetir n-ésimo:

```
Introduce el valor de n:
3
COLA REPLICADA
4
5
1
2
3
3
3
```

Captura de la opción 3 del Menú Principal: Comprobar frase palíndroma:

```
Opción 3: Comprobar Frase Palíndroma
seleccione una opción:
0. Salir de la opción 3 del Menú Principal
1. Comprobar Frase Palíndroma
1
Introduzca la frase:
Dabale arroz a la zorra el abad
La frase es palíndroma.
```

Captura del Menú Principal con el mensaje de despedida de la aplicación:

```
Menú Principal
seleccione una opción:
0. Salir
1. Submenú Pila de Enteros
2. Submenú Cola de Enteros
3. Comprobar frase palíndroma
0
Gracias por utilizar nuestros TADs Pila y Cola
```