PRÁCTICA 1
LISTAS CIRCULARES

PROGRAMACIÓN Y ESTRUCTURA DE DATOS

Ángel Martín Bartolomé

Mario Vázquez Onrubia

1. Introducción

Para la realización de esta práctica hemos elegido el módulo ListaC, el cual consisten en implementarla por mediación de una lista circular. Además, se utilizará como tipo de datos Persona, que consta de DNI, Fecha de nacimiento, Nombre y Apellidos.

Las listas circulares son un tipo de listas genéricas enlazadas, con referencia al último nodo en la que el siguiente al último nodo es el primer nodo. En este caso, podremos prescindir de una referencia al primer nodo, ya que haciendo `ultimo.siguiente` se obtendría el primer nodo, es decir, el ultimo nodo siempre referencia al primero.

A continuación, se van a presentar los paquetes, clases utilizadas y los métodos de cada una de ellas explicando brevemente en qué consisten.

2. Paquetes, clases y métodos

Las clases utilizadas, con sus correspondientes métodos, son las siguientes:

- Paquete `entrada`:
  - Clase `MyInput.java`
    - Método `readString()`: lee una cadena de caracteres por teclado
    - Método `readInt()`: lee un número de tipo entero por teclado
- Paquete `excepciones`
  - Clase `EmptyListException`: excepción que se lanzará cuando la lista esté vacia
  - Clase `InvalidDniException`: excepción que se lanzará cuando el dni leído no cumpla el formato pedido
  - Clase `NotCreatedException`: excepción que se lanzará cuando la lista no haya sido creada
  - Clase `NotFoundPersonException`: excepción que se lanzará cuando no se encuentre una persona
- Paquete `listaCircular`
  - Clase `LEGCircular`
    - Método tal`la`: devuelve un entero con la talla de la lista
    - Método `getNodo`: devuelve un objeto de tipo E que está en la posición que recibe por parámetro
    - Método `insertarEnFin`: recibe un objeto por parámetro y lo inserta a final de la lista
    - Método `insertarOrdenado`: inserta un nodo de manera ordenada en la lista según el método `compareTo()` del objeto que recibe por parámetro.
    - Método `insertarOrdenadoInvertido()`: inserta un nodo de manera ordenada, pero de manera inversa al anterior (este se utilizará cuando la lista esté invertida)
    - Método `eliminar`: elimina el nodo en el cual se encuentra el objeto que recibe por parámetro
    - Método `imprimir`: imprime la lista completa de manera ordenada

        - Método `imprimirInversa`: imprime la lista completa de manera inversa a la anterior
        - Método `vacia`: lanza una excecpción si está vacía o devuelve false si no lo está
        - Método `invertida`: devuelve un objeto de tipo LEGCircular, el cual es la inversión de la lista actual
    - Clase `NodoLEG`: es la clase de los nodos que se van a insertar en la lista
    - Clase `Persona`
        - Método `compareTo(Persona o)`: se reescribe este método para comparar las personas por dni. Si el dni de la persona actual es menor que la pasada por parámetro, devuelve -1, si es igual, devuelve 0, y si es mayor, 1.
        - Método `toString`: se ha reescrito el método toString para dar formato a la impresión de una persona.
        - Además de los métodos anteriores, están los gets y sets pertinentes.
- Paquete `practica1_ped`
    - Clase `Practica1_PED`
        - Método `menú`: imprime el menú del programa
        - Método `crearLista`: crea la lista circular
        - Método `longitudLista`: Muestra la longitud de la lista por pantalla
        - Método `insertarPersona`: Método para insertar una persona en la lista
        - Método `acceso`: Método que muestra la persona del dni que se pide
        - Método `eliminar`: Método para eliminar una persona
        - Método `destruirLista`: para destruir una lista
        - Metodo `creada`: devuelve true si está creada y lanza una excepción si no está.
        - Método `pedirDNI`: para pedir el DNI con formato
        - Método `imprimir`: para imprimir la lista
        - Método `imprimirInversa`: para imprimir la lista de manera inversa
        - Método `invertir`: para invertir la lista
    - Clase `Aplicacion`: clase que ejecuta el método menú.

3. Código comentado

Clase MyInput:

```
package entrada;

import java.io.*;
import java.util.*;
/**
 *
 * @author Mario
 */
public class MyInput {

 /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {}
       // Lee una cadena de caracteres desde el teclado
```

```
    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; }
    // Lee un dato tipo int desde el teclado
    public static int readInt() {
    return Integer.parseInt(readString()); }


}
```

Clase EmptyListException:

```
package excepciones;

/**
 *
 * @author Mario
 */
public class EmptyListException extends Exception{
    public EmptyListException (String mensaje){
                super(mensaje);
        }
}
```

Clase InvalidDniException:

```
package excepciones;

/**
 *
 * @author Mario
 */
public class InvalidDniException extends Exception{
    public InvalidDniException (String mensaje){
                super(mensaje);
        }
}
```

Clase NotCreatedException:

```
package excepciones;

/**
 *
 * @author Mario
 */
public class NotCreatedException extends Exception {
    public NotCreatedException (String mensaje){
                super(mensaje);
        }
}
```

Clase NotFoundPersonException:

```
package excepciones;

/**
 *
 * @author Mario
 */
public class NotFoundPersonException extends Exception{
     public NotFoundPersonException (String mensaje){
                super(mensaje);
        }
}
```

Clase LEGCircular:

```
package listaCircular;

import excepciones.EmptyListException;

/**
 *
 * @author Mario
 */
public class LEGCircular <E extends Comparable <E>> {

    protected NodoLEG<E> ultimo; //referencia al último
    protected int talla;

    public LEGCircular()
    {
        ultimo = null;
    }

    //Metodo que devuelve la talla de la lista
    public int talla()
    {
        return this.talla;
    }

    //Método que devuelve un objeto recibiendo por parametro una posición
    public E getNodo(int posicion)
    {
        //Si no esta vacia la lista
        if(ultimo != null)
        {
            //Si la posicion recibida está dentro del dominio de la lista
            if(posicion >= 0 && posicion < this.talla)
            {
                //Si es la prima
                if(posicion == 0)
                {
                    //Devolvemos el primero, es decir, el siguiente al ultimo
                    return ultimo.siguiente.dato;
                }
                //Si no es la primera
                else
                {
                    //Declaramos un nodo auxiliar inicializandolo al primero
                    NodoLEG<E> aux = ultimo.siguiente;

                    //Recorremos la lista hasta la posicion
                    for(int i = 0; i < posicion; i++)
                    {
                        //Avanzamos un nodo en la lista
                        aux = aux.siguiente;
                    }
```

```
                    //Devolvemos el nodo de la posicion
                    return aux.dato;
                }
            }

        }

        //Si esta vacia
        else
        {
            System.out.println("Posicion no permitida");
            return null;
        }
        //Devolvemos null
        return null;
    }

    //Metodo para insertar en el final de la lista el objeto que recibe por parametro
    public void insertarEnFin(E x)
    {
        //Inicializamos un nodo con el objeto obtenido por parametro
        NodoLEG<E> nuevo = new NodoLEG<E>(x);


        //Si la lista está vacia hacemos que el nodo a insertar sea el primero y a su
vez el ultimo
        if(this.talla == 0)
        {

            ultimo = nuevo; //hacemos ultimo
            ultimo.siguiente = nuevo;//hacemos primero
            //Incrementamos en uno la talla
            this.talla++;
        }
        //Si la lista no está vacia, hacemos que el nuevo nodo apunte al primero, y que
este pase a ser el último
        else
        {
            //El siguiente del nuevo será el primero
            nuevo.siguiente = ultimo.siguiente;
            //El siguiente al que era el ultimo, será el nuevo
            ultimo.siguiente = nuevo;
            //El ultimo nodo será el nuevo
            ultimo = nuevo;
            //Incrementamos la talla en uno
            this.talla++;
        }

    }

    //Metodo para insertar un nodo de manera ordenada según el metodo compareTo() del
objeto que recibe
    public  void  insertarOrdenado(E x)
    {
        //Inicializamos un nodo con el objeto obtenido por parametro
        NodoLEG<E> nuevo = new NodoLEG<E>(x);

        //Si la lista está vacia el nuevo nodo será el ultimo y a su vez el primero y se
apuntará a si mismo
        if(ultimo == null)
        {
            ultimo = nuevo; //hacemos ultimo
            ultimo.siguiente = nuevo;//hacemos primero
            //incrementamos la talla en uno
            this.talla++;
        }
        //Si no está vacia
        else
        {
            //Si el nuevo es menor que el primero de todos
            if(x.compareTo(ultimo.siguiente.dato) < 0)
            {
                //Hacemos que el siguiente a nuevo sea el que era el primero
                nuevo.siguiente = ultimo.siguiente;
                //Y que el primero sea el nuevo
                ultimo.siguiente = nuevo;
                //Incrementamos la talla
```

```
                this.talla++;
            }
            else
            {
                //Creamos dos nodos auxiliares inicializandolos al primero
                NodoLEG<E> aux = ultimo.siguiente;
                NodoLEG<E> ant = ultimo.siguiente;

                //Mientras el nodo a insertar sea mayor que el auxiliar y el siguiente
al auxiliar no sea el primero,
                //es decir, que no hayamos recorrido la lista entera
                //Mientras el objeto a insertar sea mayor que el nodo con el que se está
comparando y no se llegue al final
                //de la lista
                while((x.compareTo(aux.dato) > 0) && (aux.siguiente !=
ultimo.siguiente))
                {
                    //El nodo anterior tomará el valor del auxiliar
                    ant = aux;
                    //Y el auxiliar apuntará al siguiente al que era
                    aux = aux.siguiente;
                }

                //Si insertamos en la ultima posicion, el siguiente al auxiliar deberá
ser el primero
                //y el siguiente al que antes era el ultimo, el auxiliar.
                if(x.compareTo(aux.dato) > 0)
                {
                    //El siguiente al nuevo será el que era el primero
                    nuevo.siguiente = ultimo.siguiente;
                    //Hacemos que el siguiente al que era el ultimo, sea el nuevo
                    ultimo.siguiente = nuevo;
                    //Y que el ultimo sea el nuevo
                    ultimo = nuevo;
                    //Incrementamos la talla
                    this.talla++;
                }
                //Si insertamos en medio de la lista, puesto que estabamos comparando
con el siguiente al nodo actual,
                //Habrá que hacer que el nuevo nodo sea el auxiliar y el siguiente al
anterior al auxiliar, el nuevo
                else
                {
                    //El siguiente al nuevo será el auxiliar
                    nuevo.siguiente = aux;
                    //Y el siguiente al anterior al auxiliar, el nuevo
                    ant.siguiente = nuevo;
                    //Incrementamos la talla
                    this.talla++;
                }
            }
        }
    }

    //Metodo para insertar invertido (esto se usará cuando se invierta la lista, para
que se siga insertando ordenado)
    public void insertarOrdenadoInvertido(E x)
    {
        //Inicializamos un nodo con el objeto obtenido por parametro
        NodoLEG<E> nuevo = new NodoLEG<E>(x);

        //Si la lista está vacia el nuevo nodo será el ultimo y a su vez el primero y se
apuntará a si mismo
        if(ultimo == null)
        {
            ultimo = nuevo; //Hacemos ultimo
            ultimo.siguiente = nuevo; //Hacemos primero
            //Incrementamos en uno la talla
            this.talla++;
        }
        //Si no está vacia
        else
        {
            //Si el nuevo es mayor que el primero de todos, se inserta al principio
puesto que está invertida
            if(x.compareTo(ultimo.siguiente.dato) > 0)
            {
```

```
                //Hacemos que el siguiente a nuevo sea el que era el primero
                nuevo.siguiente = ultimo.siguiente;
                //Y que el primero sea el nuevo
                ultimo.siguiente = nuevo;
                //incrementamos en uno la talla
                this.talla++;
            }
            //Si el nodo a insertar no es el mayor de todos
            else
            {
                //Declaramos dos nodos auxiliares, inicializandolos al primero
                NodoLEG<E> aux = ultimo.siguiente;
                NodoLEG<E> ant = ultimo.siguiente;

                //Mientras el nodo a insertar sea menor que el auxiliar y el siguiente
al auxiliar no sea el primero,
                //es decir, que no hayamos recorrido la lista entera
                //Mientras el objeto a insertar sea menor que el objeto auxiliar
                while((x.compareTo(aux.dato) < 0) && (aux.siguiente !=
ultimo.siguiente))
                {
                    //Hacemos que el anterior sea igual al auxiliar
                    ant = aux;
                    //Avanzamos un nodo
                    aux = aux.siguiente;
                }

                //Al salir del bucle, el auxiliar saldrá siendo el ultimo, y ant saldrá
siendo su anterior
                //Si insertamos en la ultima posicion, el siguiente al auxiliar deberá
ser el primero
                //y el siguiente al que antes era el ultimo, el auxiliar.
                if(x.compareTo(aux.dato) < 0)
                {
                    //El siguiente al nuevo será el primero
                    nuevo.siguiente = ultimo.siguiente;
                    //El siguiente del ultimo será el nuevo
                    ultimo.siguiente = nuevo;
                    //El ultimo será el nuevo
                    ultimo = nuevo;
                    //Incrementamos en uno la talla
                    this.talla++;
                }
                //Si insertamos en medio de la lista, puesto que estabamos comparando
con el siguiente al nodo actual,
                //habrá que hacer que el siguiente al nuevo nodo sea el auxiliar y el
siguiente al anterior del auxiliar, el nuevo
                else
                {
                    //El siguiente al nuevo será el auxiliar
                    nuevo.siguiente = aux;
                    //El siguiente del anterior al auxiliar, será el nuevo
                    ant.siguiente = nuevo;
                    //Incrementamos en uno la talla
                    this.talla++;
                }
            }
        }
    }

    //Método para borrar un nodo
    public void eliminar(E x)
    {
       //Declaramos un nodo y su anterior, y los inicializamos con el primero y el
ultimo respectivamente
       NodoLEG<E> aux = ultimo.siguiente;
       NodoLEG<E> ant = ultimo;

       do
       {
           //Si encontramos el objeto en la lista
           if(aux.dato == x)
           {
               //Si el objeto es el primero
               if(aux == ultimo.siguiente)
               {
                   //Hacemos que el primero tome el valor de su siguiente
```

```
                    ultimo.siguiente = ultimo.siguiente.siguiente;
                    //Decrementamos la talla en uno
                    this.talla--;
                }
                //Si no si el nodo en el que esta el objeto es el ultimo -> aux == ultimo
                else if(aux == ultimo)
                {
                    //Hacemos que el siguiente al anterior del ultimo sea el primero (lo
que eliminará el ultimo)
                    ant.siguiente = ultimo.siguiente;
                    //Y que el ultimo sea el anterior al que era el ultimo
                    ultimo = ant;
                    this.talla--;
                }
                else
                {
                    //Hacemos que el anterior al que queremos borrar apunte al siguiente
al que queremos borrar
                    //Esto borraría el nodo en el que está el objeto
                    ant.siguiente = aux.siguiente;
                    //Decrementamos la talla en uno
                    this.talla--;
                }

            }
            //Hacemos que el que era anterior al nodo actual sea el actual
            ant = aux;
            //Y pasamos el actual al siguiente
            aux = aux.siguiente;
        }
        while(aux != ultimo.siguiente); //Hasta que se recorra la lista entera
    }


    //Metodo para imprimir la lista que lanza la excepcion de que la lista esté vacia
    public void imprimir() throws EmptyListException
    {
        //Si hay algun nodo en la lista
        if (talla > 0)
        {
            //Declaramos un auxiliar inicializandolo al primero
            NodoLEG<E> aux = ultimo.siguiente;
            //Imprimimos todos los nodos
            do
            {
                System.out.println(aux.dato.toString());
                //Avanzamos un nodo
                aux = aux.siguiente;
            }
            //Mientras que el auxiliar sea distinto del primero
            while(aux != ultimo.siguiente);

        }
        //Si está vacia
        else
        {
            //Lanzamos la excepcion
            throw new EmptyListException("La lista está vacia");
        }

    }

    //Metodo que imprime la lista de manera inversa (esto se usara cuando la lista haya
sido invertida)
    public void imprimirInversa() throws EmptyListException
    {
        //Si hay algun nodo en la lista
        if(this.talla > 0)
        {
            //Recorremos la lista al reves, empezando desde talla - 1 hasta 0
            for(int i = this.talla - 1; i >= 0  ; i--)
            {
                //Imprimimos los datos del nodo
                System.out.println(this.getNodo(i).toString());
            }
        }
        //Si la lista está vacia
```

```
    }

    //Metodo que nos dirá si la lista está vacia
    public boolean vacia() throws EmptyListException
    {
        //Si está vacia, devolvemos true
        if(this.talla == 0)
        {
            throw new EmptyListException("La lista esta vacia");
        }
        //Si no, devolvemos false
        else
        {
            return false;
        }
    }

    //Metodo que invierte la lista devolviendo la misma invertida
    public LEGCircular<E> invertida()
    {
            //Declaramos una lista auxiliar
            LEGCircular<E> aux = new LEGCircular();

            //Recorremos la lista actual del reves desde talla - 1 hasta 0
            for(int i = this.talla - 1; i >= 0; i--)
            {
                //Insertamos los nodos de la lista actual desde el ultimo al primero en
la lista auxiliar
                aux.insertarEnFin(this.getNodo(i));
            }

            //Devolvemos la lista auxiliar
            return aux;
    }
}
```

Clase NodoLEG:

```
package listaCircular;

/**
 *
 * @author Mario
 */
public class NodoLEG<E>
{
    protected E dato;
    protected NodoLEG <E> siguiente;

    public NodoLEG(E dato)
    {
        this.dato = dato;
    }

    public NodoLEG(E dato,NodoLEG <E> siguiente)
    {
        this.dato = dato;
        this.siguiente = siguiente;
    }


}
```

Clase Persona:

```
package listaCircular;

/**
 *
 * @author Mario
 */
public class Persona implements Comparable<Persona>{

    private String dni;
    private String fecha_nacimiento;
    private String nombre;
    private String apellidos;

    public Persona()
    {}

    public Persona(String dni)
    {
        this.dni = dni;
    }

    public Persona(String dni, String fecha_nacimiento)
    {
        this.dni = dni;
        this.fecha nacimiento = fecha nacimiento;
    }

    public Persona(String dni, String fecha_nacimiento, String nombre)
    {
        this.dni = dni;
        this.fecha nacimiento = fecha nacimiento;
        this.nombre = nombre;
    }

      public Persona(String dni, String fecha_nacimiento, String nombre, String
apellidos)
    {
        this.dni = dni;
        this.fecha nacimiento = fecha nacimiento;
        this.nombre = nombre;
        this.apellidos = apellidos;
    }

    public void setDNI(String dni)
    {
        this.dni = dni;
    }

    public String getDNI()
    {
        return this.dni;
    }

    public void setFecha_Nacimiento(String fecha_nacimiento)
    {
        this.fecha nacimiento = fecha nacimiento;
    }

    public String getFecha_Nacimiento()
    {
        return this.fecha nacimiento;
    }

    public void setNombre(String nombre)
    {
        this.nombre = nombre;
    }

    public String getNombre()
```

```
    {
        return this.nombre;
    }
    public void setApellidos(String apellidos)
    {
        this.apellidos = apellidos;
    }

    public String getApellidos()
    {
        return this.apellidos;
    }

    @Override
    public int compareTo(Persona o)
    {

        //Si el DNI de la persona actual es menor que el de la persona pasada por
parámetro
        if(this.getDNI().compareTo(o.getDNI()) < 0)
        {
            //Se devuelve -1
            return -1;
        }
        //Si el DNI de la persona actual es mayor que el de la persona pasada por
parámetro
        else if(this.getDNI().compareTo(o.getDNI()) > 0)
        {
            //Se devuelve 1
            return 1;
        }
        else
        {
            return 0;
        }


    }

    @Override
    public String toString()
    {
        return "DNI: " + this.getDNI() +
                " | Nombre completo: " + this.getNombre() + " " + this.getApellidos() +
                " | Fecha de nacimiento: " + this.getFecha_Nacimiento();
    }
}
```

Clase Practica1_PED:

```
package practica1 ped;

import java.util.Scanner;
import listaCircular.*;
import entrada.*;
import excepciones.*;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 *
 * @author Mario
 */
public class Practica1_PED {

    /**
     * @param args the command line arguments
     */

    private LEGCircular<Persona> lista = null;
    private boolean invertida = false;
```

```
    public void menu() throws NotFoundPersonException, EmptyListException,
NotCreatedException
    {
        int opt = -1;

        do
        {
            try
            {

                System.out.println("\n-------------------------------------------------
-");
                System.out.println("|               MENU LISTA CIRCULAR
|");
                System.out.println("-------------------------|-------------------------
");
                System.out.println("1. Crear lista           | 6. Destruir lista");
                System.out.println("2. Longitud de la lista  | 7. Imprimir lista");
                System.out.println("3. Acceso                | 8. Imprimir
inversamente");
                System.out.println("4. Insertar persona      | 9. Invertir lista");
                System.out.println("5. Suprimir persona      | 0. Salir");
                System.out.println("-------------------------|-------------------------
");
                System.out.print("\n Elija opcion: ");
                opt = MyInput.readInt();
                System.out.print("\n");

                switch(opt)
                {
                        case 1:
                            System.out.println("\nOpcion 1: CREAR LISTA");
                            System.out.println("-----------------------");
                            this.crearLista();
                        break;

                        case 2:
                            System.out.println("Opcion 2: LONGITUD DE LA LISTA");
                            System.out.println("------------------------------");
                            this.longitudLista();
                        break;

                        case 3:

                                System.out.println("Opcion 3: ACCESO");
                                System.out.println("----------------");
                                this.acceso();

                        break;

                        case 4:

                                System.out.println("Opcion 4: INSERTAR PERSONA");
                                System.out.println("--------------------------");
                                this.insertarPersona();


                        break;

                        case 5:

                                System.out.println("Opcion 5: ELIMINAR PERSONA");
                                System.out.println("--------------------------");
                                this.eliminar();


                        break;

                        case 6:

                                System.out.println("Opcion 6: DESTRUIR LISTA");
                                System.out.println("------------------------");
                                this.destruirLista();


                        break;
```

```
                        case 7:

                                System.out.println("Opcion 7: IMPRIMIR LISTA");
                                System.out.println("------------------------");
                                this.imprimir();

                        break;

                        case 8:

                            System.out.println("Opcion 8: IMPRIMIR INVERSAMENTE");
                            System.out.println("-------------------------------");
                            this.imprimirInversa();


                        break;

                        case 9:
                            System.out.println("Opcion 9: INVERTIR LISTA");
                            System.out.println("------------------------");
                            this.invertir();

                        break;
                    }

                }
                catch(NumberFormatException e)
                {
                    System.out.println("Debe introducir un numero");
                }
                catch(EmptyListException e)
                {
                    System.out.println(e.getMessage());
                }
                catch(NotCreatedException e)
                {
                    System.out.println(e.getMessage());
                }
                catch(NotFoundPersonException e)
                {
                    System.out.println(e.getMessage());
                }

            }
            while(opt != 0);

        }

        //Metodo que crea la lista
        public  void crearLista() throws NotCreatedException
        {
            //Si no esta creada
            if(lista == null)
            {
                //Creamos la lista
                this.lista = new LEGCircular<Persona>();
                System.out.println("Lista creada con éxito.");
            }
            //Si esta creada
            else
            {
                //Mostramos mensaje de error
                System.out.println("La lista ya ha sido creada. Para crear una nueva debe
destruir la actual (Opcion 6 en el menú).");
            }
        }

        //Metodo que nos muestra la longitud de la lista
        public void longitudLista() throws NotCreatedException
        {
            //Si esta creada
            if (creada())
            {
                //Mostramos la longitud
                System.out.println("La longitud de la lista es: "+ this.lista.talla());
            }
            //Si no está creada
```

```
        else
        {
            //Mostramos mensaje de alerta
            System.out.println("La lista no ha sido creada. Para crear una nueva acceda
a la opcion 1 del menú.");
        }
    }

    //Metodo para insertar una persona y que lanza la excepcion de que la lista no este
creada
    public void insertarPersona() throws NotCreatedException
    {
        //Si esta creada
        if(creada())
        {
            try
            {
                //Declaracion de variables
                Persona p = null;
                String dni= null, nombre = null, apellidos = null, fecha de nacimiento =
null;
                //Pedimos los datos
                System.out.println("Inserte los siguientes datos: ");
                System.out.print("- DNI: ");
                dni = pedirDNI(); //Pedimos dni de esta manera para validarlo con el
metodo pedirDNI
                System.out.print("- Nombre: ");
                nombre = MyInput.readString();
                System.out.print("- Apellidos: ");
                apellidos = MyInput.readString();
                System.out.print("- Fecha de nacimiento (DD-MM-AAAA):");
                fecha de nacimiento = MyInput.readString();
                p = new Persona(dni,fecha de nacimiento,nombre, apellidos);
                System.out.println("\nLa persona ha sido insertada con exito.");


                //Si la lista está invertida
                if(this.invertida)
                {
                    //Insertamos de manera ordenada invertida
                    this.lista.insertarOrdenadoInvertido(p);
                }
                //Si la lista no está invertida
                else
                {
                    //Insertamos ordenado normal
                    this.lista.insertarOrdenado(p);
                }
                //Capturamos la excepción del formato del dni que lanza pedirDNI()
            } catch (InvalidDniException ex) {
                System.out.println(ex.getMessage());
            }
        }
        //Si la lista no está creada

    }

    //Metodo de acceso que lanza varias excepciones
   public void acceso() throws NotCreatedException, EmptyListException,
NotFoundPersonException
   {

       //Si está creada y no está vacia
       if(creada() && !this.lista.vacia())
       {
           //Pedimos el dni de la persona a mostrar
            System.out.println("Introduzca el DNI de la persona a mostrar: ");
            String dni;
           try
           {
               //Pedimos el dni con pedirDNI para que sea un DNI valido
               dni = pedirDNI();

            //Declaramos una persona a null
            Persona p = null;
            //Buscamos la persona en la lista
            for(int i = 0; i < this.lista.talla(); i++)
```

```
            {
                //Si encontramos esa persona
                if(dni.equals(this.lista.getNodo(i).getDNI()))
                {
                    //Igualamos la persona auxiliar a la persona encontrada
                    p = this.lista.getNodo(i);
                    //Mostramos los datos de la persona
                    System.out.println("Los datos de la persona introducida son: ");
                    System.out.println(p.toString());
                }
            }

            //Si la persona no se encuentra
            if(p == null)
            {   //Lanzamos la excepcion
                throw new NotFoundPersonException("La persona con DNI " + dni + " no se
encuentra en la lista");
            }
            //Capturamos la excepcion de dni invalido lanzada por pedirDNI()
            } catch (InvalidDniException ex) {
                System.out.println(ex.getMessage());
            }
        }

    }

   //Metodo para eliminar una persona
   public void eliminar() throws NotFoundPersonException, EmptyListException,
NotCreatedException
   {
       //Si la lista no está vacia y está creada
       if(creada() && !this.lista.vacia())
       {
           //Pedimos el dni de la persona a borrar
            System.out.println("Introduzca el DNI de la persona a borrar: ");
            String dni;
           try
           {
               //Pedimos el dni con pedirDNI para que sea un DNI valido
               dni = pedirDNI();

            //Declaramos una persona a null
            Persona p = null;
            //Busca en la lista la persona
            for(int i = 0; i < this.lista.talla(); i++)
            {
                //Si encontramos una persona con ese dni
                if(dni.equals(this.lista.getNodo(i).getDNI()))
                {
                    //Igualamos la persona auxiliar a la encontrada
                    p = this.lista.getNodo(i);
                    //Borramos la persona encontrada
                    this.lista.eliminar(p);
                    //Mostramos mensaje
                    System.out.println("La persona con DNI" + p.getDNI() + "ha sido
borrada");
                    //Salimos del bucle
                    break;
                }
            }
            //Si la persona no se encuentra
            if(p == null)
            {
                //Lanzamos la excepcion
                throw new NotFoundPersonException("La persona con DNI" + p.getDNI() +
"no se ha encontrado");
            }
            //Capturamos la excepcion de dni invalido que lanza el metodo pedirDNI()
            } catch (InvalidDniException ex) {
              System.out.println(ex.getMessage());
            }
       }
   }

   //Metodo que destruye la lista
   public void destruirLista() throws NotCreatedException
   {
```

```
        //Si la lista no está creada
        if(creada())
        {
        //Inicializamos la opcion a null
        String opt = null;
        do
        {
            //Recogemos la opcion introducida por el usuario
             System.out.println("¿Está seguro de que desea destruir la lista? (s/n)");
             opt = MyInput.readString();

             //Si la opcion es s o S
             if(opt.equals("s") || opt.equals("S"))
             {
                 //Destruimos la lista haciendo que apunte a null
                 this.lista = null;
                 System.out.println("La lista ha sido destruida");
             }
             //Si la opcion es n
             else if(opt.equals("n") || opt.equals("N"))
             {
                 System.out.println("La lista no ha sido destruida");
             }
        }
        //mientras se inserte una opcion diferente a s, S, n, o N se repite lo anterior
        while(!opt.equals("s") || !opt.equals("s") || !opt.equals("s") ||
!opt.equals("s"));
        }
    }

    //Metodo que nos dirá si la lista está creada o no
    public boolean creada() throws NotCreatedException
    {

        //Si no está creada
        if(this.lista == null)
        {
            //Lanza la excepcion
            throw new NotCreatedException("La lista no está creada. Acceda a la opcion
uno del menu para crarla");
        }
        else
        {
            //Si no devuelve true
            return true;
        }

    }

    //Metodo con el que pedimos el dni para asegurar el formato
    public String pedirDNI() throws InvalidDniException
    {
        //Cogemos el dni mediante el metodo readString de la clase MyInput
        String dni = MyInput.readString();

        //Si el dni tiene un tamaño de 9 digitos
        if(dni.length() == 9)
        {
            //Cogemos el caracter en la posicion 8 del dni
            char letra = dni.charAt(8);

            //Si el caracter es una letra
             if(Character.isLetter(letra))
             {
                 //Devolvemos el dni ya que es correcto
                 return dni;
             }
             else
             {
                 //Si no es una letra, lanzamos la excepcion de que no es valido
                 throw new InvalidDniException("El ultimo caracter debe ser una letra");
             }

         }
        else
        {
            //Si el dni no tiene 9 digitos, lanzamos la excepcion de que no es valido
```

```
            throw new InvalidDniException("El DNI debe tener 9 digitos (XXXXXXXL)");
        }
    }

    //Metodo para imprimir la lista
    public void imprimir() throws NotCreatedException, EmptyListException
     {
     //Si la lista esta creada y no está vacia
     if(creada() && !this.lista.vacia())
     {
             this.lista.imprimir();
     }

     }

    public void imprimirInversa() throws NotCreatedException, EmptyListException
    {
        //Si la lista esta creada y no está vacia
     if(creada() && !this.lista.vacia())
     {
             this.lista.imprimirInversa();
     }
    }

    public void invertir() throws NotCreatedException, EmptyListException
    {
        if(creada() && !this.lista.vacia())
        {
             this.lista = this.lista.invertida();
             System.out.println("La lista ha sido invertida");

             if(this.invertida)
             {
                 this.invertida = false;
             }
             else
             {
                 this.invertida = true;
             }
        }
    }
}
```

Clase Aplicación

```
package practica1 ped;

import excepciones.EmptyListException;
import excepciones.NotCreatedException;
import excepciones.NotFoundPersonException;

/**
 *
 * @author Mario
 */
public class Aplicacion {

     public static void main(String[] args) throws NotFoundPersonException,
EmptyListException, NotCreatedException  {

         Practica1_PED lista_circular = new Practica1_PED();
        lista circular.menu();


    }


}
```