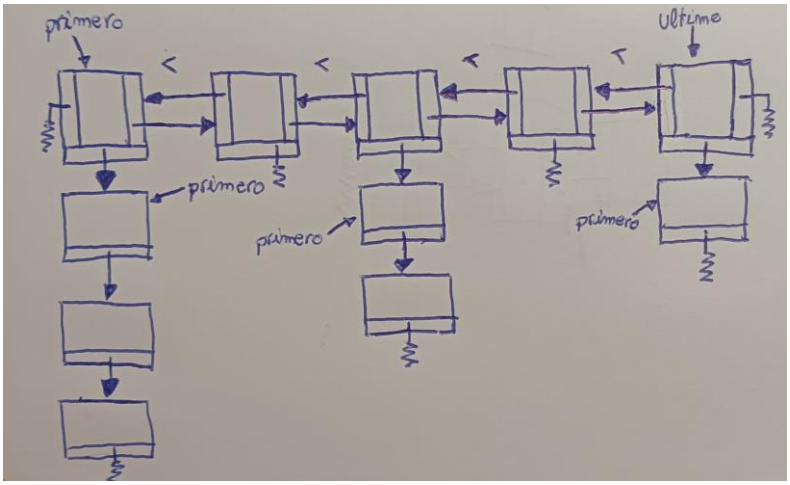
MEMORIA DE LA PRÁCTICA 1: GESTIÓN DE VIAJES DE UNA EMPRESA DE AUTOBUSES 2023



23 ENERO

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 11 de abril de 2023, 22h

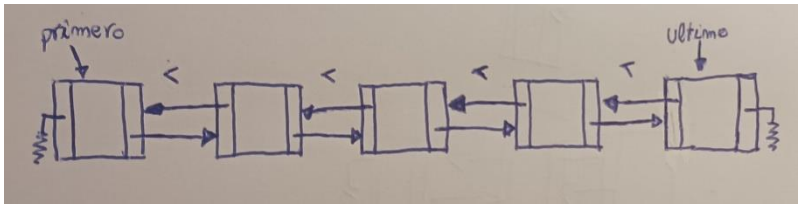
CAMPUS PÚBLICO
MARÍA ZAMBRANO
SEGOVIA

Especificación lógica completa

Especificación lógica completa de los TAD diseñados

Especificación Lógica del TAD LDEG ordenada con último

1.-Definición: el TAD a diseñar es básicamente una lista doblemente enlazada genérica, es decir una lista la cual puedes recorrer de manera ascendente y descendente, ordenada de forma ascendente según la capacidad de los autobuses almacenados en cada nodo de la lista enlazada y con una referencia al último nodo de la lista.



2.-Elementos: como ya he dicho anteriormente los elementos que almacena este TAD son autobuses, con una serie de atributos que los describen.

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

4.-Dominio de los elementos del TAD:
El dominio de los autobuses del TAD es su propio orden dentro de la lista de menor número de plazas a mayor número de plazas no puede haber dos autobuses con la misma matrícula ni dos o más autobuses con un mismo viaje asignado.

5.-Operaciones Básicas:
Nombre: insertar
Descripción: inserta un elemento/dato de manera ordenada en la lista doblemente enlazada.
Datos de entrada: el dato a insertar en la LDEG ordenada con último (principal) y el primer nodo de la lista secundaria (LEG).
Datos de Salida: no devuelve nada.
Precondiciones: no tiene.
Postcondiciones: un nuevo elemento se añade al inicio LDEG ordenada con último. Si la LDEG ordenada estaba vacía antes de la inserción, el nuevo elemento se convierte en la cabeza y cola de la LDEG ordenada con último.

Nombre: buscarPorMatricula
Descripción: busca por matricula en la LDEG ordenada con último para ver si existe un autobús con la misma matrícula, es decir comprueba si una matrícula ya ha sido añadida en otro autobús diferente dentro de la lista.
Datos de entrada: la matrícula que se busca
Datos de Salida: el nodo con el autobús con la matrícula repetida o null
Precondiciones: la lista debe tener al menos un nodo, no debe estar vacía.
Postcondiciones: ninguna.

Nombre: eliminar
Descripción: elimina el dato que se pasa por parámetro de la LDEG ordenada con último.
Datos de entrada: el dato que se quiere eliminar de la LDEG ordenada con último.
Datos de Salida: si no elimino ningún elemento en la LDEG ordenada con último devuelve false y si elimino el dato especificado, devuelve true.
Precondiciones: la LDEG ordenada con último debe contener al menos un elemento.
Postcondiciones: si el elemento x se encuentra en la LDEG ordenada con último, el método lo elimina de la LDEG ordenada con último y devuelve true. Si el elemento x no se encuentra en la LDEG ordenada con último, el método no realiza ninguna acción y devuelve false. Después de la eliminación, la estructura de la LDEG ordenada con último se actualiza adecuadamente para que los elementos restantes estén correctamente enlazados.

Nombre: toString
Descripción: es un método que devuelve una cadena de caracteres que representa los elementos almacenados en la lista doblemente enlazada y ordenada en este caso los autobuses.
Datos de entrada: nada.
Datos de Salida: la cadena que representa los elementos de la lista.
Precondiciones: no tiene.
Postcondiciones: El método devuelve una cadena que representa la LDEG ordenada con último.

Nombre: mostrarInfoAscendente
Descripción: muestra la información de la LDEG ordenada con último de forma ascendente.
Datos de entrada: ninguno.
Datos de Salida: ninguno.
Precondiciones: la lista debe tener al menos un nodo, no debe estar vacía.
Postcondiciones: ninguna.

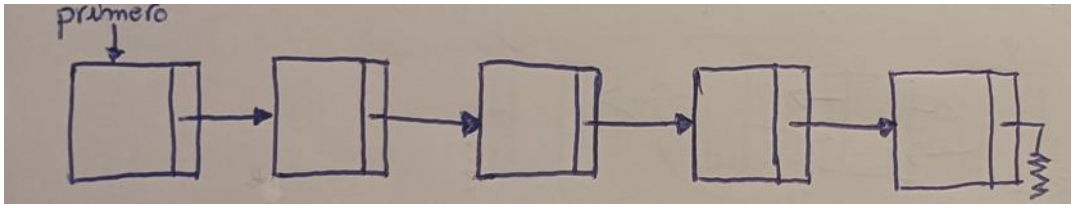
Nombre: mostrarInfoDescendente
Descripción: muestra la información de la LDEG ordenada con último de forma descendente.
Datos de entrada: ninguno.
Datos de Salida: ninguno.
Precondiciones: la lista debe tener al menos un nodo, no debe estar vacía.
Postcondiciones: ninguna.

Nombre: buscarPorCapacidad
Descripción: busca por capacidad (número de plazas) en la LDEG ordenada con último para encontrar un nodo con la misma capacidad y devolverlo y si no devuelve null.
Datos de entrada: la capacidad buscada.
Datos de Salida: el nodo encontrado o null.
Precondiciones: la lista debe tener al menos un nodo, no debe estar vacía.
Postcondiciones: ninguna.

Nombre: mostrarInfoApartirDeValor
Descripción: muestra la información de la LDEG ordenada con último a partir de un valor mínimo introducido por el usuario.
Datos de entrada: la capacidad mínima.
Datos de Salida: ninguno.
Precondiciones: la lista debe tener al menos un nodo, no debe estar vacía.
Postcondiciones: ninguna.

Especificación Lógica del TAD LEG

1.-Definición: el TAD a diseñar es básicamente una lista enlazada genérica.



2.-Elementos: los elementos que almacena este TAD son los viajes de cada autobús, con una serie de atributos que los describen.

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

4.-Dominio de los elementos del TAD:

El dominio de los viajes este TAD son el número de viajes que puede hacer un autobús en un determinado tiempo entre cuatro ciudades, no puede haber dos viajes a la vez en un único autobús y tampoco la ciudad de origen y destino no puede ser la misma.

5.-Operaciones Básicas:

Nombre: insertar

Descripción: inserta un elemento/dato en la lista enlazada genérica.

Datos de entrada: el dato a insertar en la LEG.

Datos de Salida: no devuelve nada.

Precondiciones: no tiene.

Postcondiciones: un nuevo elemento se añade al inicio LEG. Si la LEG estaba vacía antes de la inserción, el nuevo elemento se convierte en la cabeza de la LEG.

Nombre: insertarEnFin

Descripción: inserta un elemento/dato al final de la lista enlazada genérica.

Datos de entrada: el dato a insertar en la LEG.

Datos de Salida: no devuelve nada.

Precondiciones: no tiene.

Postcondiciones: El método agrega el elemento x al final de la LEG. Después de la inserción, la estructura de la LEG se actualiza adecuadamente para que los elementos restantes estén correctamente enlazados. Si la LEG estaba vacía antes de la inserción, el nuevo elemento se convierte en la cabeza de la LEG.

Nombre: toString
Descripción: es un método que devuelve una cadena de caracteres que representa los elementos almacenados en la lista enlazada genérica en este caso los autobuses.
Datos de entrada: nada
Datos de Salida: la cadena que representa los elementos de la lista
Precondiciones: no tiene.
Postcondiciones: El método devuelve una cadena que representa la LEG.

Nombre: eliminar
Descripción: elimina el dato que se pasa por parámetro de la LEG.
Datos de entrada: el dato que se quiere eliminar de la LEG.
Datos de Salida: si no elimino ningún elemento en la LEG devuelve false y si elimino el dato especificado, devuelve true.
Precondiciones: la LEG debe contener al menos un elemento.
Postcondiciones: si el elemento x se encuentra en la LEG, el método lo elimina de la LEG y devuelve true. Si el elemento x no se encuentra en la LEG, el método no realiza ninguna acción y devuelve false. Después de la eliminación, la estructura de la LEG se actualiza adecuadamente para que los elementos restantes estén correctamente enlazados.

Nombre: recuperar
Descripción: obtiene el elemento/dato/objeto que ocupa la i-ésima posición.
Datos de entrada: el número entero de la posición del elemento/dato/objeto
Datos de Salida: devuelve el elemento/dato/objeto que ocupa la posición que representa el entero que se pasa por parámetro.
Precondiciones: La LEG debe contener al menos un elemento. El índice i debe ser un número entero válido que representa la posición del elemento a recuperar en la LEG. El índice i debe estar dentro de los límites de la LEG, es decir, debe ser mayor o igual a 0 y menor que el tamaño de la LEG.
Postcondiciones: Si el índice i está dentro de los límites de la LEG, el método devuelve el elemento que esta en la posición i-ésima de la LEG. Si el índice i está fuera de los límites de la LEG, el método devuelve null. La LEG no se ve modificada por este método.

Breve descripción de los procesos

Breve descripción de los procesos llevados a cabo en los distintos métodos que implementan cada una de las opciones del menú principal

El menú principal tiene dos opciones que son el menú de mantenimiento y el menú de listados. Empezaremos describiendo los procesos llevados a cabo en los métodos del menú de mantenimiento y después haremos lo mismo con el menú de listados.

Menú de Mantenimiento:

1. Método altaAutobus(): este método realiza el alta de un autobus dentro de la aplicación, pidiendo al usuario que introduzca los datos del autobus y controlando que la matrícula del autobus que se este dando de alta en la aplicación no coincida con la matrícula de cualquier autobus que tenga registrado la aplicación.
2. Método bajaAutobus(): este método realiza la baja de un autobus dentro de la aplicación, pidiendo al usuario la matrícula del autobus que se quiere dar de baja, buscando el autobus en la lista de autobuses, si este es encontrado entonces se muestran sus datos y se pregunta si se quiere dar de baja y si no se encuentra se muestra un mensaje de error.
3. Método modificacionAutobus(): este método realiza la modificación de un autobus dentro de la aplicación, pidiendo al usuario la matrícula del autobus que se quiere modificar, buscando el autobus en la lista de autobuses, si este es encontrado entonces se da la opción de modificar el número de plazas y el año de compra del autobus, si no es encontrado se mostrara un mensaje de error.
4. Método altaViaje(): este método realiza el registro de un viaje para un autobus dentro de la aplicación, pidiendo al usuario que introduzca la matrícula del autobus y los datos del viaje, antes de registrarlo comprobamos si el autobus esta dado de alta y comprobamos si no hay un viaje similar registrado y si no lo hay registramos el viaje en la aplicación.

5. Método bajaViaje(): este método realiza la baja de un viaje dentro de la aplicación, pididendo al usuario la matrícula del autobus que realiza el viaje que se quiere borrar, buscando el autobus en la lista de autobuses, si este es encontrado entonces se accede a su lista secundaria de viajes y se busca el viaje que queremos borrar y si se encuentra se muestran sus datos y se pregunta si se quiere dar de baja y si no se encuentra se muestra un mensaje de error.
6. Método modificacionViaje(): este método realiza la modificacióon de un viaje en la aplicación, pidiendo al usuario la matricula del autobus al que pertenece el viaje que quiere modificar, te muestra un listado de los viajes que pertenecen a un autobus y te pide el código del viaje que quieres modificar y te muestra en pantalla sus datos y te pide que modifiques sus datos.

Menu de Listados:

1. Método listadoGeneralAutobusesAscendente(): este método hace un listado de forma ascendente (teniendo en cuenta el número de plazas de los autobuses) de la información de los autobuses registrados en la aplicación.
2. Método listadoGeneralAutobusesDescendente(): este método hace un listado de forma descendente (teniendo en cuenta el número de plazas de los autobuses) de la información de los autobuses registrados en la aplicación.
3. Método listadoGeneralViajes(): te muestra un listado de todos los viajes que hay registrados en la aplicación.
4. Método listadoViajesPartenCiudad(): te pide que introduzcas la ciudad de origen que consideres y te muestra los viajes que parten de esa ciudad
5. Método listadoViajesLleganCiudad(): te pide que introduzcas la ciudad que consideres y te muestra un listado con los viajes que tienen esa ciudad de destino.
6. Método listBusesMayorIgualAlValorIndicado(): este método hace un listado de la información de los autobuses registrados en la aplicación, que tienen una capacidad (número de plazas) mayor o igual a la que haya indicado el usuario.
7. Método mayorViajes(): te muestra un listado con los autobuses o el autobus que realizan mas viajes de toda la aplicación.
8. Método totalViajes(): te muestra la cantidad total de viajes que realizan los autobuses de la empresa
9. Método totalPasajeros(): te muestra la cantidad total de pasajeros que viajan de una ciudad a otra.

Código fuente

Listado de todo el código fuente desarrollado

Clase NodoLDEG:

```
package librerias.estructurasDeDatos.lineales;

import gestion_empresa_autobuses.Viaje;

/**
 *
 * @author Usuario
 * @param <E>
 */
public class NodoLDEG<E> extends LDEGOrdenada{
    protected E dato;
    protected NodoLDEG<E> siguiente, anterior;
    protected LEG<E> listaSecundaria;

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

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

    public E getDato(){
        return this.dato;
    }

    public LEG getListaSecundaria(){
        return this.listaSecundaria;
    }

    public void setListaSecundaria(LEG listaSecundaria){
        this.listaSecundaria = listaSecundaria;
    }

    public boolean existeNodoSecundarioIgual(Viaje nuevoViaje) {
        boolean encontrado = false;
        NodoLEG actual = listaSecundaria.primero;
```

```
        while (actual != null && !encontrado) {
            Viaje v = (Viaje) actual.dato;
            if (v.getDestino().equals(nuevoViaje.getDestino()) &&
v.getOrigen().equals(nuevoViaje.getOrigen()) &&
v.getHora().equals(nuevoViaje.getHora())){
                encontrado = true;
            } else {
                actual = actual.siguiente;
            }
        }
        return encontrado;
    }

}
```

Clase LDEGOrdenadaConUltimo:

```
public class LDEGOrdenada<E extends Comparable<E>> implements Serializable{
    private NodoLDEG<E> primero;
    private NodoLDEG<E> ultimo;

    public LDEGOrdenada() {
        primero = null;
        ultimo = null;
    }

    public void insertar(E x){
        NodoLDEG<E> nuevo = new NodoLDEG<E>(x);

        // Caso base: lista vacía
        if (primero == null) {
            primero = nuevo;
            ultimo = nuevo;
        // Caso concreto con último: elemento a insertar mayor o igual al
último de la lista
        } else if(x.compareTo(ultimo.dato)>=0){
            ultimo.siguiente = nuevo;
            nuevo.anterior = ultimo;
            ultimo = nuevo;
        }else{
            NodoLDEG<E> actual = primero;
            //Caso general: recorremos la lista hasta encontrar un nodo
adecuado
            while (actual != null && x.compareTo(actual.dato) > 0) {
                actual = actual.siguiente;
            }
            //Si no encontramos un nodo adecuado lo insertamos al final de
la lista
            if (actual == null) {
```

```
                ultimo.siguiente = nuevo;
                nuevo.anterior = ultimo;
                ultimo = nuevo;
            //Si lo encontramos insertamos el nuevo nodo antes del nodo
actual que es el adecuado
            } else {
                NodoLDEG<E> anterior = actual.anterior;
                if(anterior != null){
                    anterior.siguiente = nuevo;
                    nuevo.anterior = anterior;
                    nuevo.siguiente = actual;
                    actual.anterior = nuevo;
                }else{
                    nuevo.anterior = null;
                    nuevo.siguiente = actual;
                    actual.anterior = nuevo;
                    primero = nuevo;
                }
            }
        }
    }

    public NodoLDEG buscarPorMatricula(String matriculaBuscada) {
        NodoLDEG nodoActual = primero;

        while (nodoActual != null) {
            Autobus a = (Autobus) nodoActual.dato;

            if (a.getMatricula().equals(matriculaBuscada) ) {
                return nodoActual; // El valor se encuentra en este nodo
            }
            nodoActual = nodoActual.siguiente;
        }

        return null; // El valor no se encuentra en la lista
    }

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

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

    public void mostrarInfoAscendente(){
        NodoLDEG nodoActual = primero;

        while (nodoActual != null) {
            Autobus a = (Autobus) nodoActual.dato;


System.out.println("\t"+a.getMatricula()+"\t\t"+a.getAnioCompra()+"\t\t"+a.g
etNumPlazas());

            nodoActual = nodoActual.siguiente;
        }
    }

    public void mostrarInfoDescendente(){
        NodoLDEG nodoActual = ultimo;

        while (nodoActual != null) {
            Autobus a = (Autobus) nodoActual.dato;


System.out.println("\t"+a.getMatricula()+"\t\t"+a.getAnioCompra()+"\t\t"+a.g
etNumPlazas());

            nodoActual = nodoActual.anterior;
        }
    }

    private NodoLDEG buscarPorCapacidad(int capacidadBuscada){
        NodoLDEG nodoActual = primero;

        while (nodoActual != null) {
            Autobus a = (Autobus) nodoActual.dato;

            if (a.getNumPlazas()==capacidadBuscada) {
                return nodoActual; // El valor se encuentra en este nodo
            }
            nodoActual = nodoActual.siguiente;
        }
        return null; // El valor no se encuentra en la lista
    }
```

```
    public void mostrarInfoApartirDeValor(int capacidadMin){
        NodoLDEG nodoEncontrado = buscarPorCapacidad(capacidadMin);

        while(nodoEncontrado != null){
            Autobus a = (Autobus) nodoEncontrado.dato;


System.out.println("\t"+a.getMatricula()+"\t\t"+a.getAnioCompra()+"\t\t"+a.g
etNumPlazas());

            nodoEncontrado = nodoEncontrado.siguiente;
        }
    }
}
```

Clase Autobus:

```
package gestion_empresa_autobuses;

import java.io.Serializable;


/**
 * Esta clase contiene los métodos y atributos del autobus
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class Autobus implements Comparable<Autobus>, Serializable{
    private String matricula;
    private int anioCompra;
    private int numPlazas;

    public Autobus(String matricula, int anioCompra, int numPlazas){
        this.matricula = matricula;
        this.anioCompra = anioCompra;
        this.numPlazas = numPlazas;
    }

    public String getMatricula(){
        return this.matricula;
    }

    public int getAnioCompra(){
        return this.anioCompra;
    }

    public int getNumPlazas(){
        return this.numPlazas;
    }

    public void setNumPlazas(int nPlazas){
```

```
        this.numPlazas = nPlazas;
    }

    public void setAnioCompra(int aCompra){
        this.anioCompra = aCompra;
    }

    @Override
    public int compareTo(Autobus otroAutobus) {
        return Integer.compare(this.numPlazas, otroAutobus.numPlazas);
    }
}
```

Clase MenuPrincipal:

```
package Menus;

import java.util.ArrayList;
import EntradaSalida.MyInput;
import librerias.estructurasDeDatos.lineales.LDEGOrdenada;

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

    public MenuPrincipal(LDEGOrdenada listaBuses){
        super();
        menus.add(new MenuMantenimiento(listaBuses));
        menus.add(new MenuListados(listaBuses));
    }

    @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 del programa");
```

```
            System.out.println("1. Mantenimiento");
            System.out.println("2. Listados");

        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
                return "n";

    }
}
```

Clase MenuMantenimiento:

```
package Menus;

import EntradaSalida.MyInput;
import gestion_empresa_autobuses.Autobus;
import gestion_empresa_autobuses.Viaje;
import librerias.estructurasDeDatos.lineales.LDEGOrdenada;
import librerias.estructurasDeDatos.lineales.LEG;
import librerias.estructurasDeDatos.lineales.NodoLDEG;
import librerias.estructurasDeDatos.lineales.NodoLEG;

/**
 * Esta clase contiene los métodos y atributos del menú mantenimiento
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class MenuMantenimiento extends Menus {
    private LDEGOrdenada listaBuses;

    public MenuMantenimiento(LDEGOrdenada listaBuses){
        this.listaBuses = listaBuses;
    }

    @Override
    public String ejecutarOpciones() {
        System.out.println("");
        System.out.println("Menú Mantenimiento");
        System.out.println("Seleccione una opción:");
        System.out.println("0. Volver al menú principal");
        System.out.println("");
        System.out.println("1. Alta de autobús");
        System.out.println("2. Baja de autobús");
        System.out.println("3. Modificación de autobús");
        System.out.println("");
        System.out.println("4. Registrar viaje");
        System.out.println("5. Borrar viaje");
        System.out.println("6. Modificar viaje");
        String s=MyInput.readString();
        switch(s){
                case "0": {return "n";}
                case "1": {altaAutobus();return "s";}
                case "2": {bajaAutobus();return "s";}
                case "3": {modificacionAutobus();return "s";}
                case "4": {altaViaje();return "s";}
```

```
                case "5": {bajaViaje();return"s";}
                case "6": {modificacionViaje();return "s";}
                default: {System.out.println("Opción no válida. Vuelva a
intentarlo."); return "s";}
            }
    }

    public void altaAutobus(){
        String respuesta;
        NodoLDEG nodoEncontrado;

        do{
            System.out.println("Alta de Autobuses");
            System.out.println("");
            do{
                System.out.println("Introduzca la matricula del autobus que
quiere dar de alta: ");
                String matricula = MyInput.readString();

                //comprobamos si la matricula esta o no esta en el sistema y
si esta mostramos un mensaje de error y si no esta seguimos con el alta
                nodoEncontrado = listaBuses.buscarPorMatricula(matricula);

                if(nodoEncontrado==null){
                    System.out.println("Introduzca el año de compra del
autobus: ");
                    int anioCompra = MyInput.readInt();

                    System.out.println("Introduzca el número de plazas o
asientos que tiene el autobus: ");
                    int nPlazas = MyInput.readInt();

                    Autobus autobus = new Autobus(matricula, anioCompra,
nPlazas);

                    listaBuses.insertar(autobus);
                }
                else{
                    System.out.println("ERROR: ya existe un autobús
registrado con esa matrícula.");
                    System.out.println("Intentelo de nuevo.");
                }
            }while(nodoEncontrado!=null);
```

```
            System.out.println("¿Desea añadir un nuevo autobus?(S/N):");
            respuesta = MyInput.readString().toUpperCase();
        }while(respuesta.equals("S"));
    }

    public void bajaAutobus(){
        String respuesta;
        NodoLDEG nodoEncontrado;

        do{
            System.out.println("Baja de Autobuses");
            System.out.println("");
            do{
                System.out.println("Introduzca la matricula del autobus que
quiere dar de baja: ");
                String matricula = MyInput.readString();

                nodoEncontrado = listaBuses.buscarPorMatricula(matricula);

                if(nodoEncontrado!=null){
                    Autobus autobus = (Autobus) nodoEncontrado.getDato();

                    System.out.println("DATOS DEL AUTOBUS QUE QUIERE DAR DE
BAJA");
                    System.out.println("");
                    System.out.println("Matrícula:
"+autobus.getMatricula());
                    System.out.println("Año de compra:
"+autobus.getAnioCompra());
                    System.out.println("Número de plazas:
"+autobus.getNumPlazas());

                    System.out.println("¿Desea dar de baja este
autobus?(S/N): ");
                    respuesta = MyInput.readString().toUpperCase();

                    if(respuesta.equals("S")){
                        listaBuses.eliminar(autobus);
                        System.out.println("Baja registrada");
                    }
                    else{
                        System.out.println("Proceso de baja abortado");
                    }
                }
```

```
                else{
                    System.out.println("ERROR: esta matrícula no está
registrada en la aplicación.");
                    System.out.println("Intentelo de nuevo.");
                }
            }while(nodoEncontrado==null);


            System.out.println("¿Desea dar de baja otro autobus?(S/N): ");

            respuesta = MyInput.readString().toUpperCase();
        }while(respuesta.equals("S"));
    }

    public void modificacionAutobus(){
        String respuesta;
        NodoLDEG nodoEncontrado;

        do{
            System.out.println("Modificación de Autobuses");
            System.out.println("");
            do{
                System.out.println("Introduzca la matricula del autobus que
quiere modificar: ");
                String matricula = MyInput.readString();

                nodoEncontrado = listaBuses.buscarPorMatricula(matricula);

                if(nodoEncontrado!=null){
                    Autobus autobus = (Autobus) nodoEncontrado.getDato();

                    System.out.println("Introduzca el número de plazas del
autobus: ");
                    int nPlazas = MyInput.readInt();
                    autobus.setNumPlazas(nPlazas);

                    System.out.println("Introduzca el año de compra del
autobus: ");
                    int aCompra = MyInput.readInt();
                    autobus.setAnioCompra(aCompra);
                }
                else{
                    System.out.println("ERROR: está matrícula no está
registrada en la aplicación");
```

```
                }
            }while(nodoEncontrado==null);

            System.out.println("¿Desea modificar los datos de otro
autobus?(S/N): ");

            respuesta = MyInput.readString().toUpperCase();
        }while(respuesta.equals("S"));
    }

    public void altaViaje(){
        boolean existe=false;
        System.out.println("Registrar viaje");

        System.out.println("Introduzca la matrícula del autobus al que
quiere registrar este viaje:");
        String matricula=MyInput.readString();
        System.out.println("Introduzca el código númerico del viaje:");
        int codigo=MyInput.readInt();
        System.out.println("Introduzca la ciudad de origen del viaje:");
        String origen=MyInput.readString();
        System.out.println("Introduzca la ciudad de destino del viaje:");
        String destino=MyInput.readString();
        System.out.println("Introduzca la hora a la que sale el viaje:");
        String hora=MyInput.readString();

        // Buscar el autobús con la matrícula indicada
        NodoLDEG nodoEncontrado = listaBuses.buscarPorMatricula(matricula);

        if (nodoEncontrado == null) {
            System.out.println("El autobús con matrícula " + matricula + "
no está registrado en la aplicación.");
            return;
        }

        // Crear el nuevo viaje
        Viaje nuevoViaje = new Viaje(codigo, origen, destino, hora);

        if(nodoEncontrado.getListaSecundaria() != null){
            // Comprobar si el autobús ya tiene un viaje con los mismos
datos
            existe = nodoEncontrado.existeNodoSecundarioIgual(nuevoViaje);
        }
```

```
        if(existe){
            System.out.println("Ya existe un viaje similar registrado en
este autobus");
        }
        else{
            // Añadir el nuevo viaje al autobús correspondiente
            if (nodoEncontrado.getListaSecundaria() == null) {
                nodoEncontrado.setListaSecundaria(new LEG());
                nodoEncontrado.getListaSecundaria().insertar(nuevoViaje);
            }
            else{
                nodoEncontrado.getListaSecundaria().insertar(nuevoViaje);
            }

            System.out.println("Viaje registrado correctamente.");
        }
    }

    public void bajaViaje(){
        String respuesta;

        do{
            System.out.println("Baja de Viajes");
            System.out.println("");
            System.out.println("Introduzca la matrícula del autobus que
realiza el viaje que quiere borrar: ");
            String matricula = MyInput.readString();
            System.out.println("Introduzca el codigo del viaje que quiere
dar de baja o borrar: ");
            int codigo = MyInput.readInt();

            NodoLDEG nodoEncontrado =
listaBuses.buscarPorMatricula(matricula);

            if(nodoEncontrado!=null){
                NodoLEG nodoViaje =
nodoEncontrado.getListaSecundaria().buscar(codigo);
                Viaje viaje = (Viaje) nodoViaje.getDato();

                System.out.println("DATOS DEL VIAJE QUE QUIERE DAR DE
BAJA");
                System.out.println("");
                System.out.println("Origen: "+viaje.getOrigen());
                System.out.println("Destino: "+viaje.getDestino());
```

```
                System.out.println("Hora: "+viaje.getHora());

                System.out.println("Desea dar de baja este viaje(S/N): ");
                respuesta = MyInput.readString().toUpperCase();

                if(respuesta.equals("S")){
                    nodoEncontrado.getListaSecundaria().eliminar(viaje);
                    System.out.println("Baja registrada");
                }
                else{
                    System.out.println("Proceso de baja abortado");
                }
            }
            else{
                System.out.println("ERROR: esta viaje no está registrado en
el autobús.");
                System.out.println("Intentelo de nuevo.");
            }

            System.out.println("¿Desea dar de baja otro viaje?(S/N): ");
            respuesta = MyInput.readString().toUpperCase();

        }while(respuesta.equals("S"));
    }

    public void modificacionViaje(){
       String respuesta;
        NodoLDEG nodoEncontrado;
        do{
            System.out.println("Modificación de Viajes");
            System.out.println("");
            do{
                System.out.println("Introduzca la matricula del autobus del
viaje que desea modificar: ");
                String matricula = MyInput.readString();
                nodoEncontrado = listaBuses.buscarPorMatricula(matricula);

                if(nodoEncontrado!=null){
                    Autobus autobus = (Autobus) nodoEncontrado.getDato();
                    System.out.println("LISTADO DE VIAJES DEL AUTOBUS CON
MATRICULA: " + autobus.getMatricula());
                    System.out.printf("%-10s %-10s %-10s %-10s\n", "Codigo",
"Origen", "Destino", "Hora");
```

```
                    System.out.println("-----------------------------------
--------------");
                    LEG listaViajes = nodoEncontrado.getListaSecundaria();
                    listaViajes.mostrarInfoViajes();
                    System.out.println("Introduzca el código del viaje que
desea modificar: ");
                    int codigoViaje = MyInput.readInt();
                    NodoLEG nodoViaje = listaViajes.buscar(codigoViaje);
                    if(nodoViaje!=null){
                        Viaje viaje = (Viaje) nodoViaje.getDato();
                        System.out.println("Los datos del viaje " +
viaje.getCodigo() + " son: ");
                        System.out.println("Origen: " + viaje.getOrigen());
                        System.out.println("Destino: " +
viaje.getDestino());
                        System.out.println("Hora: " + viaje.getHora());
                        System.out.println("Introduzca el nuevo origen: ");
                        String nuevoOrigen = MyInput.readString();
                        viaje.setOrigen(nuevoOrigen);
                        System.out.println("Introduzca el nuevo destino: ");
                        String nuevoDestino = MyInput.readString();
                        viaje.setDestino(nuevoDestino);
                        System.out.println("Introduzca la nueva hora: ");
                        String nuevaHora = MyInput.readString();
                        viaje.setHora(nuevaHora);
                        System.out.println("Autobús " +
autobus.getMatricula() + ": Viaje " + viaje.getCodigo() + " modificado
correctamente");
                    }
                    else{
                        System.out.println("ERROR: este codigo no está
registrado en la aplicación");
                    }
                }
                else{
                    System.out.println("ERROR: está matrícula no está
registrada en la aplicación");
                }
            }while(nodoEncontrado==null);
            System.out.println("¿Desea modificar los datos de otro
viaje?(S/N): ");
            respuesta = MyInput.readString().toUpperCase();
        }while(respuesta.equals("S"));
    }
}
```

Clase MenuListados:

```
package Menus;

import EntradaSalida.MyInput;
import gestion_empresa_autobuses.Autobus;
import gestion_empresa_autobuses.Viaje;
import librerias.estructurasDeDatos.lineales.LDEGOrdenada;
import librerias.estructurasDeDatos.lineales.NodoLDEG;
import librerias.estructurasDeDatos.lineales.LEG;
import librerias.estructurasDeDatos.lineales.NodoLEG;

/**
 * Esta clase contiene los métodos y atributos del menú listados
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class MenuListados extends Menus{
    private LDEGOrdenada listaBuses;


    public MenuListados(LDEGOrdenada listaBuses){
        this.listaBuses = listaBuses;

    }


    @Override
    public String ejecutarOpciones() {
       System.out.println("");
        System.out.println("Menú Listados");
        System.out.println("Seleccione una opción:");
        System.out.println("0. Volver al menú principal");
        System.out.println("");
        System.out.println("1. Listado general de autobuses ordenador por
número de plazas (orden ascendente)");
        System.out.println("2. Listado general de autobuses ordenador por
número de plazas (orden descendente)");
        System.out.println("3. Listado general de viajes");
        System.out.println("4. Listado de viajes que parten de una ciudad
determinada");
        System.out.println("5. Listado de viajes que llegan a una ciudad
determinada");
        System.out.println("6. Listado de autobuses que tienen una capacidad
mayor o igual a la indicada por el usuario");
        System.out.println("7. Listado del autobús o autobuses que realizan
la mayor cantidad de viajes");
        System.out.println("8. Cantidad total de viajes que realizan los
autobuses de la empresa");
        System.out.println("9. Cantidad total de pasajeros que viajan de una
ciudad a otra");
```

```
        String s=MyInput.readString();
        switch(s){
                case "0": {return "n";}
                case "1": {listadoGeneralAutobusesAscendente();return "s";}
                case "2": {listadoGeneralAutobusesDescendente();return "s";}
                case "3": {listadoGeneralViajes();return "s";}
                case "4": {listadoViajesPartenCiudad();return "s";}
                case "5": {listadoViajesLleganCiudad();return"s";}
                case "6": {listBusesMayorIgualAlValorIndicado();return "s";}
                case "7": {mayorViajes();return "s";}
                case "8": {totalViajes();return "s";}
                case "9": {totalPasajeros();return "s";}
                default: {System.out.println("Opción no válida. Vuelva a
intentarlo."); return "s";}
            }
    }

    public void listadoGeneralAutobusesAscendente(){
        System.out.println("LISTADO GENERAL DE AUTOBUSES POR NÚMERO DE
PLAZAS (Orden Ascendente)");
        System.out.println("\tMatrícula\tAño Compra\tPlazas");
        this.listaBuses.mostrarInfoAscendente();
    }

    public void listadoGeneralAutobusesDescendente(){
        System.out.println("LISTADO GENERAL DE AUTOBUSES POR NÚMERO DE
PLAZAS (Orden Descendente)");
        System.out.println("\tMatrícula\tAño Compra\tPlazas");
        this.listaBuses.mostrarInfoDescendente();
    }

    public void listadoGeneralViajes(){
        System.out.println("LISTADO GENERAL DE VIAJES");
        System.out.printf("%-10s %-10s %-10s %-10s %-10s\n", "Bus",
"Codigo", "Origen", "Destino", "Hora");
        System.out.println("----------------------------------------------
--");

        NodoLDEG actualBus = listaBuses.getPrimero();

        while(actualBus != null) {
            Autobus a = (Autobus) actualBus.getDato();
            LEG actualViaje = actualBus.getListaSecundaria();

            if(actualViaje==null){
                actualBus = actualBus.getSiguiente();
            }
            else{
                NodoLEG nodoActual = actualViaje.getPrimero();
```

```
                while(nodoActual != null) {
                    Viaje v = (Viaje) nodoActual.getDato();
                    System.out.printf("%-10s %-10s %-10s %-
10s\n",a.getMatricula(), v.getCodigo(), v.getOrigen(), v.getDestino(),
v.getHora());
                    nodoActual = nodoActual.getSiguiente();
                }

                actualBus = actualBus.getSiguiente();
            }


        }
    }

    public void listBusesMayorIgualAlValorIndicado(){
        System.out.println("LISTADO DE AUTOBUSES QUE TIENEN UNA CAPACIDAD
MAYOR O IGUAL A LA INDICADA POR EL USUARIO");
        System.out.println("Introduzca la capacidad mínima que considere:
");
        int capacidadMin = MyInput.readInt();

        System.out.println("\tMatrícula\tAño Compra\tPlazas");
        this.listaBuses.mostrarInfoApartirDeValor(capacidadMin);
    }

    public void listadoViajesPartenCiudad(){
        System.out.println("LISTADO DE VIAJES QUE PARTEN DE UNA CIUDAD
DETERMINADA");
        System.out.println("Introduzca la ciudad que considere: ");
        String ciudad = MyInput.readString();

        System.out.printf("%-10s %-10s %-10s %-10s\n", "Codigo", "Origen",
"Destino", "Hora");
        System.out.println("-----------------------------------------------
--");

        NodoLDEG actualBus = listaBuses.getPrimero();
        while(actualBus != null){
            LEG actualViaje = actualBus.getListaSecundaria();
            if(actualViaje != null) {
                NodoLEG nodoEncontrado =
actualViaje.buscarPorOrigen(ciudad);
                while(nodoEncontrado != null){
                    Viaje v = (Viaje) nodoEncontrado.getDato();
                    System.out.printf("%-10s %-10s %-10s %-10s\n",
v.getCodigo(), v.getOrigen(), v.getDestino(), v.getHora());
                    nodoEncontrado = nodoEncontrado.getSiguiente();
                }
            }
```

```
            else {
                System.out.println("No se encontraron más viajes que partan
de la ciudad " + ciudad);
            }
            actualBus = actualBus.getSiguiente();
        }

    }

    public void listadoViajesLleganCiudad(){
        System.out.println("LISTADO DE VIAJES QUE LLEGAN DE UNA CIUDAD
DETERMINADA");
        System.out.println("Introduzca la ciudad que considere: ");
        String ciudad = MyInput.readString();

        System.out.printf("%-10s %-10s %-10s %-10s\n", "Codigo", "Origen",
"Destino", "Hora");
        System.out.println("----------------------------------------------
-------------");

        NodoLDEG actualBus = listaBuses.getPrimero();
        while(actualBus != null){
            LEG actualViaje = actualBus.getListaSecundaria();
            if(actualViaje != null) {
                NodoLEG nodoEncontrado =
actualViaje.buscarPorDestino(ciudad);
                while(nodoEncontrado != null){
                    Viaje v = (Viaje) nodoEncontrado.getDato();
                    System.out.printf("%-10s %-10s %-10s %-10s\n",
v.getCodigo(), v.getOrigen(), v.getDestino(), v.getHora());
                    nodoEncontrado = nodoEncontrado.getSiguiente();
                }
            } else {
                System.out.println("No se encontraron más viajes que lleguen
de la ciudad " + ciudad);
            }
            actualBus = actualBus.getSiguiente();
        }

    }

    public void mayorViajes() {
        int maxViajes = 0;

        // Primero encontramos el número máximo de viajes realizados por un
autobús
        for (NodoLDEG autobusActual = listaBuses.getPrimero(); autobusActual
!= null; autobusActual = autobusActual.getSiguiente()) {
            if(autobusActual.getListaSecundaria()!=null){
```

```
                int numViajes =
autobusActual.getListaSecundaria().contarElementos();
                if (numViajes > maxViajes) {
                    maxViajes = numViajes;
                }
            }

        }

        // Luego recorremos la lista de autobuses para imprimir la
información de los autobuses que realizan el número máximo de viajes
        System.out.println("LISTADO DEL AUTOBÚS O AUTOBUSES QUE REALIZAN MÁS
VIAJES");
        System.out.println("----------------------------------------------
--------");
        System.out.println("Autobus/es con mayor cantidad de viajes:");
        for (NodoLDEG autobusActual = listaBuses.getPrimero(); autobusActual
!= null; autobusActual = autobusActual.getSiguiente()) {
            if(autobusActual.getListaSecundaria()!=null){
                Autobus autobus = (Autobus) autobusActual.getDato();
                int numViajes =
autobusActual.getListaSecundaria().contarElementos();
                if (numViajes == maxViajes) {
                    System.out.println("Autobús " + autobus.getMatricula() +
" realiza " + numViajes + " viajes.");
                }
            }


        }
    }

    public void totalViajes() {
        int total = 0;
        NodoLDEG autobusActual = listaBuses.getPrimero();
        while (autobusActual != null) {
            if(autobusActual.getListaSecundaria()!=null){
                NodoLEG viajeActual =
autobusActual.getListaSecundaria().getPrimero();
                while (viajeActual != null) {
                    total++;
                    viajeActual = viajeActual.getSiguiente();
                }

            }
            autobusActual = autobusActual.getSiguiente();
        }
        System.out.println("CANTIDAD TOTAL DE VIAJES QUE REALIZAN LOS
AUTOBUSES DE LA EMPRESA");
```

```
        System.out.println("-----------------------------------------------
-----------------");
        System.out.println("La cantidad total de viajes realizados es: " +
total);
    }

    public void totalPasajeros() {
        int totalPasajeros = 0;

        System.out.println("CANTIDAD TOTAL DE PASAJEROS QUE VIAJAN DE UNA
CIUDAD A OTRA");
        System.out.println("-----------------------------------------------
-----------------");

        System.out.println("Introduzca la ciudad de origen que considere:
");
        String origen = MyInput.readString();

        System.out.println("Introduzca la ciudad de destino que considere:
");
        String destino = MyInput.readString();

        NodoLDEG autobusActual = listaBuses.getPrimero();

        while(autobusActual != null) {
            Autobus autobus = (Autobus) autobusActual.getDato();
            if(autobusActual.getListaSecundaria()!=null){
                NodoLEG viajeActual =
autobusActual.getListaSecundaria().getPrimero();

                while(viajeActual != null) {
                    Viaje v = (Viaje) viajeActual.getDato();

                    if(v.getOrigen().equals(origen) &&
v.getDestino().equals(destino)) {
                        totalPasajeros += autobus.getNumPlazas();
                    }

                    viajeActual = viajeActual.getSiguiente();
                }
            }

            autobusActual = autobusActual.getSiguiente();
        }

        System.out.println("La cantidad total de pasajeros que viajan de " +
origen + " a " + destino + " es de: " + totalPasajeros);
    }
}
```

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 Principal (main) o GestionEmpresaAutobuses:

```
package gestion_empresa_autobuses;

import EntradaSalida.MyInput;
import Menus.MenuPrincipal;
import librerias.estructurasDeDatos.lineales.*;

/**
 * Esta clase es la clase principal del programa la cual lo ejecuta.
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class GestionEmpresaAutobuses {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("Bienvenido a la aplicación GEST_BUS");

        LDEGOrdenada<Autobus> listaBuses= MyInput.deserialize("datos.dat");
        if(listaBuses==null) listaBuses= new LDEGOrdenada<Autobus>();

        MenuPrincipal mp = new MenuPrincipal(listaBuses);
        mp.ejecutar();

        MyInput.serialize(listaBuses, "datos.dat");
    }
```

Clase Viaje:

```
package gestion_empresa_autobuses;

import java.io.Serializable;


/**
 * Esta clase contiene los métodos y atributos del viaje
 * @author Raúl Colindres y Alfredo Sobrados
 */
public class Viaje implements Serializable{
    private int codigo;
    private String origen;
    private String destino;
    private String hora;

    public Viaje(int codigo, String origen, String destino, String hora){
        this.codigo = codigo;
        this.origen = origen;
        this.destino = destino;
        this.hora = hora;
    }

    public int getCodigo(){
        return this.codigo;
    }

    public String getOrigen(){
        return this.origen;
    }

    public String getDestino(){
        return this.destino;
    }

    public String getHora(){
        return this.hora;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
```

```
    public void setOrigen(String origen) {
        this.origen = origen;
    }

    public void setDestino(String destino) {
        this.destino = destino;
    }

    public void setHora(String hora) {
        this.hora = hora;
    }
}
```

Clase NodoLEG:

```
package librerias.estructurasDeDatos.lineales;

/**
 * Esta clase contiene los metodos y atributos de NodoLEG
 * @author Raúl Colindres y Alfredo Sobrados
 * @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 Object getDato() {
        return this.dato;
    }

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

Clase LEG:

```
package librerias.estructurasDeDatos.lineales;

import gestion_empresa_autobuses.Viaje;
import java.io.Serializable;

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

    public LEG(){
       primero=null;
    }

    public NodoLEG<E> getPrimero() {
        return primero;
    }

    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;
    }

    public NodoLEG buscar(int codigo){
        NodoLEG nodoActual = primero;

        while (nodoActual != null) {
            Viaje v = (Viaje) nodoActual.dato;

            if (v.getCodigo()==codigo ) {
                return nodoActual; // El valor se encuentra en este nodo
            }
            nodoActual = nodoActual.siguiente;
        }

       return null; // El valor no se encuentra en la lista
    }


    public NodoLEG buscarPorOrigen(String origen){
        NodoLEG nodoActual = primero;

        while (nodoActual != null) {
            Viaje v = (Viaje) nodoActual.dato;

            if (v.getOrigen().equals(origen)) {
                return nodoActual; // El valor se encuentra en este nodo
            }
            nodoActual = nodoActual.siguiente;
        }

       return null; // El valor no se encuentra en la lista
    }


    public NodoLEG buscarPorDestino(String destino){
        NodoLEG nodoActual = primero;
```

```
        while (nodoActual != null) {
            Viaje v = (Viaje) nodoActual.dato;

            if (v.getDestino().equals(destino)) {
                return nodoActual; // El valor se encuentra en este nodo
            }
            nodoActual = nodoActual.siguiente;
        }

       return null; // El valor no se encuentra en la lista
    }

    public int contarElementos() {
        int contador = 0;
        NodoLEG<E> actual = this.primero;
        while (actual != null) {
            contador++;
            actual = actual.siguiente;
        }
        return contador;
    }

    public void mostrarInfoViajes() {
    NodoLEG actual = primero;

    while (actual != null) {
        Viaje v = (Viaje) actual.getDato();
        System.out.printf("%-10s %-10s %-10s %-10s\n", v.getCodigo(),
v.getOrigen(), v.getDestino(), v.getHora());
        System.out.println("-----------------------------------------------
");

        actual = actual.getSiguiente();
    }
}

}
```

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 ;
}

}
```

Pruebas de ejecución

Pruebas de ejecución (pantallazos de ejecución de la aplicación)

Captura de la bienvenida y el menú principal de la aplicación:

```
Bienvenido a la aplicación GEST_BUS
DeSerializing...

Menú Principal
seleccione una opción:
0. Salir del programa
1. Mantenimiento
2. Listados
```

Captura de la opción 1 del menú principal (menú mantenimiento):

```
Menú Mantenimiento
Seleccione una opción:
0. Volver al menú principal

1. Alta de autobús
2. Baja de autobús
3. Modificación de autobús

4. Registrar viaje
5. Borrar viaje
6. Modificar viaje
```

Captura de la opción 1 del menú mantenimiento (Alta de autobuses):

```
Alta de Autobuses

Introduzca la matricula del autobus que quiere dar de alta:
8456HTE
Introduzca el año de compra del autobus:
2017
Introduzca el número de plazas o asientos que tiene el autobus:
73
¿Desea añadir un nuevo autobus?(S/N):
n
```

Captura de la opción 2 del menú mantenimiento (Baja de autobuses):

```
Baja de Autobuses

Introduzca la matricula del autobus que quiere dar de baja:
8456HTE
DATOS DEL AUTOBUS QUE QUIERE DAR DE BAJA

Matrícula: 8456HTE
Año de compra: 2017
Número de plazas: 73
¿Desea dar de baja este autobus?(S/N):
s
Baja registrada
¿Desea dar de baja otro autobus?(S/N):
n
```

Captura de la opción 3 del menú mantenimiento (Modificación de autobuses):

```
Modificación de Autobuses

Introduzca la matricula del autobus que quiere modificar:
2345KLC
Introduzca el número de plazas del autobus:
36
Introduzca el año de compra del autobus:
2022
¿Desea modificar los datos de otro autobus?(S/N):
n
```

Captura de la opción 4 del menú mantenimiento (Alta de viajes):

```
Registrar viaje
Introduzca la matrícula del autobus al que quiere registrar este viaje:
1093PLB
Introduzca el código númerico del viaje:
0000
Introduzca la ciudad de origen del viaje:
Segovia
Introduzca la ciudad de destino del viaje:
Madrid
Introduzca la hora a la que sale el viaje:
13:00
Viaje registrado correctamente.
```

Captura de la opción 5 del menú mantenimiento (Baja de Viajes):

```
Baja de Viajes

Introduzca la matrícula del autobus que realiza el viaje que quiere borrar:
1093PLB
Introduzca el codigo del viaje que quiere dar de baja o borrar:
0
DATOS DEL VIAJE QUE QUIERE DAR DE BAJA

Origen: Segovia
Destino: Madrid
Hora: 13:00
Desea dar de baja este viaje(S/N):
n
Proceso de baja abortado
¿Desea dar de baja otro viaje?(S/N):
n
```

Captura de la opción 6 del menú mantenimiento (Modificación de Viajes):

```
Modificación de Viajes

Introduzca la matricula del autobus del viaje que desea modificar:
2345KLC
LISTADO DE VIAJES DEL AUTOBUS CON MATRICULA: 2345KLC
Codigo     Origen     Destino    Hora
-------------------------------------------------
4          Barcelona  Madrid     12:00
-------------------------------------------------
Introduzca el código del viaje que desea modificar:
4
Los datos del viaje 4 son:
Origen: Barcelona
Destino: Madrid
Hora: 12:00
Introduzca el nuevo origen:
Sevilla
Introduzca el nuevo destino:
Barcelona
Introduzca la nueva hora:
11:00
Autobús 2345KLC: Viaje 4 modificado correctamente
¿Desea modificar los datos de otro viaje?(S/N):
n
```

Captura de la opción 2 del menú principal (menú listados):

```
Menú Listados
Seleccione una opción:
0. Volver al menú principal

1. Listado general de autobuses ordenador por número de plazas (orden ascendente)
2. Listado general de autobuses ordenador por número de plazas (orden descendente)
3. Listado general de viajes
4. Listado de viajes que parten de una ciudad determinada
5. Listado de viajes que llegan a una ciudad determinada
6. Listado de autobuses que tienen una capacidad mayor o igual a la indicada por el usuario
7. Listado del autobús o autobuses que realizan la mayor cantidad de viajes
8. Cantidad total de viajes que realizan los autobuses de la empresa
9. Cantidad total de pasajeros que viajan de una ciudad a otra
```

Captura de la opción 1 del menú listados (Listado general de autobuses ordenador por número de plazas (orden ascendente)):

```
LISTADO GENERAL DE AUTOBUSES POR NÚMERO DE PLAZAS (Orden Ascendente)
        Matrícula       Año Compra      Plazas
        1093PLB         2015            19
        2345KLC         2020            32
        9754ERD         2019            53
        5423THL         2021            65
        6749FTR         2018            87
        4389TRD         2023            109
```

Captura de la opción 2 del menú listados (Listado general de autobuses ordenador por número de plazas (orden descendente)):

```
LISTADO GENERAL DE AUTOBUSES POR NÚMERO DE PLAZAS (Orden Descendente)
        Matrícula       Año Compra      Plazas
        4389TRD         2023            109
        6749FTR         2018            87
        5423THL         2021            65
        9754ERD         2019            53
        2345KLC         2022            36
        1093PLB         2015            19
```

Captura de la opción 3 del menú listados (Listado general de viajes):

```
LISTADO GENERAL DE VIAJES
Bus         Codigo      Origen      Destino     Hora
-------------------------------------------------------
1093PLB     1           16:00       Segovia     Madrid
1093PLB     0           Segovia     Madrid      13:00
2345KLC     4           Sevilla     Barcelona   11:00
4389TRD     5           Sevilla     Barcelona   19:00
```

Captura de la opción 4 del menú listados (Listado de viajes que parten de una ciudad determinada):

```
LISTADO DE VIAJES QUE PARTEN DE UNA CIUDAD DETERMINADA
Introduzca la ciudad que considere:
Segovia
        Codigo  Origen  Destino Hora
-------------------------------------------------
        2       Segovia Madrid  18:00
        1       Segovia Madrid  16:00
        0       Segovia Madrid  13:00
```

Captura de la opción 5 del menú listados (Listado de viajes que llegan a una ciudad determinada):

```
LISTADO DE VIAJES QUE LLEGAN DE UNA CIUDAD DETERMINADA
Introduzca la ciudad que considere:
Madrid
Codigo      Origen      Destino     Hora
-------------------------------------------------------------
2           Segovia     Madrid      18:00
1           Segovia     Madrid      16:00
0           Segovia     Madrid      13:00
4           Barcelona   Madrid      12:00
```

Captura de la opción 6 del menú listados (Listado de autobuses que tienen una capacidad mayor o igual a la indicada por el usuario):

```
LISTADO DE AUTOBUSES QUE TIENEN UNA CAPACIDAD MAYOR O IGUAL A LA INDICADA POR EL USUARIO
Introduzca la capacidad mínima que considere:
65
        Matrícula       Año Compra      Plazas
        5423THL         2021            65
        6749FTR         2018            87
        4389TRD         2023            109
```

Captura de la opción 7 del menú listados (Listado de autobuses que tienen una capacidad mayor o igual a la indicada por el usuario):

```
LISTADO DEL AUTOBÚS O AUTOBUSES QUE REALIZAN MÁS VIAJES
-------------------------------------------------------
Autobus/es con mayor cantidad de viajes:
Autobús 1093PLB realiza 3 viajes.
```

Captura de la opción 8 del menú listados (Cantidad total de viajes que realizan los autobuses de la empresa):

```
CANTIDAD TOTAL DE VIAJES QUE REALIZAN LOS AUTOBUSES DE LA EMPRESA
-----------------------------------------------------------------
La cantidad total de viajes realizados es: 4
```

Captura de la opción 9 del menú listados (Cantidad total de pasajeros que viajan de una ciudad a otra):

```
CANTIDAD TOTAL DE PASAJEROS QUE VIAJAN DE UNA CIUDAD A OTRA
-----------------------------------------------------------
Introduzca la ciudad de origen que considere:
Segovia
Introduzca la ciudad de destino que considere:
Madrid
La cantidad total de pasajeros que viajan de Segovia a Madrid es de: 57
```