

ENCAPSULAMIENTO

Clases administradoras con listas de objetos, clase LinkedList

CAPACIDAD DE PROCESO:

- Diseña clases administradoras utilizando la clase LinkedList.
- Desarrolla aplicaciones, con interface gráfica de usuarios, con objetos pertenecientes a clases administradoras

Clase Administradora

Una clase administradora es aquella que tiene todo lo necesario para administrar información de cualquier tipo, considerando los atributos y funcionalidad necesarios para ello. Administrar información implica registrar un nuevo elemento, buscarlo, obtenerlo, modificarlo, eliminarlo y saber cuántos elementos tiene.

Lista de objetos

Una lista de objetos es aquella cuyos elementos que la componen son objetos que están vinculados a través de un enlace. A diferencia de los arreglos, una lista no tiene casilleros ni se dimensiona, van creciendo o decreciendo de uno en uno.

Clase LinkedList

La clase **LinkedList** es una clase que administra una lista de objetos de cualquier tipo. Sin embargo, para un uso adecuado se recomienda particularizarlo al objeto que se quiera administrar. Utiliza eficientemente la memoria del computador ajustado estrictamente a la cantidad de objetos que lo contenga.

Dispone de métodos para obtener, remover, insertar objetos al principio, al final, después de algún objeto, antes de algún objeto, etc.

Cuando una lista sólo utiliza **addFirst** para agregar objetos, se le conoce como lista tipo **pila**. Cuando sólo utiliza **addLast** se le conoce como lista tipo **cola** y cuando utiliza indistintamente **addFirst**, **addLast**, **add** se le conoce como lista general.

Métodos de uso frecuente:

Método	Interpretación
<code>void addFirst(Object info)</code>	agrega un nuevo objeto info al inicio de la lista.
<code>void addLast (Object info)</code>	agrega un nuevo objeto info al final de la lista.
<code>void add(int posición, Object info)</code>	agrega un nuevo objeto en la posición entera que se indique.
<code>Object get(int posición)</code>	retorna el objeto de la posición indicada.
<code>Object removeFirst ()</code>	elimina el objeto del inicio de la lista.
<code>Object removeLast ()</code>	elimina el objeto del final de la lista.
<code>Object remove(int posición)</code>	elimina el objeto de la posición indicada.
<code>Object remove(Object info)</code>	elimina el objeto que se indica.
<code>int size()</code>	retorna la cantidad de objetos de la lista.
<code>void clear()</code>	elimina todos los objetos de la lista.
<code>int indexOf(Object info)</code>	retorna la posición del objeto que se indica.

Ejemplo 1:

Diseñe una clase administradora para una lista tipo pila de objetos de tipo Producto considerando la siguiente GUI:



Diseñe la clase **Producto** con constructor explícito y genere los métodos get-set:

```

public class Producto {
    // atributos privados
    private String codigo,
    
```

```
descripcion; private double
precio;
// constructor
public Producto(String codigo, String descripcion, double
    precio){ this.codigo=codigo;
    this.descripcion=descripcion;
    this.precio=precio;
}
// métodos get-set
}
```

Diseñe la clase **ListaPilaProductos**, que tenga como atributo privado un objeto de la clase **LinkedList** donde se guardarán los objetos y un constructor explícito sin parámetros.

```
public class ListaPilaProductos {
    // atributos
    // objeto LinkedList particularizado para objetos de tipo
    Producto
    private LinkedList <Producto> pila ;

    // constructor explícito
    public ListaProductos(){
        pila = new LinkedList<Producto>();
    }
    // métodos de administración
    public int tamaño(){ return pila.size(); }
    public void agrega(Producto p){
        pila.addFirst(p);
    }
    public Producto obtiene(int i){ // retorna un producto de la
    posición i
        return pila.get(i);
    }
    // retorna la posición de un producto según
    su código. public int busca(String codigo){
        for(int i=0; i<tamaño(); i++){
            if(obtiene(i).getCodigo().equals(codigo)
```

```
        )
        return i;
    }
    return -1; // no lo encontró
}

public void elimina(){// elimina el primer producto de la
    pila
    pila.removeFirst();
}

public double mayorPrecio(){// retorna el mayor
    precio double m=pila.get(0).getPrecio();
    for(Producto p: pila){ // for each: por cada producto p
        en pila
        if(p.getPrecio() > m)
            m =p.getPrecio();
    }
    return m;
}

public double menorPrecio(){// retorna el menor
    precio double m=pila.get(0).getPrecio();
    for(Producto p: pila){ // for each: por cada producto p
        en pila
        if(p.getPrecio() < m)
            m =p.getPrecio();
    }
    return m;
}

public double precioPromedio(){// retorna el precio promedio
    double suma=0;
    for(Producto p: pila){ // for each: por cada producto p
        en pil a
        suma += p.getPrecio();
    }
    return suma / tamaño();
}
} // fin de la clase ListaPilaProductos
```

Ahora en la clase de la GUI considere como atributo un objeto de tipo ListaPilaProductos

y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo producto al principio de la lista, evitando que se repita el código
- El botón Busca ubica un producto según el código ingresado y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del producto ubicado con el botón Busca.
- El botón Elimina saca de la lista el primer producto.
- El botón Lista muestra una relación de todos los productos guardados en la lista.
- El botón Reporte muestra los precios mayor, menor y promedio.
- El botón Borrar limpia todo el contenido del GUI.

Ejemplo 2:

Diseñe una clase administradora con soporte para una lista tipo cola de objetos tipo Persona considerando la siguiente GUI:



Diseñe la clase **Persona** con un constructor explícito:

```

public class Persona {
    // atributos privados
    private String dni, nombres,
    apellidos; private int edad;
    private double peso;
    // constructor explícito
    public Persona(String dni, String nombres, String apellidos,
        int edad, double peso){
        this.dni=dni;
        this.nombres=nombres;
        this.apellidos=apellidos;
    }
}
    
```

```
        this.edad=edad;  
        this.peso=peso;  
    }  
    // métodos get-set  
}
```

Diseñe la clase administradora **ListaColaPersonas**, que tenga como atributo privado un objeto de la clase **LinkedList** particularizado para la clase **Persona**, donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- Un método que retorne el peso mayor de todas las personas.
- Un método que retorne el peso menor de todas las personas.
- Un método que retorne el peso promedio de todas las personas.
- Un método que retorne la cantidad de personas cuyo peso se encuentra en un rango dado como parámetros
- Un método que retorne la cantidad de personas mayores de edad.
- Un método que retorne la cantidad de personas menores de edad.

En la clase de la GUI considere como atributo un objeto de tipo **ListaColaPersonas** y programe la acción de los botones de acuerdo a lo siguiente:

- El botón **Nuevo** permite registrar una nueva persona al final de la lista, evitando que se repita el dni
- El botón **Busca** ubica una persona según el dni ingresado y muestra sus datos donde corresponda.
- El botón **Modifica** reemplaza los datos de la persona ubicada con el botón **Busca**.
- El botón **Elimina** saca de la cola la primera persona.
- El botón **Lista** muestra una relación de todas las personas guardadas en la lista.
- El botón **Reporte** muestra el resultado de todos los métodos adicionales.
- El botón **Borrar** limpia todo el contenido del GUI.

Ejemplo 3

Diseñe la clase **Empleado** con los siguientes atributos: código(cadena), nombre(cadena), sueldo(real).

Diseñe la clase **ListaDobleEmpleados** con los siguientes atributos: un objeto lista de tipo **LinkedList**. También considere los siguientes métodos de administración: **agregaAlInicio**, **agregaAlFinal()**, **elimina()**, **busca()**, **obtiene()**, **tamaño()**.

```
public class ListaDobleEmpleados{
    private LinkedList<Empleado> lista;

    public ListaDobleEmpleados(){
        lista = new LinkedList<Empleado>();
    }

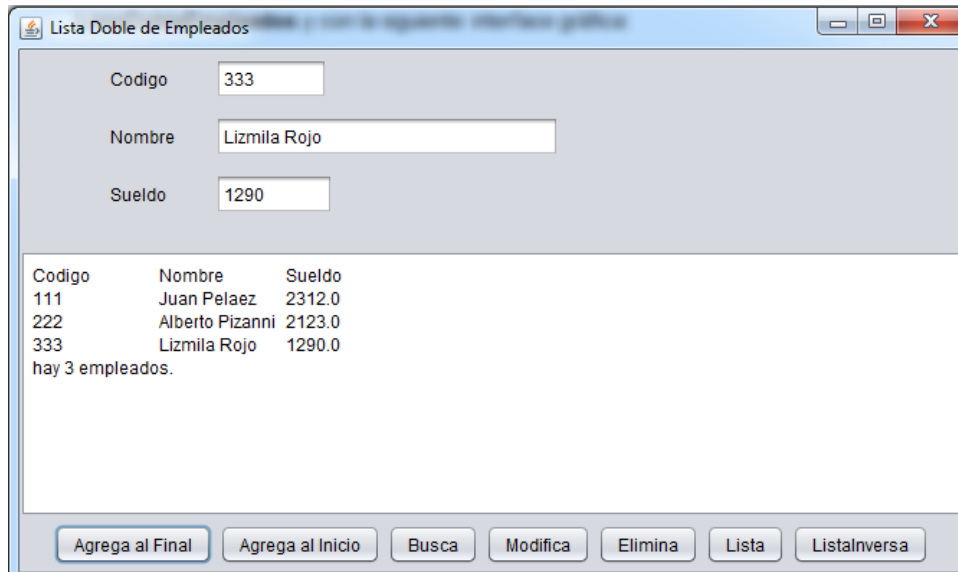
    // métodos de administración
    public int tamaño(){ return lista.size(); }
    public Empleado obtiene(int i){ return lista.get(i); }

    public void agregaAlFinal(Empleado e){
        lista.addLast(e);
    }
    public int busca(String codigo){
        for(Empleado e:lista){
            if(e.getCodigo().equalsIgnoreCase(codigo))
                return lista.indexOf(e);
        }
        return -1;
    }
    public void elimina(int n){
        lista.remove(n);
    }
    public void elimina(Empleado e){
        lista.remove(e);
    }

    public LinkedList<Empleado> getList() {
        return lista;
    }

    public void setLista(LinkedList<Empleado> lista) {
        this.lista = lista;
    }
}
```

Diseñe la clase **PanelEmpleados** con el siguiente atributo: un objeto **Id** de tipo **ListaDobleEmpleados** y con la siguiente interface gráfica:



Codigo	Nombre	Sueldo
111	Juan Pelaez	2312.0
222	Alberto Pizanni	2123.0
333	Lizmila Rojo	1290.0

hay 3 empleados.

```

public class PanelPrincipal extends javax.swing.JPanel {
    protected ListaDobleEmpleados lde;
    /** Creates new form PanelPrincipal */
    public PanelPrincipal() {
        initComponents();
        lde = new ListaDobleEmpleados();
    }
}
    
```

Programe la acción de los botones.

```

private void btnAgregaFinalActionPerformed(java.awt.event.ActionEvent evt) {
    int p = lde.busca(leeCodigo());
    if(p==-1){
        Empleado n = new Empleado(leeCodigo(), leeNombre(), leeSueldo());
        lde.agregaAlFinal(n);
        lista();
    }else{
        mensaje("Codigo repetido!");
    }
}
    
```

```

private void btnEliminaActionPerformed(java.awt.event.ActionEvent evt) {
    int p = lde.busca(leeCodigo());
    if(p==-1)
        mensaje("Codigo no registrado!");
    else{
        lde.elimina(p);
        lista();
    }
}
    
```



```

public void lista() {
    txtSalida.setText("Codigo\tNombre\t\tSueldo\n");
    for(Empleado aux:lde.getList()) {
        imprime(aux.getCodigo()+"\t"+
                aux.getNombre()+"\t\t"+
                aux.getSueldo());
    }
    imprime("Hay "+lde.getNodos()+" empleados.");
}

public void imprime(String s){
    txtSalida.append(s+"\n");
}

public void mensaje(String s){
    JOptionPane.showMessageDialog(this,s);
}

private void btnListaInversaActionPerformed(java....) {
    for (int i=lde.tamaño()-1 ; i>=0; i--){
        Empleado aux = lde.obtiene(i);
        Imprime(aux.getCodigo()+"\t" +
                aux.getNombre()+"\t\t" +
                aux.getSueldo() );
    }
    Imprime("Hay " + lde.tamaño() + " empleados. ");
}

```

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.

Ejemplo 4

Diseñe la clase TV con los siguientes atributos: serie (cadena), marca (entero), tamaño en pulgadas (entero), precio en dólares (real). Considere un constructor explícito y sus métodos get/set. Considere un método adicional que devuelve el precio en soles dado el tipo de cambio como parámetro, y un método adicional que retorne el nombre de la marca. Considere las siguientes marcas: Sony, LG, Samsung, Panasonic, otro.

Diseñela clase administradora **ListaPilaTV**, que tenga como atributo privado un objeto de la clase **LinkedList** particularizado para la clase TV donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne la cantidad de televisores que pertenecen a una marca específica dada como parámetro.
- b) Un método que retorne el precio promedio de los televisores de una marca específica dada como parámetro.
- c) Un método que modifique el precio de todos los televisores incrementándolo en un porcentaje dado como parámetro.

En la clase de la GUI considere como atributo un objeto de tipo ListaPilaTV y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo televisor evitando que se repita la serie.
- El botón Busca ubica un televisor según la serie ingresada y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del televisor ubicado con el botón Busca.
- El botón Elimina saca al primer televisor de la lista.
- El botón Lista muestra una relación de todos los televisores guardados en la lista.
- El botón Reporte muestra el resultado de todos los métodos adicionales.
- El botón AumentaPrecio aplica el resultado del método c) y muestra el resultado del botón Lista.
- El botón Borrar limpia todo el contenido del GUI.

GUIA DE LABORATORIO 3

Clases administradoras de objetos con LinkedList

Ejercicio 1:

Cree un proyecto nuevo de nombre **P03E01**. Cree un paquete nuevo de nombre **controlador** y diseñe una clase administradora para una lista tipo pila de objetos de tipo **Producto** considerando la siguiente GUI:

En el paquete **modelo** diseñe la clase **Producto** con constructor explícito y genere los métodos get-set:

```
public class Producto {
    // atributos privados
    private String codigo,
        descripcion; private double
        precio;
    // constructor
    public Producto(String codigo, String descripcion, double
        precio){ this.codigo=codigo;
        this.descripcion=descripcion;
        this.precio=precio;
    }
    // métodos get-set
}
```

En el paquete controlador diseñe la clase **ListaPilaProductos**, que tenga como atributo privado un objeto de la clase **LinkedList** donde se guardarán los objetos y un

constructor explícito sin parámetros.

```
public class ListaPilaProductos {
    // atributos
    // objeto LinkedList particularizado para objetos de tipo
    Producto
    private LinkedList <Producto> pila ;

    // constructor explícito
    public ListaProductos(){
        pila = new LinkedList<Producto>();
    }
    // métodos de administración
    public int tamaño(){ return pila.size(); }
    public void agrega(Producto p){
        pila.addFirst(p);
    }
    public Producto obtiene(int i){ // retorna un producto de la
    posición i
        return pila.get(i);
    }
    // retorna la posición de un producto según
    su código. public int busca(String codigo){
        for(int i=0; i<tamaño(); i++){
            if(obtiene(i).getCodigo().equals(codigo)
            )
                return i;
        }
        return -1; // no lo encontró
    }
    public void elimina(){// elimina el primer producto de la
    pila
        pila.removeFirst();
    }
    public double mayorPrecio(){// retorna el mayor
        precio double m=pila.get(0).getPrecio();
        for(Producto p: pila){ // for each: por cada producto p
            en pila
            if(p.getPrecio() > m)
```

```
        m =p.getPrecio();
    }
    return m;
}

public double menorPrecio(){// retorna el menor
    precio double m=pila.get(0).getPrecio();
    for(Producto p: pila){ // for each: por cada producto p
        en pila
        if(p.getPrecio() < m)
            m =p.getPrecio();
    }
    return m;
}

public double precioPromedio(){// retorna el precio promedio
    double suma=0;
    for(Producto p: pila){ // for each: por cada producto p
        enpil a
        suma += p.getPrecio();
    }
    return suma / tamaño();
}

} // fin de la clase ListaPilaProductos
```

Cree un paquete nuevo de nombre **vista** y diseñe la GUI en un Panel de nombre **PanelPrincipal** con un atributo de tipo ListaPilaProductos y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo producto al principio de la lista, evitando que se repita el código
- El botón Busca ubica un producto según el código ingresado y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del producto ubicado con el botón Busca.
- El botón Elimina saca de la lista el primer producto.
- El botón Lista muestra una relación de todos los productos guardados en la lista.
- El botón Reporte muestra los precios mayor, menor y promedio.
- El botón Borrar limpia todo el contenido del GUI.

En el paquete vista desarrolle la clase de aplicación de nombre **Principal** en un JFrame. Ejecute su aplicación.

Ejercicio 2:

Cree un proyecto nuevo de nombre **P03E02**. Cree un paquete nuevo de nombre **controlador** y diseñe una clase administradora para una lista tipo cola de objetos tipo Persona considerando la siguiente GUI:



En el paquete **modelo** diseñe la clase **Persona** con un constructor explícito:

```

public class Persona {
    // atributos privados
    private String dni, nombres,
    apellidos; private int edad;
    private double peso;
    // constructor explícito
    public Persona(String dni, String nombres, String apellidos,
    int edad, double peso){
        this.dni=dni;
        this.nombres=nombres;
        this.apellidos=apellidos;
        this.edad=edad;
        this.peso=peso;
    }
    // métodos get-set
}
    
```

En el paquete **controlador** Diseñe la clase administradora **ListaColaPersonas**, que tenga como atributo privado un objeto de la clase **LinkedList** particularizado para la

clase **Persona**, donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne el peso mayor de todas las personas.
- b) Un método que retorne el peso menor de todas las personas.
- c) Un método que retorne el peso promedio de todas las personas.
- d) Un método que retorne la cantidad de personas cuyo peso se encuentra en un rango dado como parámetros
- e) Un método que retorne la cantidad de personas mayores de edad.
- f) Un método que retorne la cantidad de personas menores de edad.

Cree un paquete nuevo de nombre **vista** y diseñe la GUI en un Panel de nombre **PanelPrincipal** con un atributo de tipo `ListaColaPersonas` y programe la acción de los botones de acuerdo a lo siguiente:

- El botón **Nuevo** permite registrar una nueva persona al final de la lista, evitando que se repita el dni
- El botón **Busca** ubica una persona según el dni ingresado y muestra sus datos donde corresponda.
- El botón **Modifica** reemplaza los datos de la persona ubicada con el botón **Busca**.
- El botón **Elimina** saca de la cola la primera persona.
- El botón **Lista** muestra una relación de todas las personas guardadas en la lista.
- El botón **Reporte** muestra el resultado de todos los métodos adicionales.
- El botón **Borrar** limpia todo el contenido del GUI.

En el paquete **vista** desarrolle la clase de aplicación de nombre **Principal** en un `JFrame`. Ejecute su aplicación.

Ejercicio 3

Cree un proyecto nuevo de nombre **P03E03**. Cree un paquete nuevo de nombre **modelo** y diseñe la clase **Empleado** con los siguientes atributos: `código(cadena)`, `nombre(cadena)`, `sueldo(real)`.

Cree un paquete nuevo de nombre **controlador** y diseñe la clase **ListaDobleEmpleados** con los siguientes atributos: un objeto lista de tipo **LinkedList**.

También considere los siguientes métodos de administración: `agregaAlInicio`, `agregaAlFinal`(), `elimina`(), `busca`(), `obtiene`(), `tamaño`().

```
public class ListaDobleEmpleados{
    private LinkedList<Empleado> lista;

    public ListaDobleEmpleados(){
        lista = new LinkedList<Empleado>();
    }

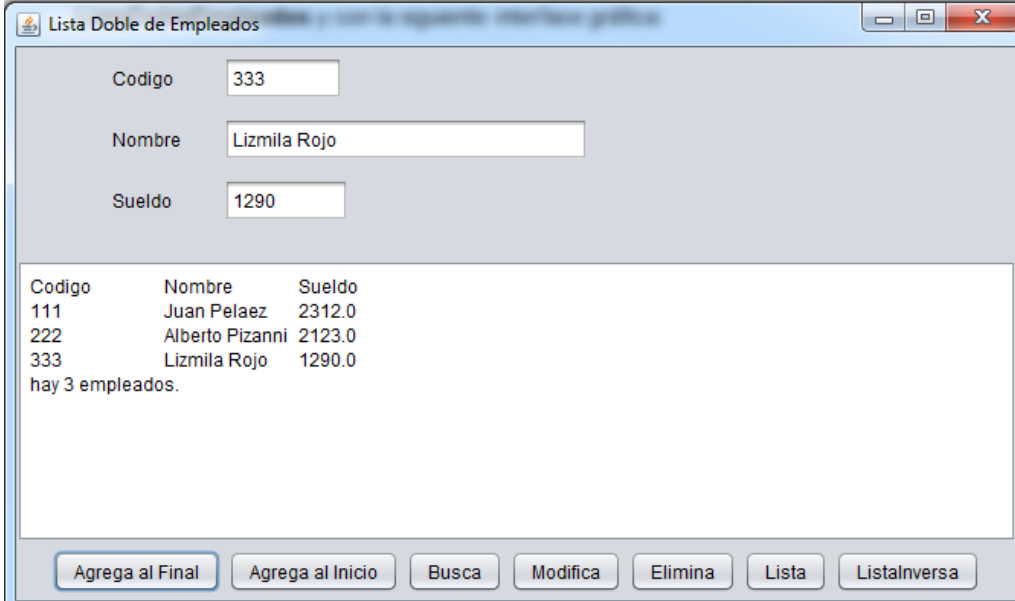
    // métodos de administración
    public int tamaño(){ return lista.size(); }
    public Empleado obtiene(int i){ return lista.get(i);}

    public void agregaAlFinal(Empleado e){
        lista.addLast(e);
    }
    public int busca(String codigo){
        for(Empleado e: lista){
            if(e.getCodigo().equalsIgnoreCase(codigo))
                return lista.indexOf(e);
        }
        return -1;
    }
    public void elimina(int n){
        lista.remove(n);
    }
    public void elimina(Empleado e){
        lista.remove(e);
    }

    public LinkedList<Empleado> getLista() {
        return lista;
    }

    public void setLista(LinkedList<Empleado> lista) {
        this.lista = lista;
    }
}
```

Cree un paquete nuevo de nombre **vista** y diseñe la clase **PanelEmpleados** con el siguiente atributo: un objeto **Idc** de tipo **ListaDobleEmpleados** y con la siguiente interface gráfica:



Codigo	Nombre	Sueldo
111	Juan Pelaez	2312.0
222	Alberto Pizanni	2123.0
333	Lizmila Rojo	1290.0

hay 3 empleados.

```

public class PanelPrincipal extends javax.swing.JPanel {
    protected ListaDobleEmpleados lde;
    /** Creates new form PanelPrincipal */
    public PanelPrincipal() {
        initComponents();
        lde = new ListaDobleEmpleados();
    }
}
    
```

Programe la acción de los botones.

```

private void btnAgregaFinalActionPerformed(java.awt.event.ActionEvent evt) {
    int p = lde.busca(leeCodigo());
    if(p==--1){
        Empleado n = new Empleado(leeCodigo(), leeNombre(), leeSueldo());
        lde.agregaAlFinal(n);
        lista();
    }else
        mensaje("Codigo repetido!");
}
    
```

```

private void btnEliminaActionPerformed(java.awt.event.ActionEvent evt) {
    int p = lde.busca(leeCodigo());
    if(p==--1)
        mensaje("Codigo no registrado!");
    else{
        lde.elimina(p);
        lista();
    }
}
    
```

```

public void lista() {
    txtSalida.setText("Codigo\tNombre\t\tSueldo\n");
    for(Empleado aux:lde.getLista()) {
        imprime(aux.getCodigo()+"\t"+
            aux.getNombre()+"\t\t"+
            aux.getSueldo());
    }
    imprime("Hay "+lde.getNodos()+" empleados.");
}

public void imprime(String s) {
    txtSalida.append(s+"\n");
}

public void mensaje(String s) {
    JOptionPane.showMessageDialog(this,s);
}

private void btnListaInversaActionPerformed(java.awt.event.ActionEvent evt) {
    for (int i=lde.tamaño()-1 ; i>=0; i--) {
        Empleado aux = lde.obtiene(i);
        Imprime(aux.getCodigo()+"\t" +
            aux.getNombre()+"\t\t" +
            aux.getSueldo() );
    }
    Imprime("Hay " + lde.tamaño() + " empleados. ");
}

```

En el paquete **vista** desarrolle la clase de aplicación de nombre **Principal** en un JFrame. Ejecute su aplicación.

Ejercicio 4

Cree un proyecto nuevo de nombre **P03E04**. Cree un paquete nuevo de nombre **modelo** y diseñe la clase TV con los siguientes atributos: serie (cadena), marca (entero), tamaño en pulgadas (entero), precio en dólares (real). Considere un constructor explícito y sus métodos get/set. Considere un método adicional que devuelve el precio en soles dado el tipo de cambio como parámetro, y un método adicional que retorne el nombre de la marca. Considere las siguientes marcas: Sony, LG, Samsung, Panasonic, otro.

Cree un paquete nuevo de nombre **controlador** y diseñe la clase administradora **ListaPilaTV**, que tenga como atributo privado un objeto de la clase **LinkedList** particularizado para la clase TV donde se guardarán los objetos.

Desarrolle los siguientes métodos adicionales en la clase administradora:

- a) Un método que retorne la cantidad de televisores que pertenecen a una marca específica dada como parámetro.
- b) Un método que retorne el precio promedio de los televisores de una marca específica dada como parámetro.
- c) Un método que modifique el precio de todos los televisores incrementándolo en un porcentaje dado como parámetro.

Cree un paquete nuevo de nombre **vistay** diseñe la GUI Principal donde considere como atributo un objeto de tipo ListaPilaTV y programe la acción de los botones de acuerdo a lo siguiente:

- El botón Nuevo permite registrar un nuevo televisor evitando que se repita la serie.
- El botón Busca ubica un televisor según la serie ingresada y muestra sus datos donde corresponda.
- El botón Modifica reemplaza los datos del televisor ubicado con el botón Busca.
- El botón Elimina saca al primer televisor de la lista.
- El botón Lista muestra una relación de todos los televisores guardados en la lista.
- El botón Reporte muestra el resultado de todos los métodos adicionales.
- El botón AumentaPrecio aplica el resultado del método c) y muestra el resultado del botón Lista.
- El botón Borrar limpia todo el contenido del GUI.

Diseñe la clase **Principal** (Frame) y haga funcionar su aplicación.