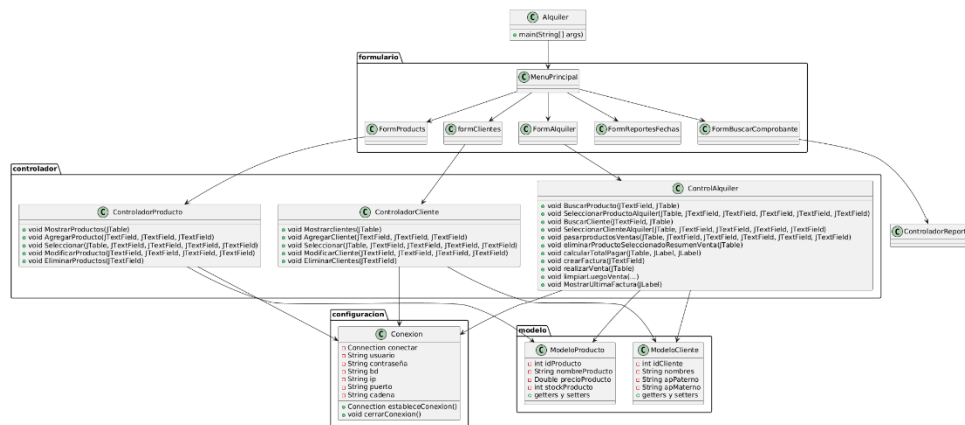


UML ORIGINAL



El Patrón Memento permite guardar y restaurar el estado de un objeto sin violar su encapsulación. Esto es útil cuando queremos implementar la opción de deshacer cambios en una aplicación.

En nuestro caso, lo aplicamos a la gestión de clientes para permitir que, si un usuario modifica un cliente y quiere revertir el cambio, pueda hacerlo sin problemas.

Estructura del Patrón Memento en el código

El patrón Memento se compone de tres partes clave:

1. Memento (ClienteMemento.java) → Guarda un estado del objeto.
2. Originator (ModeloCliente.java) → Crea y restaura estados a partir de Mementos.
3. Caretaker (ClienteAdministrador.java) → Administra el historial de estados.

UML PATRON MEMENTO



CODIGO MODIFICADO PARA LA IMPLEMENTACION DEL PATRON MEMENTO

Clase: ClienteMemento.java.

Esta clase es inmutable y solo almacena información sobre un estado anterior del cliente.

```
package modelo;

/**
 *
 * @author Michel Mendez
 */

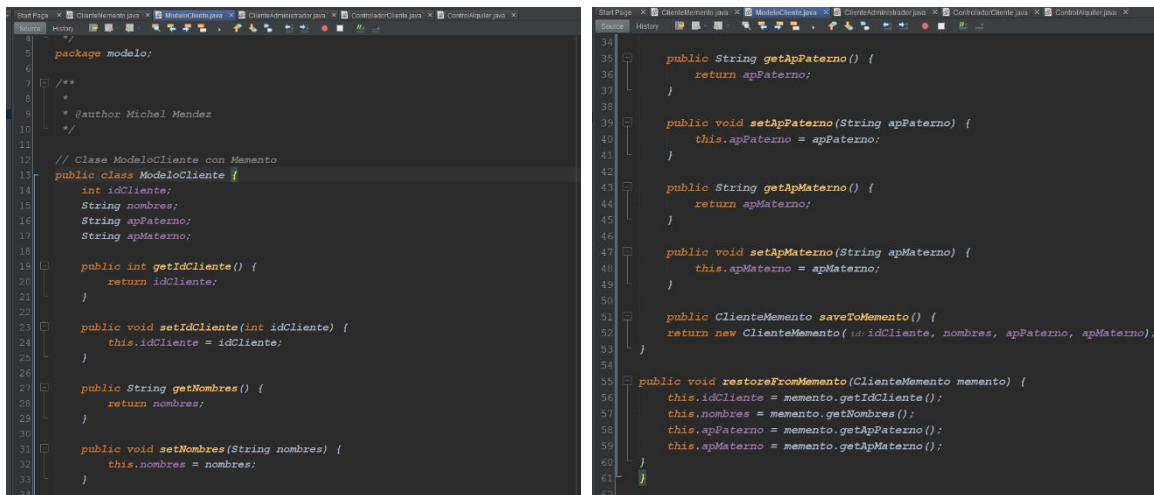
// Clase Memento para guardar el estado de un cliente
public class ClienteMemento {
    private final int idCliente;
    private final String nombres;
    private final String apPaterno;
    private final String apMaterno;

    public ClienteMemento(int id, String nombres, String apPaterno, String apMaterno) {
        this.idCliente = id;
        this.nombres = nombres;
        this.apPaterno = apPaterno;
        this.apMaterno = apMaterno;
    }

    public int getIdCliente() { return idCliente; }
    public String getNombres() { return nombres; }
    public String getApPaterno() { return apPaterno; }
    public String getApMaterno() { return apMaterno; }
}
```

Esta clase solo almacena datos, no tiene lógica de negocio. Es inmutable ya que no se puede modificar un memento una vez creado.

Clase: ModeloCliente.java (El Originator)



```
package modelo;

/**
 *
 * @author Michel Mendez
 */
// Clase ModeloCliente con Memento
public class ModeloCliente {
    int idCliente;
    String nombres;
    String apPaterno;
    String apMaterno;

    public int getIdCliente() {
        return idCliente;
    }

    public void setIdCliente(int idCliente) {
        this.idCliente = idCliente;
    }

    public String getNombres() {
        return nombres;
    }

    public void setNombres(String nombres) {
        this.nombres = nombres;
    }

    public String getApPaterno() {
        return apPaterno;
    }

    public void setApPaterno(String apPaterno) {
        this.apPaterno = apPaterno;
    }

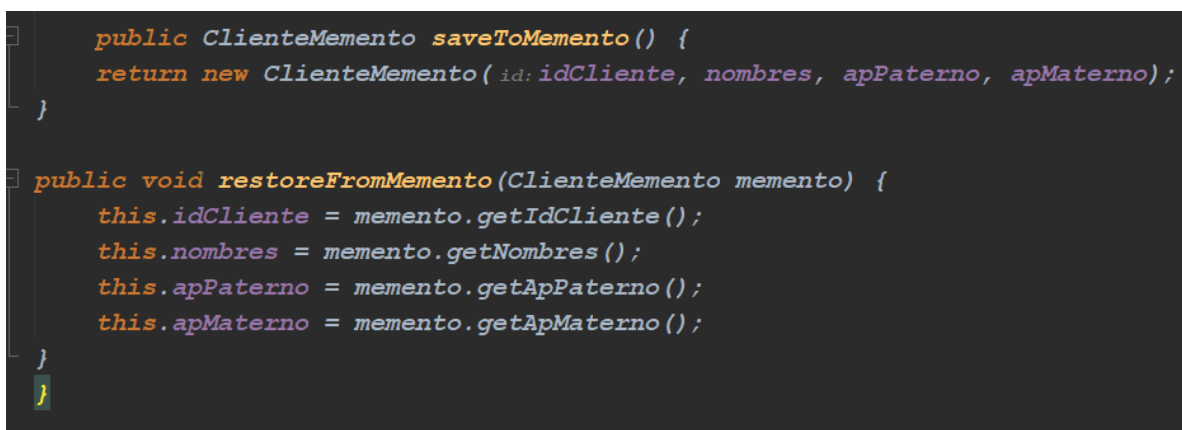
    public String getApMaterno() {
        return apMaterno;
    }

    public void setApMaterno(String apMaterno) {
        this.apMaterno = apMaterno;
    }

    public ClienteMemento saveToMemento() {
        return new ClienteMemento( id: idCliente, nombres, apPaterno, apMaterno);
    }

    public void restoreFromMemento(ClienteMemento memento) {
        this.idCliente = memento.getIdCliente();
        this.nombres = memento.getNombres();
        this.apPaterno = memento.getApPaterno();
        this.apMaterno = memento.getApMaterno();
    }
}
```

El Originator es el objeto principal que queremos guardar y restaurar. Aquí, se agregaron dos métodos nuevos:



```
public ClienteMemento saveToMemento() {
    return new ClienteMemento( id: idCliente, nombres, apPaterno, apMaterno);
}

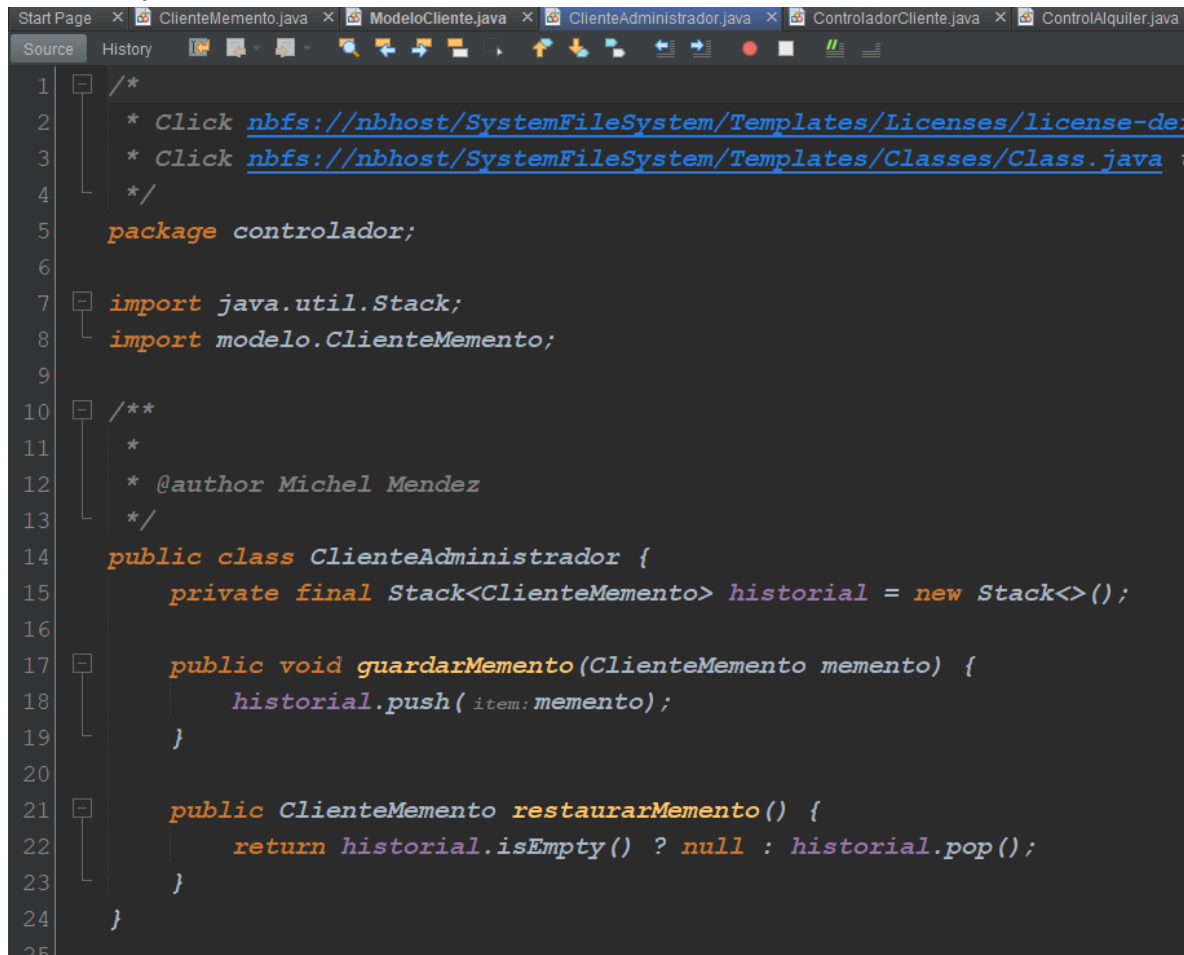
public void restoreFromMemento(ClienteMemento memento) {
    this.idCliente = memento.getIdCliente();
    this.nombres = memento.getNombres();
    this.apPaterno = memento.getApPaterno();
    this.apMaterno = memento.getApMaterno();
}
}
```

saveToMemento() crea un Memento con el estado actual.

restoreFromMemento() revierte los cambios y restaura el estado anterior.

Clase: ClienteAdministrador.java

Esta clase actúa como un historial de cambios, guardando y recuperando estados previos del cliente.



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-def
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
4   */
5   package controlador;
6
7   import java.util.Stack;
8   import modelo.ClienteMemento;
9
10  /**
11   *
12   * @author Michel Mendez
13   */
14  public class ClienteAdministrador {
15      private final Stack<ClienteMemento> historial = new Stack<>();
16
17      public void guardarMemento(ClienteMemento memento) {
18          historial.push(item: memento);
19      }
20
21      public ClienteMemento restaurarMemento() {
22          return historial.isEmpty() ? null : historial.pop();
23      }
24  }
```

Funciona como una pila de deshacer (undo): El último estado guardado es el primero en recuperarse.

guardarMemento() almacena un estado antes de cambiarlo.

restaurarMemento() revierte al estado anterior.

Clase: ControladorCliente.java (Usando el Patrón Memento)

Modificamos ControladorCliente para que, antes de modificar un cliente, guarde el estado anterior en un **memento**.

```

131
132 public void ModificarCliente(JTextField id, JTextField nombres, JTextField appaterno, JTextField apmaterno) {
133     configuracion.Conexion objetoConexion = new configuracion.Conexion();
134
135     String consulta = "UPDATE cliente SET nombres=?, appaterno=?, apmaterno=? WHERE idcliente=?";
136
137     try {
138         // Guardamos el estado antes de modificar
139         caretaker.guardarMemento(memento; cliente.saveToMemento());
140
141         cliente.setIdCliente(Integer.parseInt(id.getText()));
142         cliente.setNombres(nombres.getText());
143         cliente.setApPaterno(appaterno.getText());
144         cliente.setApMaterno(apmaterno.getText());
145
146         CallableStatement cs = objetoConexion.estableceConexion().prepareCall("sql: consulta");
147         cs.setString(parameterIndex: 1, x: cliente.getNombres());
148         cs.setString(parameterIndex: 2, x: cliente.getAppaterno());
149         cs.setString(parameterIndex: 3, x: cliente.getApMaterno());
150         cs.setInt(parameterIndex: 4, x: cliente.getIdCliente());
151
152         cs.execute();
153         JOptionPane.showMessageDialog(parentComponent: null, message: "Se modificó correctamente.");
154
155     } catch (Exception e) {
156         JOptionPane.showMessageDialog(parentComponent: null, "Error al modificar: " + e.toString());
157     } finally {
158         objetoConexion.cerrarConexion();
159     }
160 }

```

Antes de modificar, llamamos a guardarMemento() para no perder el estado anterior.

Método para revertir cambios

Si el usuario quiere deshacer la última modificación, usamos:

```

public void deshacerModificacion() {
    ClienteMemento memento = caretaker.restaurarMemento();
    if (memento != null) {
        cliente.restoreFromMemento(memento);
        JOptionPane.showMessageDialog(parentComponent: null, message: "Modificación revertida.");
    } else {
        JOptionPane.showMessageDialog(parentComponent: null, message: "No hay cambios para deshacer.");
    }
}

```

Recuperamos el último estado guardado en ClienteCaretaker.

Restauramos el estado del cliente usando restoreFromMemento().

Clase: formClientes.java

```

private void btndeshacerclientesActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    controlador.ControladorCliente objetoCliente = new controlador.ControladorCliente();
    objetoCliente.deshacerModificacion();
}

```

Esto llamará al método deshacerModificacion() del ControladorCliente.

