

PATRON METHOD FACTORY

El Patrón de Fábrica nos permite crear objetos sin exponer la lógica de instanciación directamente en el código. En tu caso, lo aplicamos a los controladores para evitar el uso de `new ControladorX()` en múltiples lugares.

```
1 package controlador;
2
3 /**
4  *
5  * @author Samuel
6  */
7 import Factory.IControlador;
8 import java.sql.CallableStatement;
9 import java.sql.ResultSet;
10 import java.sql.Statement;
11 import javax.swing.JOptionPane;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14 import javax.swing.table.DefaultTableModel;
15
16
17
18
19
20
21 public class ControladorProducto implements IControlador {
22
23     @Override
24     public void inicializar() {
25         System.out.println("Controlador de Clientes Inicializado");
26     }
27 }
```

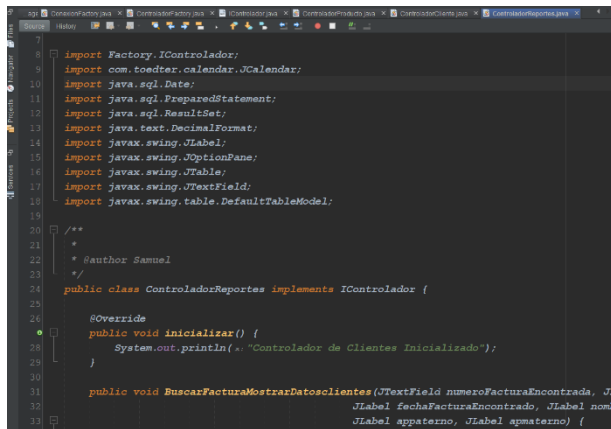
Cada controlador ahora implementa `IControlador`, lo que obliga a definir el método `inicializar()` en todos.

- Implementamos `IControlador`, asegurando que este controlador se pueda usar en la fábrica.
- Definimos el método `inicializar()`, que muestra un mensaje cuando se crea la instancia.
- Se mantiene el método `MostrarProductos()`, que contiene la lógica original.

Lo mismo se aplicó a `ControlAlquiler`, `ControladorCliente` y `ControladorReportes`.

```
1 /**
2  * Click ahí://nbhost/SystemFileSystem/Templates/Licenses/default.txt to change this template file
3  * Click ahí://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template file
4  */
5 package controlador;
6
7 import Factory.IControlador;
8 import configuration.Conexion;
9 import modelo.ModeloCliente;
10
11 import java.sql.CallableStatement;
12 import java.sql.ResultSet;
13 import java.sql.Statement;
14 import javax.swing.JOptionPane;
15 import javax.swing.JTable;
16 import javax.swing.JTextField;
17 import javax.swing.table.DefaultTableModel;
18
19 public class ControladorCliente implements IControlador {
20
21     @Override
22     public void inicializar() {
23         System.out.println("Controlador de Clientes Inicializado");
24     }
25 }
```

```
1 /**
2  *
3  * @author Samuel
4  */
5 package controlador;
6
7 import Factory.IControlador;
8 import java.sql.CallableStatement;
9 import java.sql.ResultSet;
10 import java.sql.Statement;
11 import javax.swing.JOptionPane;
12 import javax.swing.JTable;
13 import javax.swing.JTextField;
14 import javax.swing.table.DefaultTableModel;
15
16
17
18
19
20
21 public class ControladorProducto implements IControlador {
22
23     @Override
24     public void inicializar() {
25         System.out.println("Controlador de Clientes Inicializado");
26     }
27
28     public void MostrarProductos(JTable tablaTotalProductos){
29
30     }
31 }
```



```

7
8 import Factory.IControlador;
9 import com.toedter.calendar.JCalendar;
10 import java.sql.Date;
11 import java.sql.PreparedStatement;
12 import java.sql.ResultSet;
13 import java.text.DecimalFormat;
14 import javax.swing.JLabel;
15 import javax.swing.JOptionPane;
16 import javax.swing.JTable;
17 import javax.swing.JTextField;
18 import javax.swing.table.DefaultTableModel;
19
20 /**
21  *
22  * @author Samuel
23  */
24 public class ControladorReportes implements IControlador {
25
26     @Override
27     public void inicializar() {
28         System.out.println("Controlador de Clientes Inicializado");
29     }
30
31     public void BuscarFacturaMostrarDatosclientes(JTextField numeroFacturaEncontrada, J
32                                                  JLabel fechaFacturaEncontrada, JLabel nomb
33                                                  JLabel appaterno, JLabel apmaterno) {

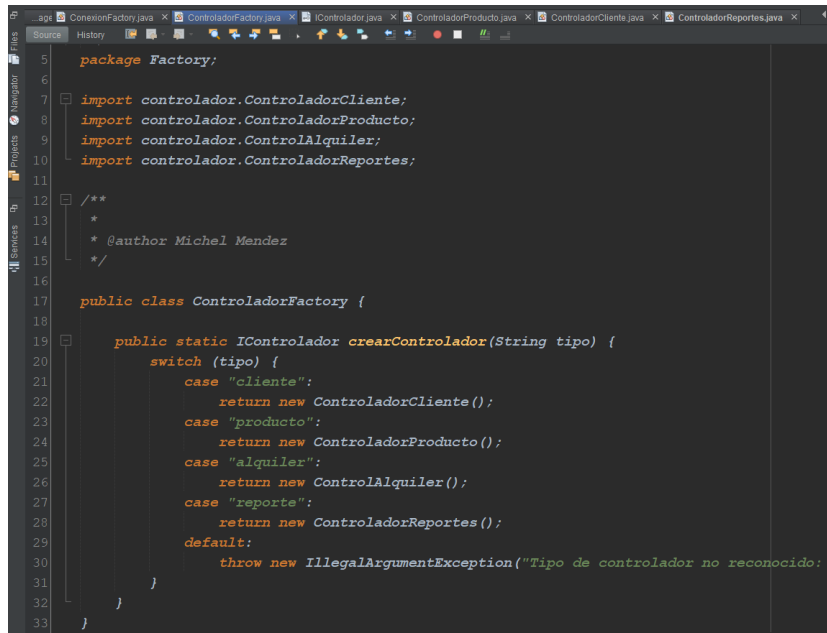
```

Ahora que todos los controladores implementan IControlador, la fábrica puede crearlos sin problemas.

Estos controladores no implementaban la interfaz IControlador. La fábrica debe devolver objetos del mismo tipo (IControlador), pero si un controlador no implementa esta interfaz, el compilador no lo reconoce como válido.

Hicimos que todos los controladores implementen IControlador, lo que permite que la fábrica los maneje de manera uniforme.

Implementación del ControladorFactory



```
5 package Factory;
6
7 import controlador.ControladorCliente;
8 import controlador.ControladorProducto;
9 import controlador.ControlAlquiler;
10 import controlador.ControladorReportes;
11
12 /**
13  *
14  * @author Michel Mendez
15  */
16
17 public class ControladorFactory {
18
19     public static IControlador crearControlador(String tipo) {
20         switch (tipo) {
21             case "cliente":
22                 return new ControladorCliente();
23             case "producto":
24                 return new ControladorProducto();
25             case "alquiler":
26                 return new ControlAlquiler();
27             case "reporte":
28                 return new ControladorReportes();
29             default:
30                 throw new IllegalArgumentException("Tipo de controlador no reconocido: " + tipo);
31         }
32     }
33 }
```

- Se recibe un String tipo que indica qué controlador crear.
- Usamos switch-case para verificar el tipo y devolver la instancia correcta.
- Si el tipo no es válido, se lanza un error con `IllegalArgumentException`.
- Se usa `toLowerCase()` para evitar errores si el usuario pasa "CLIENTE" en mayúsculas.

Con esto, ahora podemos instanciar cualquier controlador sin usar `new` directamente.

Implementación en FormClientes

```
Start Page  x ConectorFactory.java  x ControladorFactory.java  x IControlador.java  x ControladorProducto.java  x ControladorCliente.java  x ControladorReportes.java  x formClientes.java
Source  Design  History  nbfs://nbhost/SystemFileSystem/Templates/GuiForms/JFrame.java to edit this template
3 4
5 package formulario;
6
7 import Factory.ControladorFactory;
8 import controlador.ControladorCliente;
9 import Factory.IControlador;
10
11 /**
12  *
13  * @author Michel Mendez
14  */
15 public class formClientes extends javax.swing.JFrame {
16
17     private IControlador objetoCliente;
18
19     /**
20      * Creates new form formClientes
21      */
22     public formClientes() {
23         initComponents();
24
25         // Usamos la fábrica para obtener el controlador
26         objetoCliente = ControladorFactory.crearControlador( tipo: "cliente");
27
28         // Llenamos la tabla con los clientes
29         ((ControladorCliente) objetoCliente).MostrarClientes( tablaTotalClientes: tbclientes);
30     }
31
32     // Deshabilitamos el campo de ID
33     idcliente.setEnabled( enabled: false);
34
35     /**
36      * This method is called from within the constructor to initialize the form.
37      * WARNING: Do NOT modify this code. The content of this method is always
38      * regenerated by the Form Editor.
39      */
40     @SuppressWarnings("unchecked")
41     Generated Code
225
226     private void btnguardarclientesActionPerformed(java.awt.event.ActionEvent evt) {
227         ((ControladorCliente) objetoCliente).AgregarCliente( nombre: txtnombrecliente, apmaterno: txtapmaterno, apmaterno: txtapmaterno);
228         ((ControladorCliente) objetoCliente).MostrarClientes( tablaTotalClientes: tbclientes);
229     }
230
231     private void btnmodificarclientesActionPerformed(java.awt.event.ActionEvent evt) {
232         ((ControladorCliente) objetoCliente).ModificarCliente( id: idcliente, nombre: txtnombrecliente, apmaterno: txtapmaterno, apmaterno: txtapmaterno);
233         ((ControladorCliente) objetoCliente).MostrarClientes( tablaTotalClientes: tbclientes);
234     }
235
236     private void btneliminarclientesActionPerformed(java.awt.event.ActionEvent evt) {
237         ((ControladorCliente) objetoCliente).EliminarCliente( id: idcliente);
238         ((ControladorCliente) objetoCliente).MostrarClientes( tablaTotalClientes: tbclientes);
239     }
240
241
242     private void tbclientesMouseClicked(java.awt.event.MouseEvent evt) {
243         controlador.ControladorCliente objetoCliente = new ControladorCliente();
244         objetoCliente.Seleccionar( totalcliente: tbclientes, id: idcliente, nombre: txtnombrecliente, apmaterno: txtapmaterno, apmaterno: txtapmaterno);
245     }
246
247     private void idclienteActionPerformed(java.awt.event.ActionEvent evt) {
248         // TODO add your handling code here:
249     }
250
251     private void txtapmaternoActionPerformed(java.awt.event.ActionEvent evt) {
252         // TODO add your handling code here:
253     }
254
255     private void btnlimpiarcamposActionPerformed(java.awt.event.ActionEvent evt) {
256         ((ControladorCliente) objetoCliente).limpiarCampos( id: idcliente, nombre: txtnombrecliente, apmaterno: txtapmaterno, apmaterno: txtapmaterno);
257     }
258
259     private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
260         // TODO add your handling code here:
261     }
262
263 }
```

En lugar de hacer esto:

```
ControladorCliente objetoCliente = new ControladorCliente();
```

Ahora usamos la fábrica:

```
IControlador objetoCliente =  
ControladorFactory.crearControlador("cliente");  
  
((ControladorCliente) objetoCliente).Mostrarclientes(tbclientes);
```

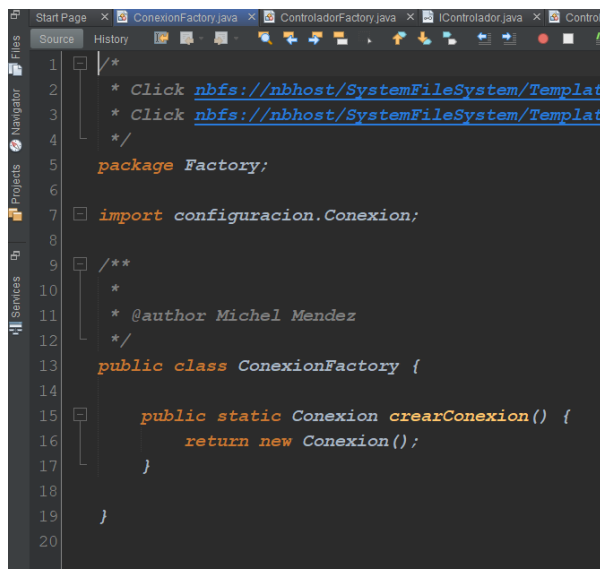
Implementacion de ConexionFactory

Al igual que con los controladores, aplicaremos el Patrón de Fábrica para la conexión a la base de datos.

Cada vez que se necesitábamos una conexión, hacíamos esto:

```
configuracion.Conexion objetoConexion = new configuracion.Conexion();
```

El problema era que cada controlador crea su propia instancia de Conexion, lo que duplica código y dificulta el mantenimiento. Si en el futuro queremos cambiar el tipo de conexión, habría que modificar todo el código.



```
1  /**  
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Comment  
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Comment  
4   */  
5   package Factory;  
6  
7   import configuracion.Conexion;  
8  
9   /**  
10    *  
11    * @author Michel Mendez  
12    */  
13    public class ConexionFactory {  
14  
15        public static Conexion crearConexion() {  
16            return new Conexion();  
17        }  
18  
19    }  
20
```

Solución:

- Crearemos una fábrica de conexiones para manejar la lógica de conexión en un solo lugar.
- Así, si se necesita cambiar la configuración de la base de datos, solo se modifica ConexionFactory.

