EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Oaxaca

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

CARRERA: INGENIERÍA EN SISTEMAS COMPUTACIONALES

Diseño E Implementacion De Software Con Patrones

Profesora: Espinosa Pérez Jacob

P R E S E N T A:

Mendez Mendoza Luisa Michel

Girón Pacheco Fernando

Mario Alberto Barbosa Santiago

Lourdes Gloria López García

Samuel Pérez Carrasco

PROTOTYPE

Es un patrón de diseño creacional, que tiene como objetivo crear a partir de un modelo. Especifique el tipo de objetos que se crearán mediante una instancia prototípica, y cree nuevos objetos copiando este prototipo.

Tiene gran simplicidad ya que su concepto es de hacer una copia exacta de otro objeto, esto en lugar de crear uno nuevo, permite crear los objetos prediseñados sin la necesidad de conocer los detalles de la creación.

- **CLONACIÓN PROFUNDA**

Esta clonación se realiza cuando el objeto que se quiere clonar tiene como atributos otros objetos (1 o más) que se deben clonar de igual manera, para retornar una clonación completa del objeto y no referencias a otros ya existentes.

- **CLONACIÓN SUPERFICIAL**

Por otro lado, esta clonación es aquella que se realiza cuando el objeto que se quiere clonar no posee otros que se deban copiar (tales como Integer, Char, Bool). Se realiza un nivel de bits.

CÓMO IMPLEMENTARLO

1. Crea la interfaz del prototipo y declara el método clonar en ella, o, simplemente, añade el método a todas las clases de una jerarquía de clase existente, si la tienes.
2. Una clase de prototipo debe definir el constructor alternativo que acepta un objeto de dicha clase como argumento. El constructor debe copiar los valores de todos los campos definidos en la clase del objeto que se le pasa a la instancia recién creada. Si deseas cambiar una subclase, debes invocar al constructor padre para permitir que la superclase gestione la clonación de sus campos privados.

   Si el lenguaje de programación que utilizas no soporta la sobrecarga de métodos, puedes definir un método especial para copiar la información del objeto. El constructor es el lugar más adecuado para hacerlo, porque entrega el objeto resultante justo después de invocar el operador new.
3. Normalmente, el método de clonación consiste en una sola línea que ejecuta un operador new con la versión prototípica del constructor. Observa que todas las clases deben sobreescribir explícitamente el método de clonación y utilizar su propio nombre de clase junto al operador new. De lo contrario, el método de clonación puede producir un objeto a partir de una clase madre.
4. Opcionalmente, puedes crear un registro de prototipos centralizado para almacenar un catálogo de prototipos de uso frecuente.

Puedes implementar el registro como una nueva clase de fábrica o colocarlo en la clase base de prototipo con un método estático para buscar el prototipo. Este método debe buscar un prototipo con base en el criterio de búsqueda que el código cliente pase al método. El criterio puede ser una etiqueta tipo string o un grupo complejo de parámetros de búsqueda. Una vez encontrado el prototipo adecuado, el registro deberá clonarlo y devolver la copia al cliente.

Por último, sustituye las llamadas directas a los constructores de las subclases por llamadas al método de fábrica del registro de prototipos.

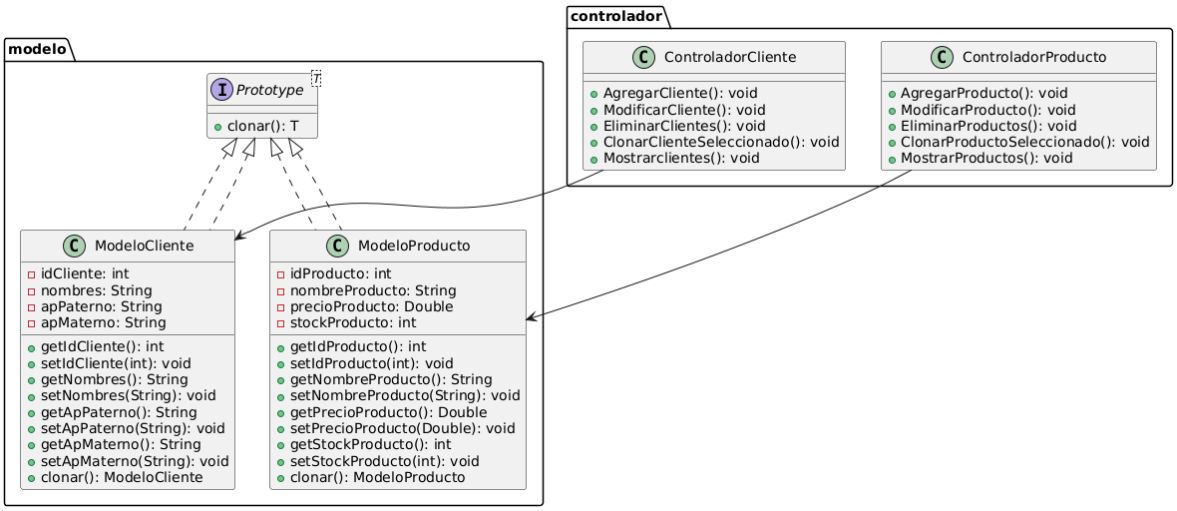
PROS Y CONTRAS

- Puedes clonar objetos sin acoplarlos a sus clases concretas.
- Puedes evitar un código de inicialización repetido clonando prototipos prefabricados.
- Puedes crear objetos complejos con más facilidad.
- Obtienes una alternativa a la herencia al tratar con preajustes de configuración para objetos complejos.
- Clonar objetos complejos con referencias circulares puede resultar complicado.

RELACIONES CON OTROS PATRONES

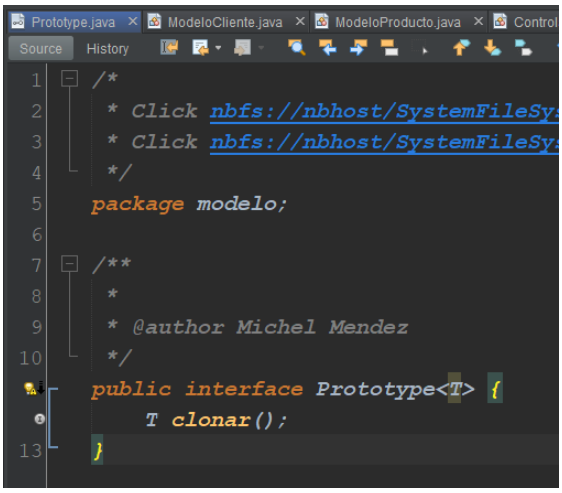
- Muchos diseños empiezan utilizando el Factory Method (menos complicado y más personalizable mediante las subclases) y evolucionan hacia Abstract Factory, Prototype, o Builder (más flexibles, pero más complicados).
- Las clases del Abstract Factory a menudo se basan en un grupo de métodos de fábrica, pero también puedes utilizar Prototype para escribir los métodos de estas clases.
- Prototype puede ayudar a cuando necesitas guardar copias de Comandos en un historial.
- Los diseños que hacen un uso amplio de Composite y Decorator a menudo pueden beneficiarse del uso del Prototype. Aplicar el patrón te permite clonar estructuras complejas en lugar de reconstruirlas desde cero.
- Prototype no se basa en la herencia, por lo que no presenta sus inconvenientes. No obstante, Prototype requiere de una inicialización complicada del objeto clonado. Factory Method se basa en la herencia, pero no requiere de un paso de inicialización.
- En ocasiones, Prototype puede ser una alternativa más simple al patrón Memento. Esto funciona si el objeto cuyo estado quieres almacenar en el historial es suficientemente sencillo y no tiene enlaces a recursos externos, o estos son fáciles de restablecer.
- Los patrones Abstract Factory, Builder y Prototype pueden todos ellos implementarse como Singletons.

DIAGRAMA UML



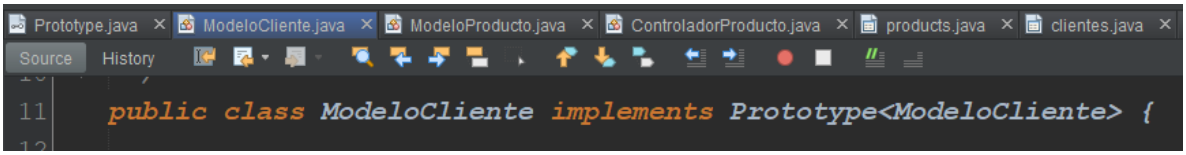
IMPLEMENTACIÓN EN EL PROYECTO

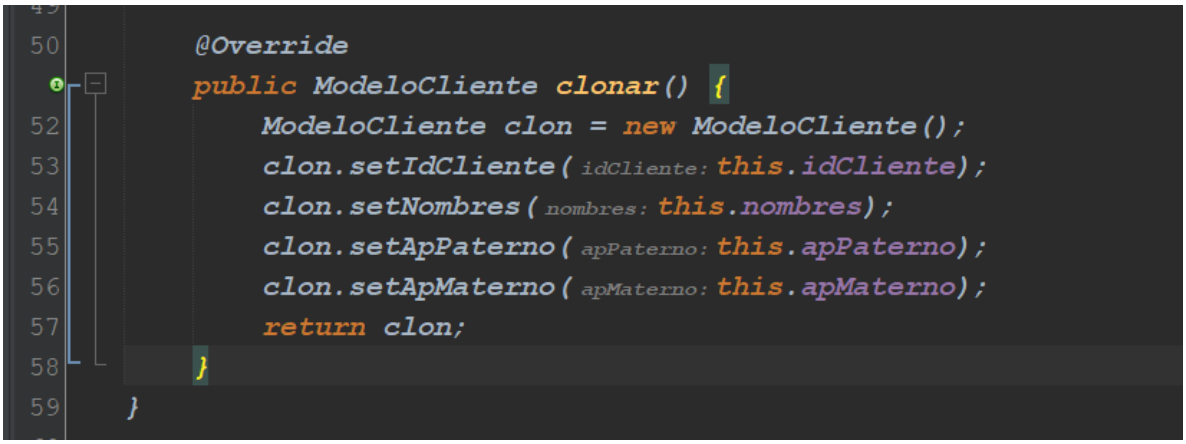
1. **Interfaz Prototype<T>**
   - Se creó la interfaz Prototype<T> con el método clonar().
   - Esta interfaz permite que cualquier clase que la implemente pueda crear una copia de sí misma.

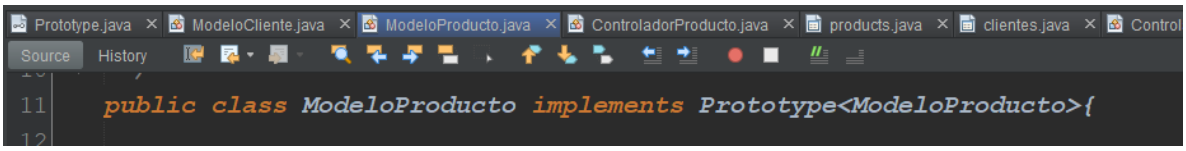


2. **Clases que implementan Prototype**
   - Las clases ModeloCliente y ModeloProducto implementan Prototype y definen el método clonar() para duplicarse.

- Esto permite crear copias de clientes y productos fácilmente, sin tener que repetir código o crear nuevos objetos manualmente.

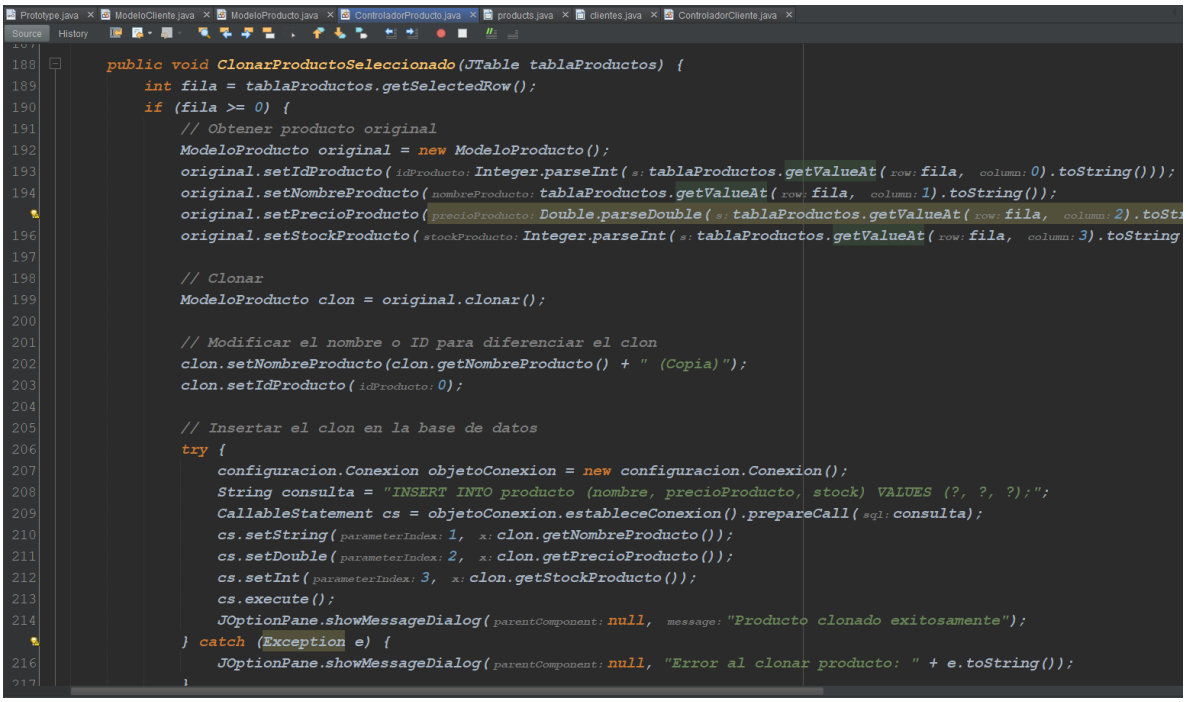






3. **Uso en controladores**

   - En ControladorCliente y ControladorProducto, se implementaron métodos como ClonarClienteSeleccionado() y ClonarProductoSeleccionado(), donde se clona el objeto original usando .clonar(), se modifica si es necesario (como agregar "(Copia)" al nombre), y luego se guarda como nuevo registro en la base de datos.

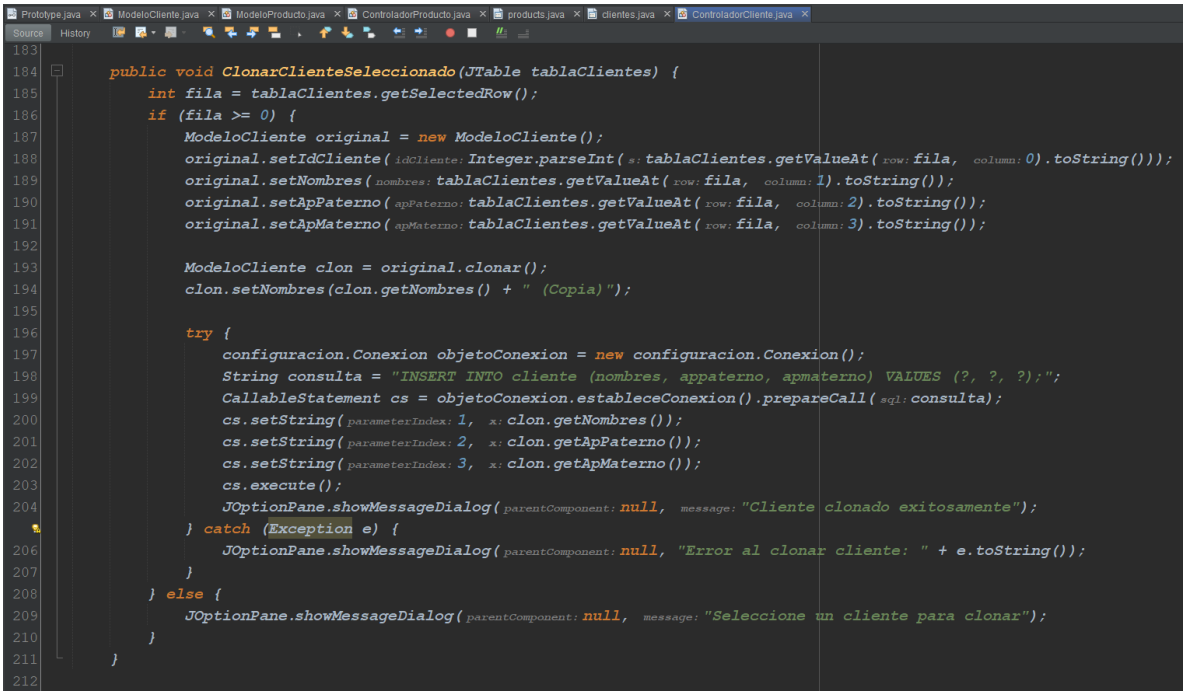


```java
public void ClonarProductoSeleccionado(JTable tablaProductos) {
    int fila = tablaProductos.getSelectedRow();
    if (fila >= 0) {
        // Obtener producto original
        ModeloProducto original = new ModeloProducto();
        original.setIdProducto( idProducto: Integer.parseInt( s: tablaProductos.getValueAt( row: fila,  column: 0).toString()));
        original.setNombreProducto( nombreProducto: tablaProductos.getValueAt( row: fila,  column: 1).toString());
        original.setPrecioProducto( precioProducto: Double.parseDouble( s: tablaProductos.getValueAt( row: fila,  column: 2).toSt
        original.setStockProducto( stockProducto: Integer.parseInt( s: tablaProductos.getValueAt( row: fila,  column: 3).toString

        // Clonar
        ModeloProducto clon = original.clonar();

        // Modificar el nombre o ID para diferenciar el clon
        clon.setNombreProducto(clon.getNombreProducto() + " (Copia)");
        clon.setIdProducto( idProducto: 0);

        // Insertar el clon en la base de datos
        try {
            configuracion.Conexion objetoConexion = new configuracion.Conexion();
            String consulta = "INSERT INTO producto (nombre, precioProducto, stock) VALUES (?, ?, ?);";
            CallableStatement cs = objetoConexion.estableceConexion().prepareCall( sql: consulta);
            cs.setString( parameterIndex: 1,  x: clon.getNombreProducto());
            cs.setDouble( parameterIndex: 2,  x: clon.getPrecioProducto());
            cs.setInt( parameterIndex: 3,  x: clon.getStockProducto());
            cs.execute();
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Producto clonado exitosamente");
        } catch (Exception e) {
            JOptionPane.showMessageDialog( parentComponent: null, "Error al clonar producto: " + e.toString());
        }
```



```java
public void ClonarClienteSeleccionado(JTable tablaClientes) {
    int fila = tablaClientes.getSelectedRow();
    if (fila >= 0) {
        ModeloCliente original = new ModeloCliente();
        original.setIdCliente( idCliente: Integer.parseInt( s: tablaClientes.getValueAt( row: fila,  column: 0).toString()));
        original.setNombres( nombres: tablaClientes.getValueAt( row: fila,  column: 1).toString());
        original.setApPaterno( apPaterno: tablaClientes.getValueAt( row: fila,  column: 2).toString());
        original.setApMaterno( apMaterno: tablaClientes.getValueAt( row: fila,  column: 3).toString());

        ModeloCliente clon = original.clonar();
        clon.setNombres(clon.getNombres() + " (Copia)");

        try {
            configuracion.Conexion objetoConexion = new configuracion.Conexion();
            String consulta = "INSERT INTO cliente (nombres, appaterno, apmaterno) VALUES (?, ?, ?);";
            CallableStatement cs = objetoConexion.estableceConexion().prepareCall( sql: consulta);
            cs.setString( parameterIndex: 1,  x: clon.getNombres());
            cs.setString( parameterIndex: 2,  x: clon.getApPaterno());
            cs.setString( parameterIndex: 3,  x: clon.getApMaterno());
            cs.execute();
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Cliente clonado exitosamente");
        } catch (Exception e) {
            JOptionPane.showMessageDialog( parentComponent: null, "Error al clonar cliente: " + e.toString());
        }
    } else {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Seleccione un cliente para clonar");
    }
}
```

Ventajas en este proyecto:

- Reutilización de objetos sin tener que instanciarlos desde cero.
- Reducción de duplicación de código para copiar productos/clientes.
- Separación clara de responsabilidades entre modelos y controladores.

CONCLUSIÓN

La implementación del patrón de diseño Prototype en de sistema de gestión de clientes y productos ha demostrado ser una estrategia efectiva para mejorar la modularidad, reutilización de código y facilidad de mantenimiento del proyecto. Al permitir la clonación de objetos como clientes y productos, se reduce la necesidad de crear nuevas instancias desde cero, lo que optimiza el desarrollo y mejora la experiencia del usuario al automatizar procesos como la duplicación de registros.

Este patrón también promueve el principio de bajo acoplamiento, ya que las clases que desean clonar objetos no necesitan conocer su lógica interna de creación, sino únicamente su interfaz Prototype. Esta separación de responsabilidades es fundamental para construir aplicaciones escalables y bien estructuradas.

Además, la aplicación de este patrón refuerza el uso de buenas prácticas de programación orientada a objetos, haciendo que el sistema sea más flexible frente a cambios y futuras extensiones. En conclusión, el uso del patrón Prototype en nuestro proyecto no solo resolvió una necesidad concreta (duplicación de datos), sino que también elevó la calidad arquitectónica del software.

REFERENCIAS

Disennio. (2018, 15 junio). Patrón de diseño Prototype - disennio - Medium. *Medium*. https://medium.com/@diseniio2016/patr%C3%B3n-de-dise%C3%B1o-prototype-8447ee519165

Prototype. (s. f.). https://refactoring.guru/es/design-patterns/prototype