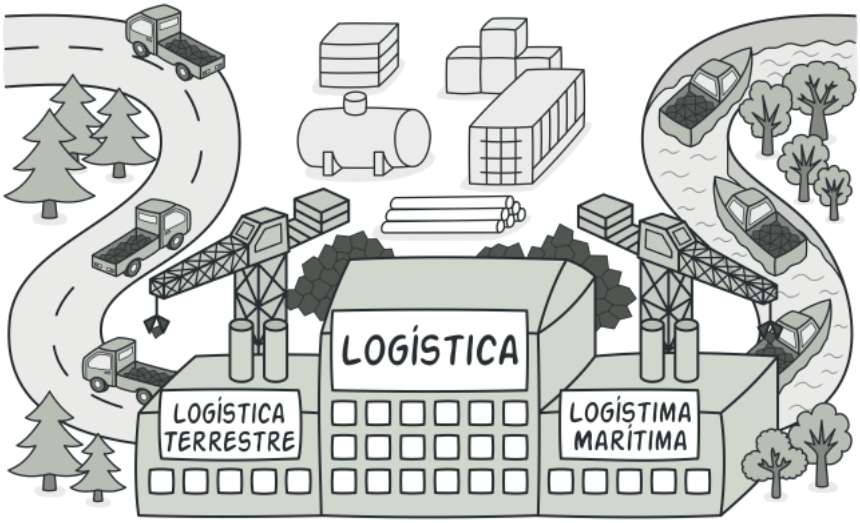
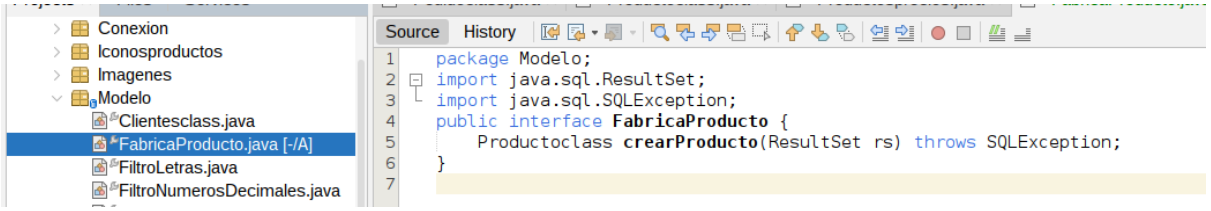
PATRÓN DE DISEÑO:MÉTODO DE FABRICACIÓN

Propósito

Factory Method es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.



Primero, necesita definir una interfaz que declare el métodopara generar productos. En este caso, será el método crearProducto.



El patrón Factory Method sugiere que, en lugar de llamar al operador new para construir objetos directamente, se invoque a un método fábrica especial. No te preocupes: los objetos se siguen creando a través del operador new, pero se invocan desde el método fábrica. Los objetos devueltos por el método fábrica a menudo se denominan productos.

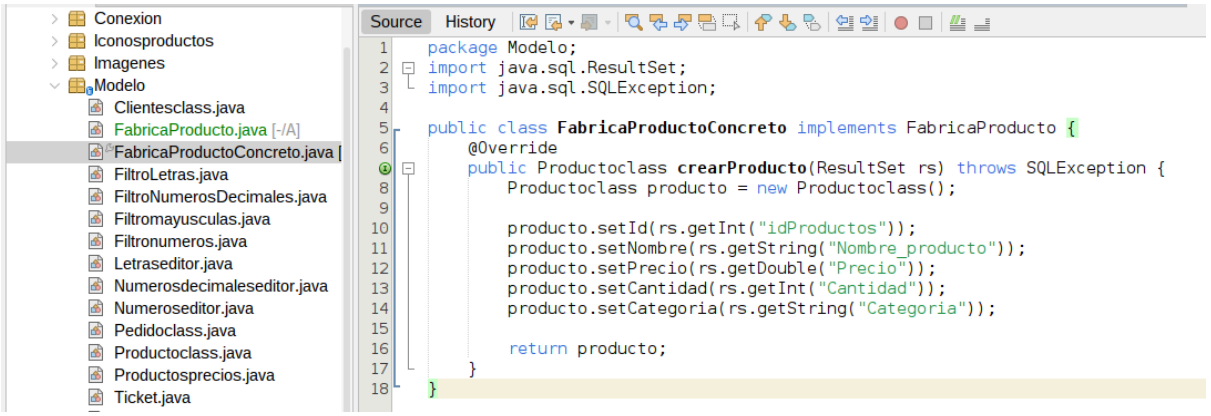
Entonces debemos de crear otra clase que nos ayude a complementar nuestra interfaz que ya tenemos que quedaria de la siguiente forma:

```
public class FabricaProductoConcreto implements FabricaProducto {
    @Override
    public Productoclass crearProducto(ResultSet rs) throws SQLException {
        Productoclass producto = new Productoclass();

        producto.setId(rs.getInt("idProductos"));
        producto.setNombre(rs.getString("Nombre_producto"));
```

```
        producto.setPrecio(rs.getDouble("Precio"));
        producto.setCantidad(rs.getInt("Cantidad"));
        producto.setCategoria(rs.getString("Categoria"));

        return producto;
    }
}
```



Ahora tendremos que cambiar la forma en la que se construyen los , ya que se crean de forma independiente, debemos de crearlas desde nuestra fábrica que ya implementamos, así que modificares la clase de Productoclass.java.

Quedando nuestra consulta de obtener productos de la siguiente forma:

```
public List<Productoclass> obtenerProductos() {
    List<Productoclass> productos = new ArrayList<>();
    Conexion conex = new Conexion();
    String sql = "SELECT idProductos, Nombre_producto, Precio, Cantidad, Categoria FROM Productos";
    FabricaProducto fabrica = new FabricaProductoConcreto();

    try (Connection con = conex.getConnection();
         PreparedStatement pst = con.prepareStatement(sql);
         ResultSet rs = pst.executeQuery()) {

        while (rs.next()) {
            Productoclass producto = fabrica.crearProducto(rs);
            productos.add(producto);
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Error al obtener productos: " + e.toString());
    }

    return productos;
}
```

Igual modificamos la de ObtenerClientesPorcategorias:

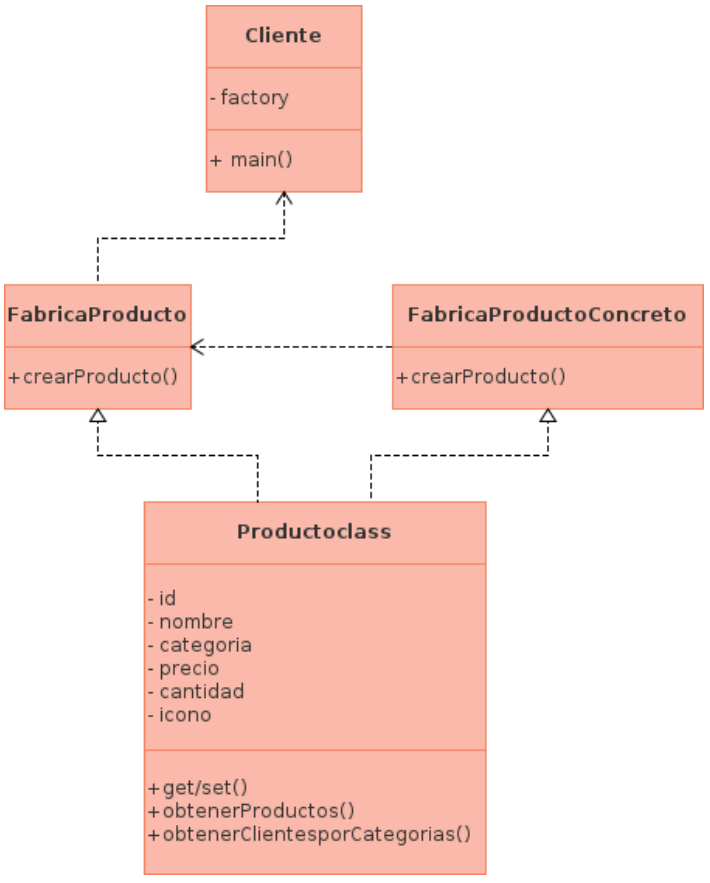
```java
public List<Productoclass> obtenerClientesProductoscatego(String categoria) {
    List<Productoclass> productos = new ArrayList<>();
    Conexion conex = new Conexion();
    String sql = "SELECT idProductos, Nombre_producto, Precio, Cantidad, Categoria FROM Productos WHERE Categoria LIKE ?";
    FabricaProducto fabrica = new FabricaProductoConcreto();

    try (Connection con = conex.getConnection();
         PreparedStatement pst = con.prepareStatement(sql)) {

        pst.setString(1, "%" + categoria + "%");

        try (ResultSet rs = pst.executeQuery()) {
            while (rs.next()) {
                Productoclass producto = fabrica.crearProducto(rs);
                productos.add(producto);
            }
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Error al obtener productos por categoría: " + e.toString());
    }

    return productos;
}
}
```



¿De esta forma aplicamos el patrón de método de fabricación en que nos ayuda esto de crear objetos de forma concreta en lugar de crear objetos de forma tradicional?

Normalmente, los productos se crean en las consultas en ObtenerProductos por ejemplo, con esta implementacion solo el método de ObtenerProductos no es necesario que sepan como se crean los objetos solo de donde obtenerlos, eso refleja en que si más adelante se agregan datos a los productos o modificar algunos solo se crea una nueva fábrica adecuada a esos productos sin afectar o modificar las fabricas ya existentes.

Ventajas	Desventajas
Desacoplamiento: Permite desacoplar el código que utiliza los objetos de su creación, mejorando la modularidad.	Complejidad adicional: Introduce más clases (interfaz de fábrica y fábricas concretas), lo que puede hacer el diseño más complejo.
Flexibilidad: Facilita cambiar la lógica de creación de objetos sin afectar otras partes del sistema.	Mayor cantidad de clases: Aumenta el número de clases en el proyecto, lo que puede ser inconveniente en sistemas pequeños.
Escalabilidad: Permite manejar fácilmente diferentes tipos de objetos sin modificar el código existente.	Más esfuerzo inicial: Requiere un esfuerzo adicional en la fase de diseño y desarrollo para implementar el patrón.

El Factory Method es un patrón de diseño muy útil cuando se necesita desacoplar la creación de objetos de su uso. Si bien introduce cierta complejidad adicional y requiere más clases, su principal ventaja es la flexibilidad y extensibilidad que ofrece. Si anticipa que el sistema cambiará con el tiempo (por ejemplo, agregar nuevos tipos de productos, nuevas fuentes de datos o nuevas formas de crear productos), este patrón proporciona una base sólida que facilita la evolución del sistema sin romper el código existente.