Tecnológico Nacional De México

Instituto Tecnológico de Oaxaca

Ingeniería En Sistemas Computacionales

Diseño e Implementación de Software con Patrones.

7 am – 8 am

Unidad 2

“Patrón de Factory Method”

Presenta:

Copy Max

Nombres	Numero de Control
Bautista Fabian Max	**C19160532**
Celis Delgado Jorge Eduardo	**21160599**
Flores Guzmán Alan Ismael	**20161193**
García Osorio Bolívar	**20161819**
Pérez Barrios Diego	**21160750**
Perez Martínez Edith Esmeralda	**21160752**
Sixto Morales Ángel	**21160797**

Periodo Escolar:

Febrero – Julio

Grupo:

7SB

Maestro:

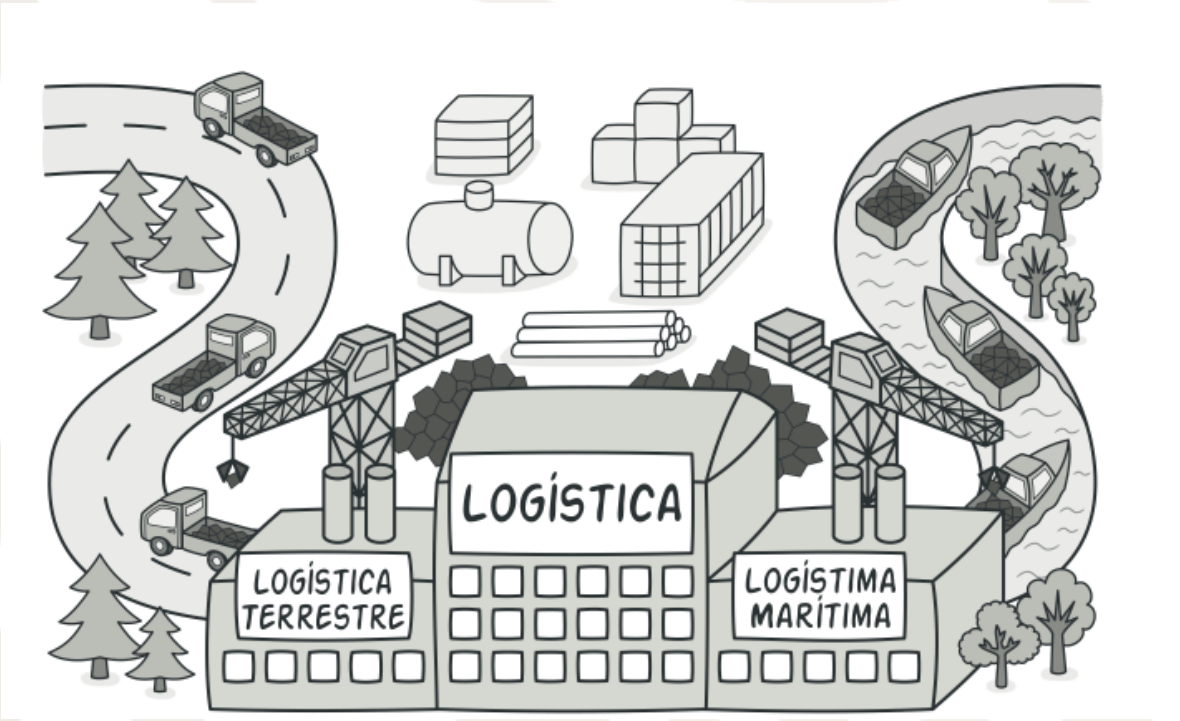
Espinoza Pérez Jacob

Oaxaca de Juárez, Oaxaca **Marzo, 2025**

INDICE

Patrón de Diseño Factory Method

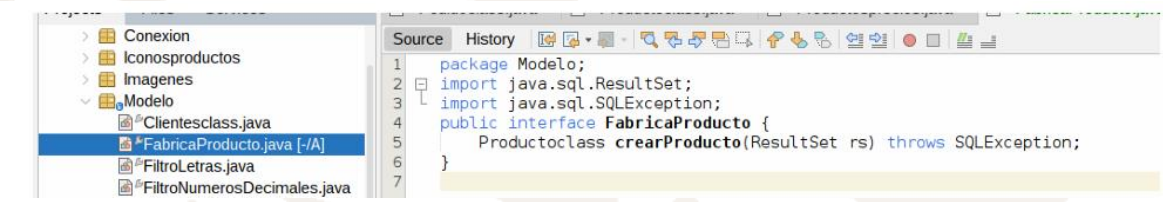
Factory Method es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.



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*.

Implementación en el Proyecto:

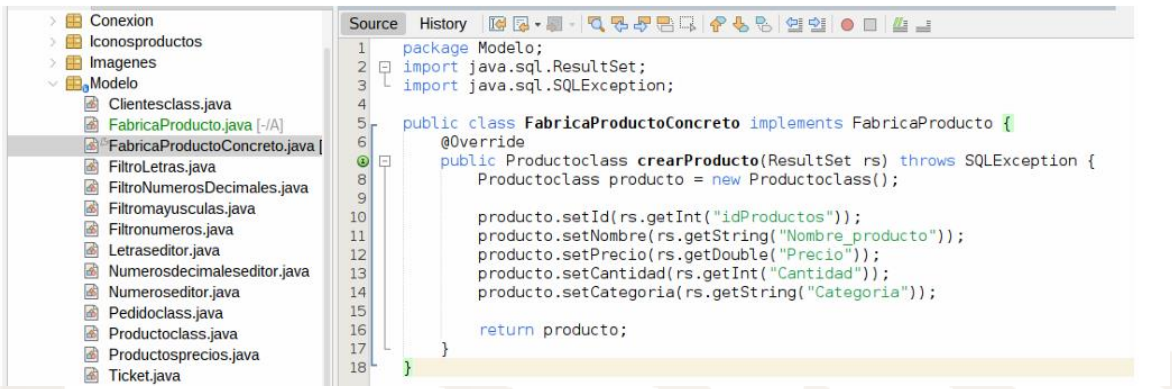
Primero, necesita definir una interfaz que declare el métodopara generar productos. En este caso, será el método crearProducto.



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:

Quedando nuestra consulta de obtener productos de la siguiente forma:

public List obtenerProductos() {

List 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);

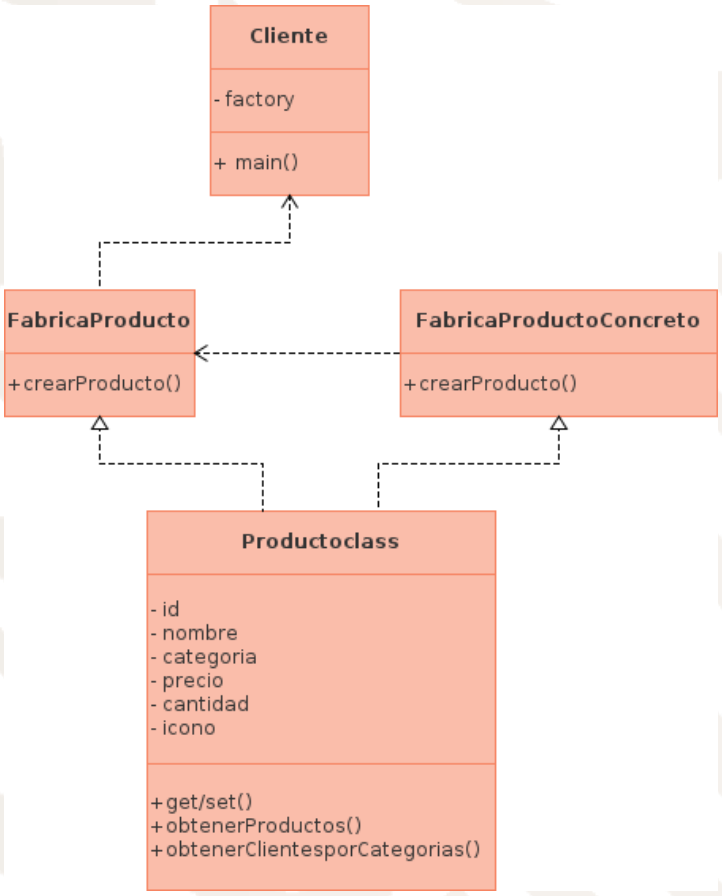
} } catch (SQLException e) { JOptionPane.showMessageDialog(null, "Error al obtener productos: " + e.toString());

} return productos; }

Igual modificamos la de ObtenerClientesPorcategorias:

```
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;
}
}
```

Estructura UML



¿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 y Desventajas

Ventajas	Desventajas
Evitas un acoplamiento fuerte entre el creador y los productos concretos.	Puede ser que el código se complique, ya que debes incorporar una multitud de nuevas subclases para implementar el patrón. La situación ideal sería introducir el patrón en una jerarquía existente de clases creadoras.
Principio de responsabilidad única. Puedes mover el código de creación de producto a un lugar del programa, haciendo que el código sea más fácil de mantener.	Más esfuerzo inicial: Requiere un esfuerzo adicional en la fase de diseño y desarrollo para implementar el patrón.
Principio de abierto/cerrado. Puedes incorporar nuevos tipos de productos en el programa sin descomponer el código cliente existente.	

Conclusión

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.