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 Diseño Composite”

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:

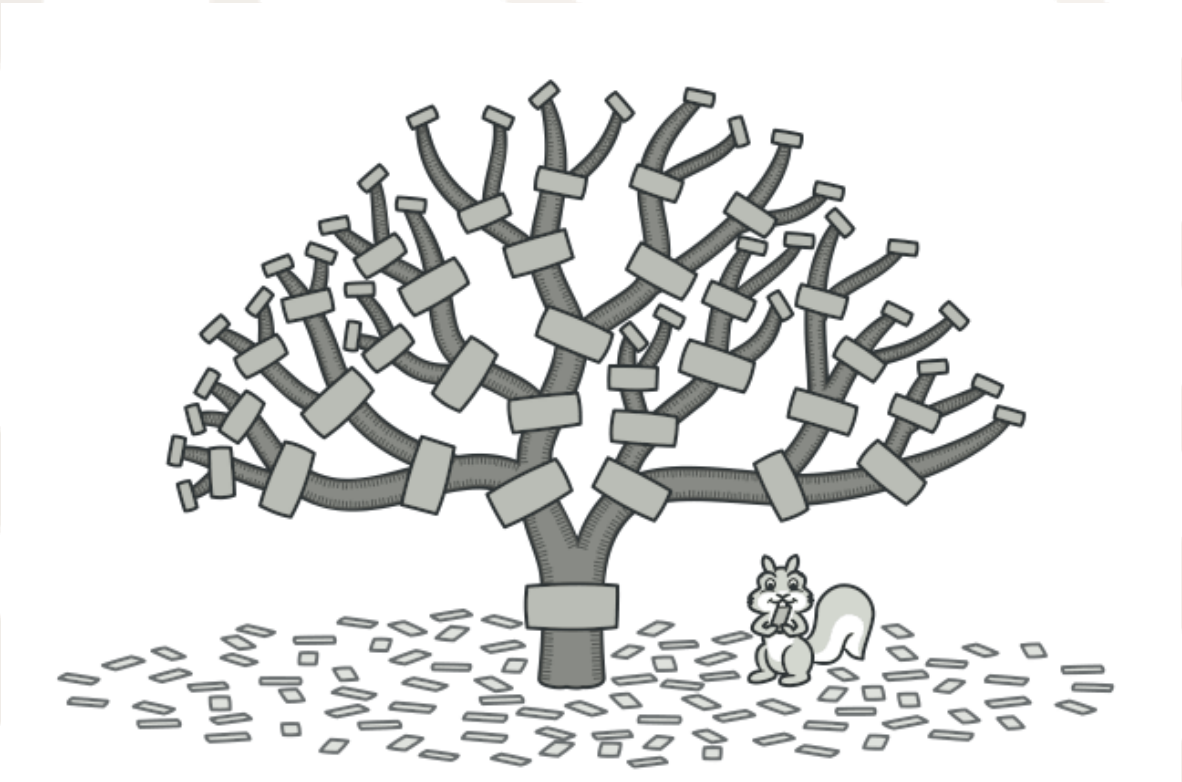
Espinoza Pérez Jacob

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

INDICE

Patrón de Diseño Composite

Composite es un patrón de diseño estructural que te permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.



Implementación en el Proyecto:

La clase PaqueteDeProductos se usara para poder generar el producto en un encapsulamiento y se pueda hacer un paquete.

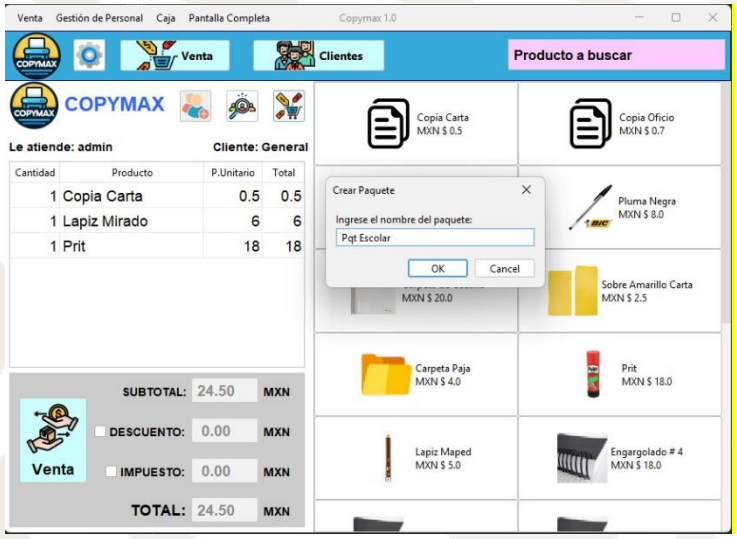
```
package Modelo;

import java.util.ArrayList;
import java.util.List;

public class PaqueteDeProductos implements ComponenteProducto {
```

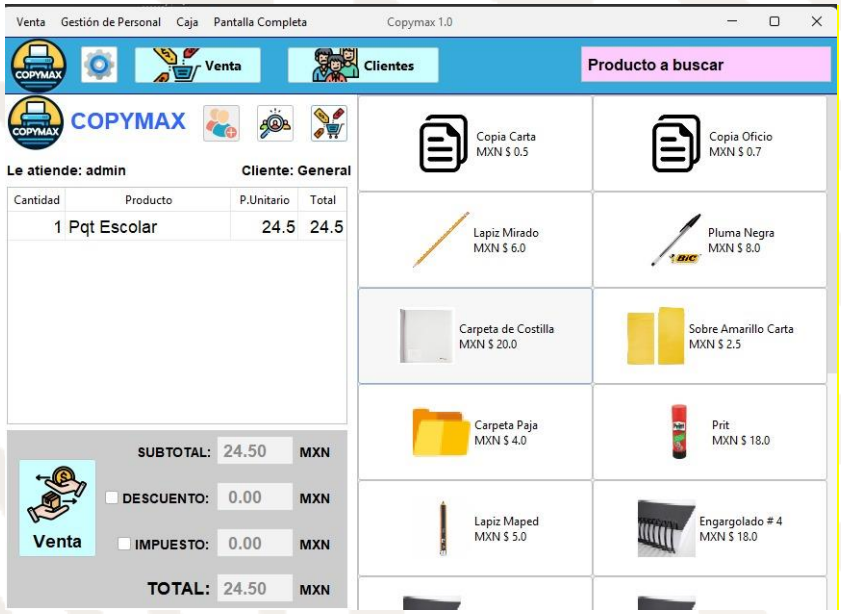
```
    private String nombre;
    private List<ComponenteProducto> productos = new ArrayList<>();

    public PaqueteDeProductos(String nombre) {
        this.nombre = nombre;
    }

    public void agregarProducto(ComponenteProducto producto) {
        productos.add(producto);
    }

    @Override
    public double getPrecio() {
        double total = 0;
        for (ComponenteProducto producto : productos) {
            total += producto.getPrecio();
        }
        return total;
    }

    @Override
    public String getDescripcion() {
        StringBuilder descripcion = new StringBuilder(nombre + ":\n");
        for (ComponenteProducto producto : productos) {
            descripcion.append(" - ").append(producto.getDescripcion()).append("\n");
        }
        descripcion.append("Total: $").append(getPrecio());
        return descripcion.toString();
    }
}
```

En el método ActionPerformed del botón Btnformarpaquete estará la implementación del programa

```
private void BtnformarpaqueteActionPerformed(java.awt.event.ActionEvent evt) {
        // Solicitar el nombre del paquete
    String nombrePaquete = JOptionPane.showInputDialog(this, "Ingrese el nombre del paquete:", "Crear Paquete", JOptionPane.PLAIN_MESSAGE);

    if (nombrePaquete == null || nombrePaquete.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "El nombre del paquete no puede estar vacío.", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    // Calcular el total de todas las filas
    double total = calcularTotalTabla();

    // Crear el paquete (usando el patrón Composite)
    PaqueteDeProductos paquete = new PaqueteDeProductos(nombrePaquete);

    // Obtener el modelo de la tabla
    DefaultTableModel model = (DefaultTableModel) jTableticket.getModel();

    // Recorrer todas las filas para agregar los productos al paquete
    for (int i = 0; i < model.getRowCount(); i++) {
```

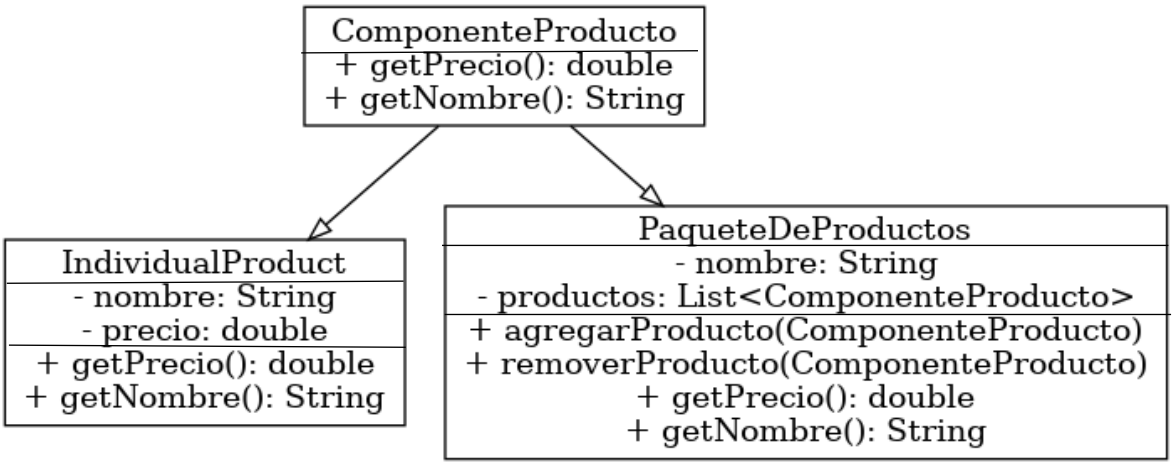
```
        Object valor = model.getValueAt(i, 1); // Columna 1: Descripción del producto
        if (valor instanceof ComponenteProducto) {
            ComponenteProducto producto = (ComponenteProducto) valor;
            paquete.agregarProducto(producto);
        }
    }
```





```
    // Limpiar la tabla
    limpiarTabla();

    // Agregar el paquete como una nueva fila en la tabla
    Object[] nuevaFila = {1, nombrePaquete, total,total}; // Ajusta las columnas según tu tabla
    model.addRow(nuevaFila);

    // Actualizar la tabla
  }
```

Estructura UML



Ventajas y Desventajas

Ventajas	Desventajas
Puedes trabajar con estructuras de árbol complejas con mayor comodidad: utiliza el polimorfismo y la recursión en tu favor.	Puede resultar difícil proporcionar una interfaz común para clases cuya funcionalidad difiere demasiado. En algunos casos, tendrás que generalizar en exceso la interfaz componente, provocando que sea más difícil de comprender.
Principio de abierto/cerrado. Puedes introducir nuevos tipos de elemento en la aplicación sin descomponer el código existente, que ahora funciona con el árbol de objetos.	

Conclusión

El patrón Composite permite tratar estructuras jerárquicas de objetos de manera uniforme, ya sea que se trate de un solo objeto o de una composición de varios como facilita la manipulación de estructuras anidadas y simplifica la gestión de objetos complejos.

Todos los elementos definidos por el patrón Composite comparten una interfaz común. Utilizando esta interfaz, el cliente no tiene que preocuparse por la clase concreta de los objetos con los que funciona.