TEMPLATE

¿QUÉ ES EL PATRÓN TEMPLATE?

El Patrón Template (Template Method) es un patrón de diseño de comportamiento que define la estructura básica de un algoritmo, permitiendo que las subclases modifiquen ciertos pasos específicos sin cambiar la estructura general del algoritmo. Es especialmente útil cuando se tienen procesos comunes con pequeños detalles que varían.

¿CUANDO USAR EL PATRON TEMPLATE?

El Patrón Template es especialmente útil en situaciones donde:

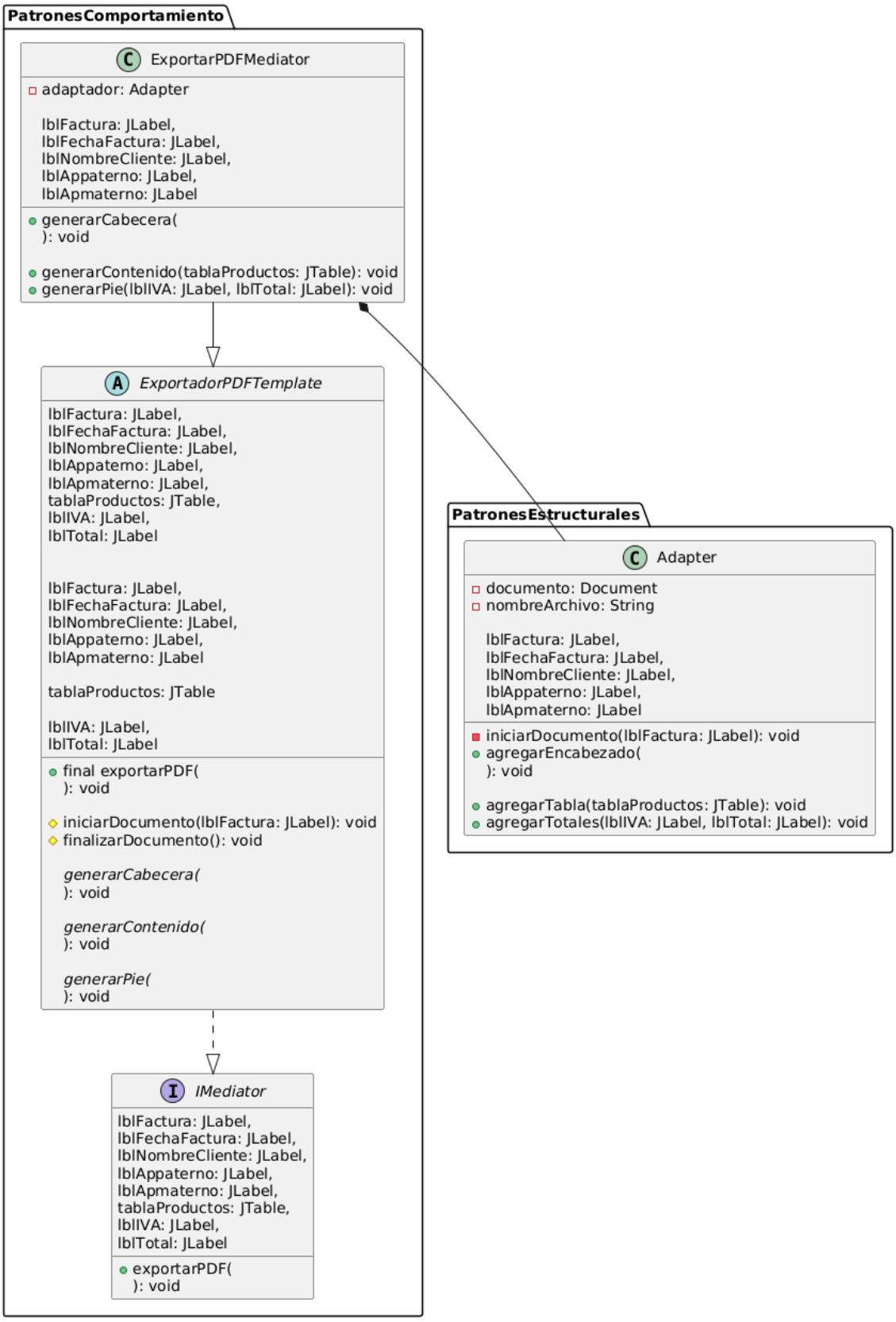
1. Existe un algoritmo general con variaciones menores:
    - Cuando tienes algoritmos con pasos comunes definidos, pero con algunos pasos específicos que cambian según el contexto o la implementación.
2. Quieres evitar duplicación de código:
    - Cuando varias clases realizan tareas similares con pequeñas diferencias. En lugar de duplicar el código común, el Patrón Template permite extraerlo a una clase abstracta.
3. Quieres mantener control sobre el algoritmo general:
    - Cuando deseas garantizar que ciertas partes de un algoritmo no puedan modificarse, asegurando así la estructura o secuencia de ejecución.
4. Quieres mejorar la flexibilidad y extensibilidad:
    - Facilita añadir nuevas variaciones de un algoritmo mediante subclases, sin alterar la lógica general.

ESTRUCTURA DEL PATRÓN TEMPLATE

El Patrón Template se estructura principalmente en:

1. Clase Abstracta
    - Define el método plantilla que contiene la secuencia de pasos del algoritmo.
    - Declara métodos abstractos o ganchos que serán implementados por subclases concretas.
2. Clases Concretas
    - Implementan los métodos abstractos específicos definidos en la clase abstracta.

EJEMPLO UML DEL PATRÓN TEMPLATE



IMPLEMENTACION DEL METODO TEMPALTE DENTRO DEL CODIGO

En nuestra solución, el Template Method se concreta en la clase abstracta ExportadorPDFTemplate, donde definimos un único método público y final:

```java
public abstract class ExportadorPDFTemplate implements IMediator {

    @Override
    public final void exportarPDF(JLabel lblFactura,
                                  JLabel lblFechaFactura,
                                  JLabel lblNombreCliente,
                                  JLabel lblAppaterno,
                                  JLabel lblApmaterno,
                                  JTable tablaProductos,
                                  JLabel lblIVA,
                                  JLabel lblTotal) {
        iniciarDocumento();
        generarCabecera(lblFactura, lblFechaFactura, lblNombreCliente, lblAppaterno, lblApmaterno);
        generarContenido(tablaProductos);
        generarPie(lblIVA, lblTotal);
        finalizarDocumento();
    }

    /** Paso fijo (hook): configuración previa al PDF */
    protected void iniciarDocumento() {
        System.out.println("⏳ Iniciando exportación de PDF…");
        // Por ejemplo: crear carpeta, abrir stream, etc.
    }

    /** Paso variable: la subclase define cómo crea la cabecera */
    protected abstract void generarCabecera(JLabel lblFactura,
                                            JLabel lblFechaFactura,
                                            JLabel lblNombreCliente,
```

1. **exportarPDF(...)**
    - Es el **Template Method** porque **define el orden** de los pasos: iniciar → cabecera → contenido → pie → finalizar.
    - Está marcado final para garantizar que NUNCA se cambie el flujo.
2. **Hooks (iniciarDocumento / finalizarDocumento)**
    - Métodos con implementación por defecto que las subclases **pueden** (pero **no deben**) sobreescribir.
    - Sirven para tareas comunes como abrir/cerrar el Document.
3. **Pasos abstractos (generarCabecera, generarContenido, generarPie)**
    - Definen los “puntos de extensión” donde las subclases inyectan la lógica concreta (en nuestro caso, delegan al Adapter).

La subclase concreta **no toca** el flujo: solo **implementa** esos pasos:

```java
import PatronesEstructurales.Adapter;
import javax.swing.*;

/**
 * Mediator que extiende el template de exportación y solo implementa
 * los pasos concretos, delegando a tu Adapter para el volcado real.
 */
public class ExportarPDFMediator extends ExportadorPDFTemplate {

    private final Adapter adaptadorPDF;

    public ExportarPDFMediator() {
        this.adaptadorPDF = new Adapter();
    }

    @Override
    protected void generarCabecera(JLabel lblFactura,
                                   JLabel lblFechaFactura,
                                   JLabel lblNombreCliente,
                                   JLabel lblAppaterno,
                                   JLabel lblApmaterno) {
        // Aquí "arma" la cabecera usando tu adapter
        adaptadorPDF.agregarEncabezado(
            lblFactura, lblFechaFactura,
            lblNombreCliente, lblAppaterno, lblApmaterno
        );
    }
```

- **La subclase no reescribe exportarPDF(...), sólo suministra las piezas que encajan en la estructura.**
- **El flujo siempre será el mismo, garantizando consistencia y evitando duplicación de código.**

Finalmente, cuando pulsamos el botón, invocamos solo el método del template