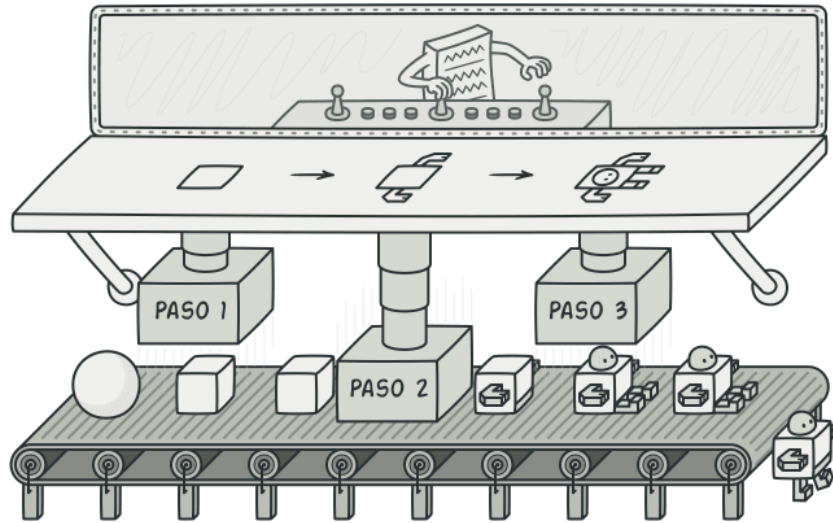


# Patrón Builder

**Builder** es un patrón de diseño creacional que nos permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.



La clase `Pedidoclass` representa un pedido en el sistema de Copymax. Su propósito es encapsular toda la información relacionada con un pedido, como:

- Datos del cliente (nombre, apellidos, celular).
- Detalles del pedido (servicio, tipo de copia, tamaño, escala, etc.).
- Fechas y horas de emisión y entrega.
- Montos (total, anticipo, resto).
- Estado del pedido (pendiente y finalizado).

El patrón **Builder** se integra en esta clase para facilitar la creación de objetos `Pedidoclass` de manera flexible y legible, especialmente cuando el objeto tiene muchos atributos opcionales.

La clase `Pedidoclass` se utiliza principalmente en las siguientes partes del sistema:

## 1. Generación de Pedidos:

- En la interfaz gráfica donde se capturan los datos del pedido (por ejemplo, en un formulario).
- En el método `llenardatos()`, donde se construye un objeto `Pedidoclass` y se inserta en la base de datos.

## 2. Consulta de Pedidos:

- En métodos que recuperan pedidos desde la base de datos y los muestran en una tabla o reporte.

### 3. Actualización de Pedidos:

- En métodos que modifican el estado de un pedido (por ejemplo, cambiar de "Pendiente" a "Finalizado").

La clase Pedidoclass se divide en dos partes principales:

- private final int numpedido;
- private final int idusuario;
- private final int idcliente;
- private final String Nombredeusuario;
- private final String Nombrecliente;
- private final String Apellidoscliente;
- private final String Celularcliente;
- private final String Servicio;
- private final String Tipodecopia;
- private final String Tamaño;
- private final String Escala;
- private final java.sql.Date fechaEmision;
- private final java.sql.Time horaEmision;
- private final java.sql.Date fechaEntrega;
- private final java.sql.Time horaEntrega;
- private final double total;
- private final double Anticipo;
- private final double Resto;
- private final String Status;

y la clase Builder esta clase interna es la que permite construir los objetos de Pedidoclass paso a paso, esta es su estructura:

```
public static class Builder {
    // Atributos (los mismos que en Pedidoclass)
    private int numpedido;
    private int idusuario;
    private int idcliente;
    private String Nombredeusuario;
    private String Nombrecliente;
    private String Apellidoscliente;
    private String Celularcliente;
    private String Servicio;
    private String Tipodecopia;
    private String Tamaño;
    private String Escala;
    private java.sql.Date fechaEmision;
    private java.sql.Time horaEmision;

    private java.sql.Date fechaEntrega;
    private java.sql.Time horaEntrega;
    private double total;
    private double Anticipo;
    private double Resto;
    private String Status;

    // Métodos para configurar los atributos
    public Builder setNumpedido(int numpedido) {
        this.numpedido = numpedido;
        return this;
    }

    public Builder setIdusuario(int idusuario) {
        this.idusuario = idusuario;
    }
}
```

```

        return this;
    }

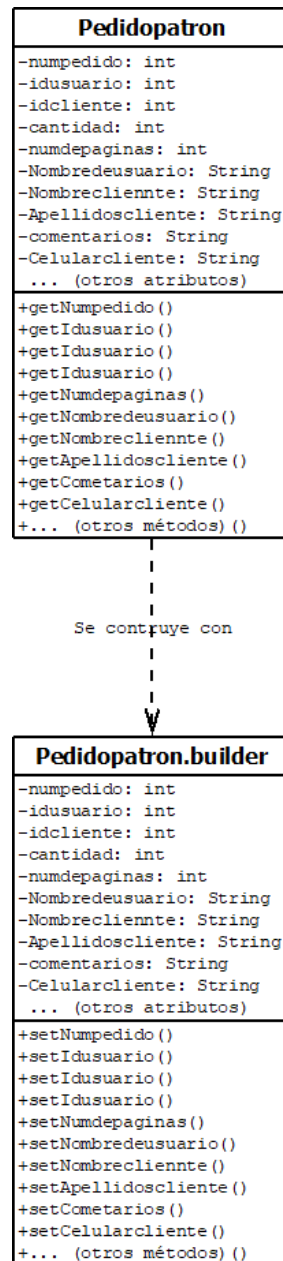
    // ... (métodos similares para los demás atributos)

    // Método build para crear la instancia de Pedidoclass
    public Pedidoclass build() {
        return new Pedidoclass(this);
    }
}

```

Esta clase nos permitirá que al momento de crear el pedido no se pueda cambiar nada de la información creada.

### Estructura:



## Ventajas y desventajas

Ventajas	Desventajas
<b>Flexibilidad:</b> Permite construir objetos Pedidoclass con solo los atributos necesarios, sin necesidad de usar constructores con muchos parámetros.	<b>Código adicional:</b> El patrón Builder requiere escribir más código (la clase Builder y sus métodos), lo que puede aumentar la complejidad inicial.
<b>Inmutabilidad:</b> Los objetos Pedidoclass pueden ser inmutables (sus atributos no cambian después de la construcción), lo que es útil en aplicaciones concurrentes.	<b>Sobrecarga para objetos simples:</b> Si el objeto tiene pocos atributos, el patrón Builder puede ser excesivo y complicar innecesariamente el código.
<b>Mantenibilidad:</b> Si en el futuro se agregan más atributos, solo necesitas modificar la clase Builder, sin afectar el resto del código.	<b>Curva de aprendizaje:</b> Los desarrolladores que no están familiarizados con el patrón Builder pueden encontrar difícil entender su uso al principio.

## Conclusión:

La integración del patrón **Builder** en la clase Pedidoclass mejora significativamente la flexibilidad, legibilidad y mantenibilidad del código, especialmente cuando se trata de objetos con muchos atributos opcionales. Aunque requiere un poco más de código inicial, los beneficios a largo plazo justifican su uso en este contexto.