ANÁLISIS Y JUSTIFICACIÓN DEL PATRÓN BUILDER
Sistema de Hamburguesas Personalizadas

1. ANÁLISIS DEL PROBLEMA

En un sistema de pedidos de comida rápida, las hamburguesas pueden configurarse de múltiples maneras dependiendo de las preferencias del cliente, todas parten de componentes obligatorios (pan y carne) y pueden agregárseles opcionales (queso, lechuga, tomate, salsas, bacon, etc.).

El problema radica en que usar constructores tradicionales con múltiples parámetros genera constructores con múltiples versiones para diferentes combinaciones, código poco legible con llamadas difíciles de entender, alta dificultad de mantenimiento al agregar nuevos ingredientes, (o atributos en general), y errores potenciales al confundir el orden de parámetros similares.

El objeto base sobre el cual trabajamos es la Hamburguesa. Los atributos obligatorios son el tipo de pan (Pan Brioche, Pan Integral, Pan Blanco, etc.) y el tipo de carne (Res, Pollo, Cerdo, Vegetariana). Los atributos opcionales incluyen el queso (Cheddar, Suizo, Americano) y una lista variable de ingredientes adicionales.

2. JUSTIFICACIÓN DEL PATRÓN BUILDER

El Patrón Builder es una muy buena solución para este tipo de problemas porque separa la construcción de la representación; permitiendo que el Builder maneje toda la lógica mientras la clase Hamburger solo representa y se encarga el producto final (respetando los principios SOLID). Evita constructores extraños al proporcionar una única forma clara de crear hamburguesas. Diferencia claramente obligatorios de opcionales, donde el constructor del Builder requiere los obligatorios y los métodos encadenables configuran los opcionales.

Mejora la legibilidad mediante código auto-documentado, garantiza objetos consistentes e inmutables con atributos final que no pueden modificarse después de la creación, y facilita el mantenimiento permitiendo agregar nuevos ingredientes opcionales sin cambiar código existente ni romper compatibilidad, nuevamente, respetando los principios SOLID.

3. VENTAJAS DE ESTA IMPLEMENTACIÓN

La implementación garantiza inmutabilidad completa con atributos “final”, ausencia de setters y copias defensivas en getters de colecciones. La validación robusta verifica atributos obligatorios en el constructor y lanza excepciones si son nulos o vacíos. La sintaxis fluida mediante métodos encadenables produce código natural e intuitivo. El encapsulamiento se logra con constructor privado, haciendo que el Builder sea la única forma de crear instancias, proporcionando control total sobre el proceso.

4. COMPARACIÓN: CON BUILDER VS SIN BUILDER

Sin Builder, un constructor con muchos parámetros sería confuso y propenso a errores, sin claridad sobre qué representa cada parámetro y con dificultad para manejar opcionales. Con Builder, el código es claro, legible y auto-documentado, donde crear hamburguesas sin opcionales es natural sin pasar valores nulos.

5. CONCEPTOS CLAVE APLICADOS

La separación de responsabilidades establece que Hamburger representa el producto mientras Builder sabe cómo construirlo. La construcción paso a paso sigue un proceso definido: crear Builder con obligatorios, configurar opcionales mediante métodos encadenables, e invocar build() para obtener el objeto inmutable final. La reutilización permite crear diferentes representaciones usando el mismo proceso, desde hamburguesas simples hasta complejas.

6. CONCLUSIÓN

La implementación del Patrón Builder demuestra cómo este patrón resuelve eficazmente la creación de objetos complejos con múltiples configuraciones. Al separar construcción de representación, diferenciar obligatorios de opcionales, y garantizar inmutabilidad, se logra código legible, mantenible y robusto que cumple con principios SOLID y mejores prácticas de programación orientada a objetos.