

REFACTORIZACIÓN: EL ARTE DEL CÓDIGO LIMPIO

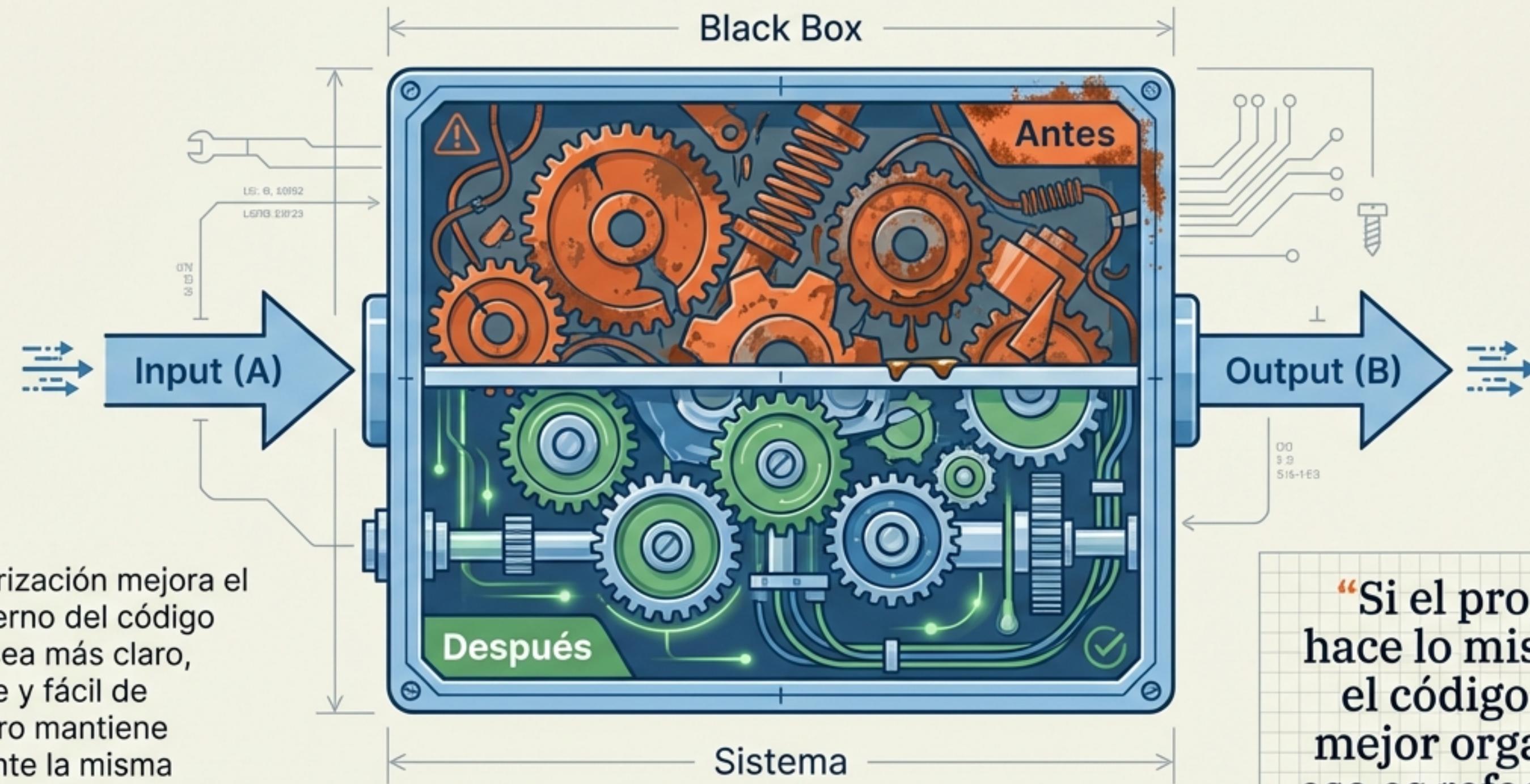
Principios, Patrones y Limitaciones



Guía técnica para mejorar el diseño interno sin alterar la funcionalidad

REF

La Definición: Cambiar el "Cómo", no el "Qué"



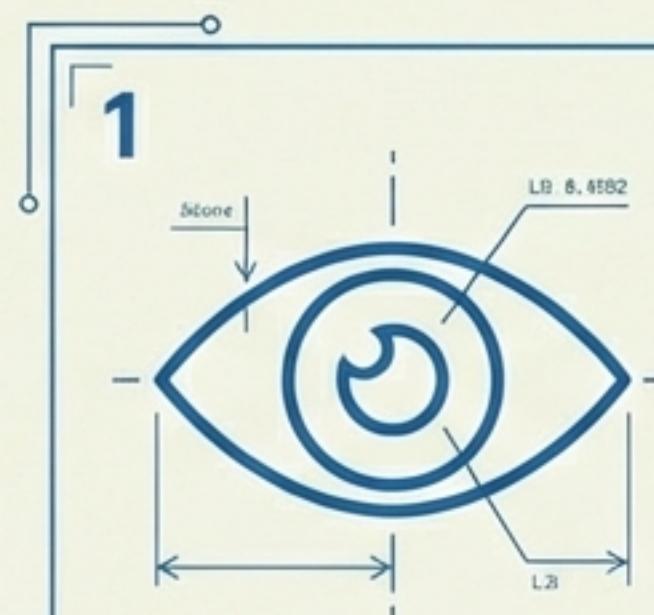
Refactorización vs. Optimización

 Optimización	 Refactorización
Focus: Recursos (Máquina) <small>JetBrains Mono </small>	Focus: Desarrolladores (Humanos) <small>JetBrains Mono </small>
Metrics: Tiempo, Memoria, Queries, Red <small>JetBrains Mono </small>	Metrics: Estructura, Legibilidad, Mantenimiento <small>JetBrains Mono </small>
Goal: Rendimiento ↗ <small>Roboto Serif </small>	Goal: Entendimiento💡 <small>Roboto Serif </small>

Relación Habitual

Muchas veces se refactoriza antes de optimizar. Un código más claro facilita detectar cuellos de botella y reduce el riesgo de romper la lógica durante la optimización.

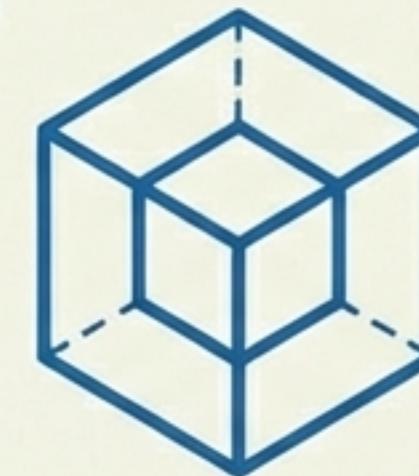
Los Cuatro Pilares Fundamentales



Código Legible

Nombres claros,
métodos cortos,
estructura fácil de
seguir.

2



Responsabilidad Única

Cada clase o método hace
“una sola cosa”. Evitar
los “métodos monstruo”.

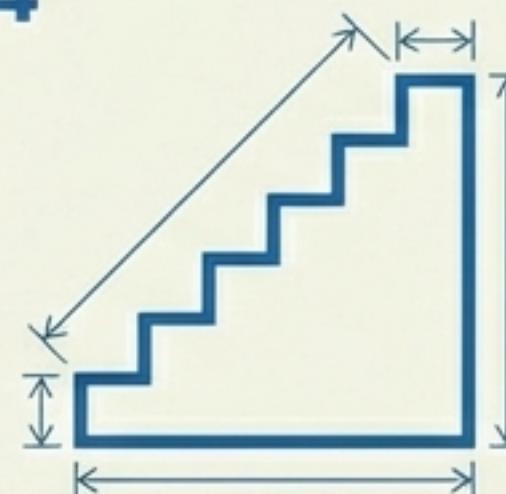
3



Evitar Duplicación (DRY)

Si copias y pegas, es
mala señal. Extraer
lógica para reutilizar.

4



Pasos Pequeños

Cambios atómicos y
seguros. Verificar
constantemente.

La Regla de Oro: Sin Pruebas no hay Refactorización



ADVERTENCIA

Refactorizar implica riesgo. La única red de seguridad son las pruebas (unitarias/integración) o casos manuales repetibles. Si no hay pruebas, no toques zonas críticas sin añadirlas antes.

Limitaciones y Riesgos Reales



Riesgos

- Firmas Rotas:** Cambiar parámetros sin actualizar todas las referencias.
- Código Legacy:** Refactorizar código heredado sin tests es peligroso.
- Pérdida de Foco:** Mezclar refactorización con cambios de funcionalidad.



Limitaciones Prácticas

- Tiempo/Presupuesto:** No es viable reescribir todo el sistema.
- Dependencias:** Cuidado con APIs externas o frameworks rígidos.

A veces "funciona" es suficiente si el riesgo de romperlo supera el beneficio de limpiarlo.

Toolkit A: Claridad e Intención

1. Rename (Renombrar)

Que el nombre explique la intención.

```
double f(double p) {  
    return p * 1.21; }
```



```
double calcularPrecioConIVA(double precio) { ... }
```



2. Extract Constant (Extraer Constante)

Eliminar "números mágicos".

```
if (password.length() < 8)
```

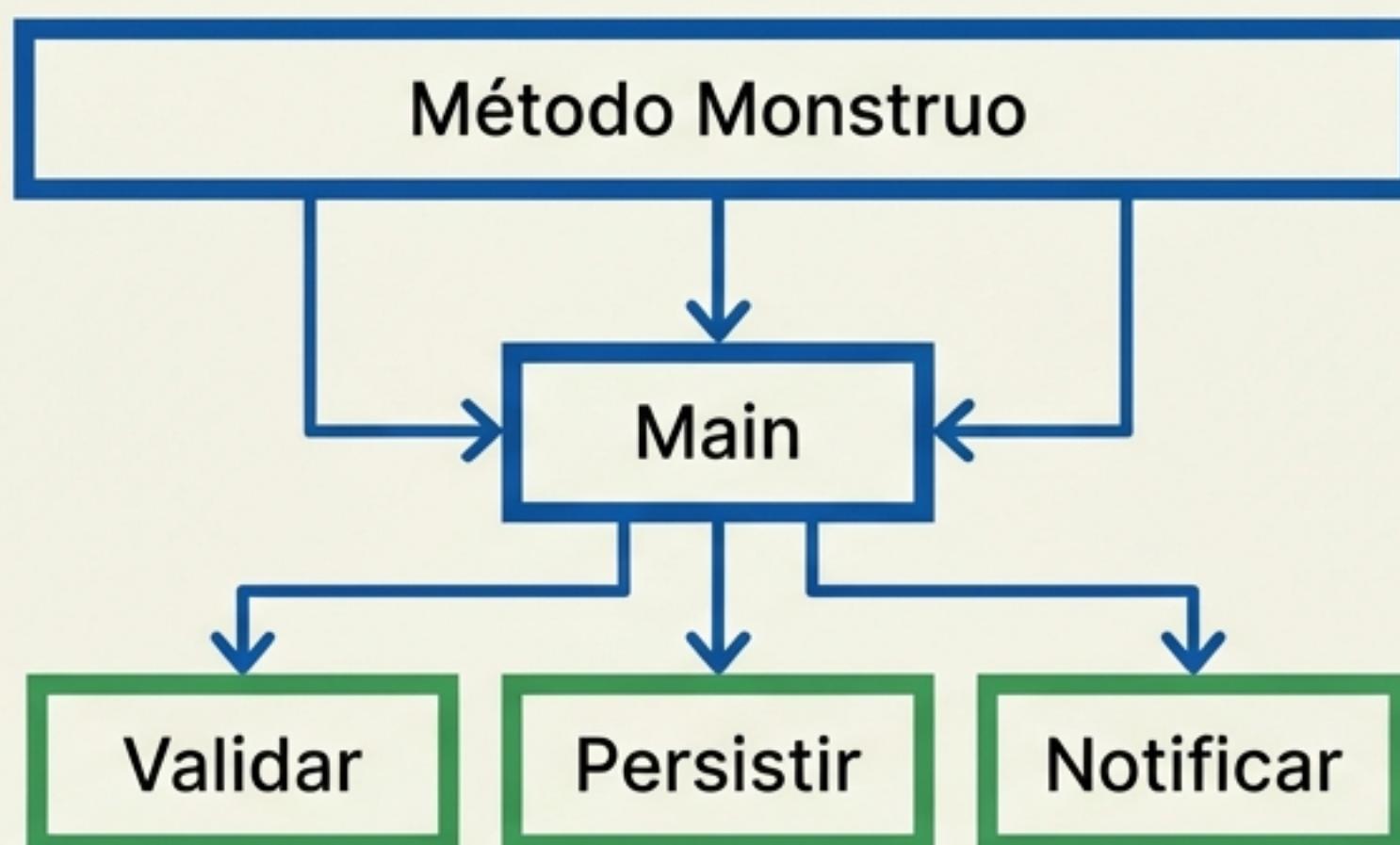


```
if (password.length() < MIN_PASSWORD_LENGTH)
```



Toolkit B: Estructura y Lógica

1. Extract Method (Extraer Método)



Dividir métodos largos en funciones específicas.

2. Replace Conditional

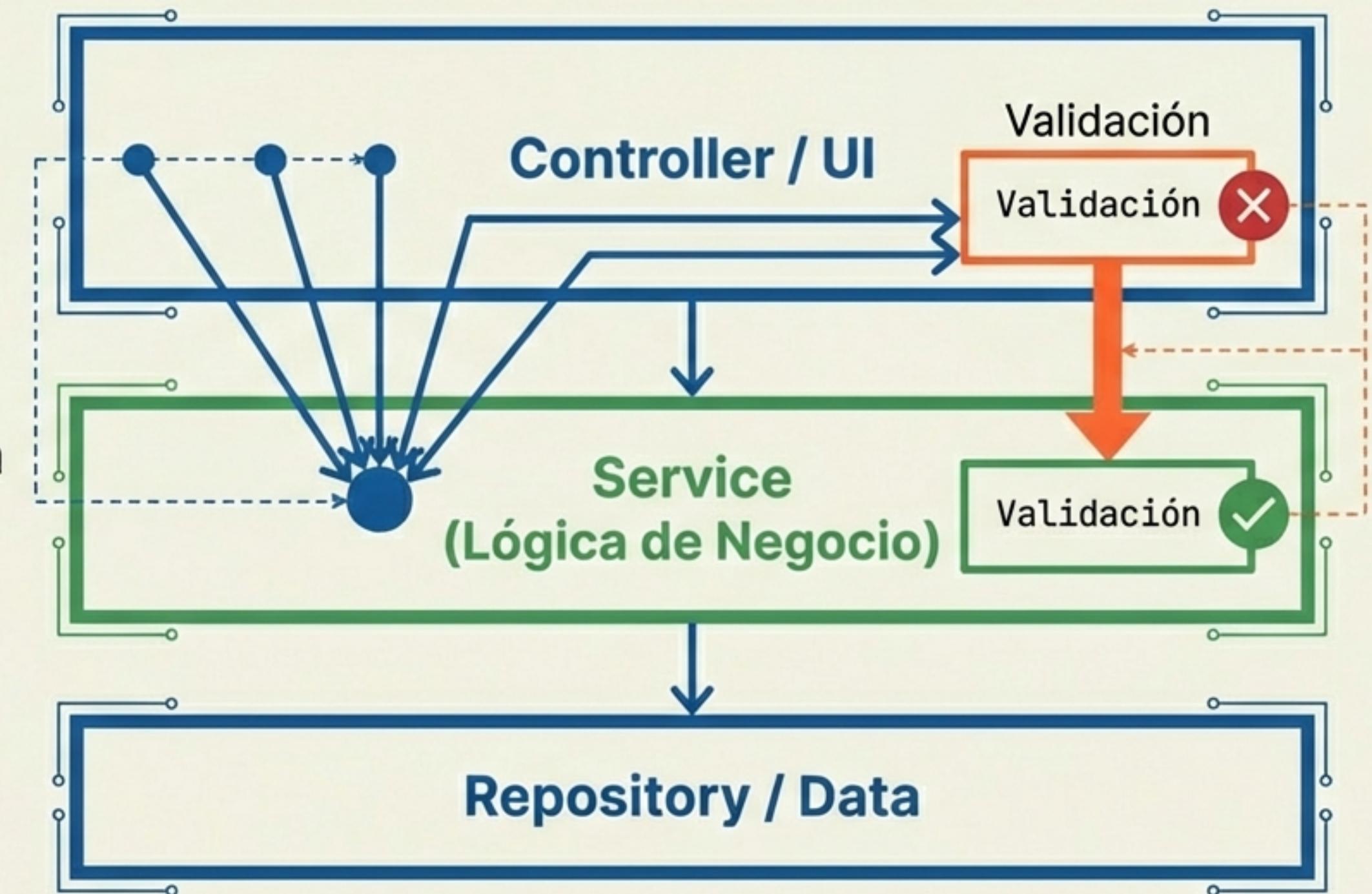
Simplificar condiciones complejas.

✗ `if (user != null && user.isActive() && role == "ADMIN")`

✓ `if (esAdminActivo(user))`

Toolkit C: Arquitectura y Reutilización

DRY:
Centralizar
lógica repetida



Move Method:
La lógica
pertenece al
Servicio, no al
Controlador.

Tu Copiloto: El IDE

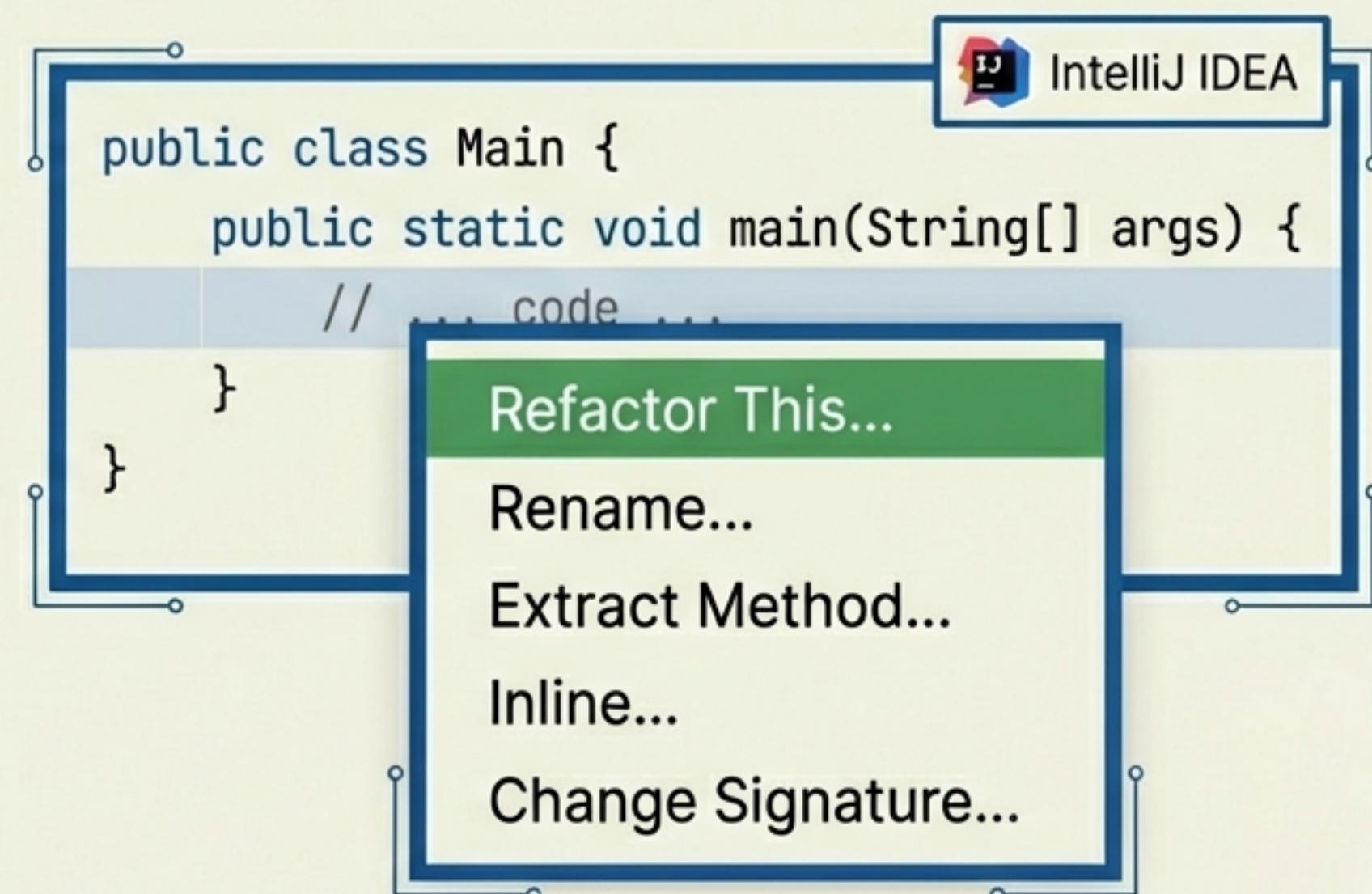
Automatización para reducir errores humanos.

Rename: Actualiza todas las referencias en el proyecto.

Extract Method: Crea el método y pasa parámetros automáticamente.

Inline: Invierte la extracción si es excesiva.

Change Signature: Añade/quita parámetros de forma segura.



Caso de Estudio: El "Checkout" de Pesadilla

```
public class CheckoutService {  
    public Object processCheckout(Order order, String userType, String couponCode) {  
        if (order != null) {  
            if (userType.equals("premium")) {  
                if (couponCode != null) {  
                    if (couponCode.equals("DAW10")) {  
                        order.applyDiscount(0.90); // 10% off  
                        if (order.isValid()) {  
                            if (paymentService.process(order)) {  
                                return "Order processed successfully!";  
                            } else {  
                                return "Payment failed.";  
                            } else {  
                                return "Invalid order items.";  
                            }  
                        } else {  
                            paymentService.process(order);  
                            if (order.isValid()) {  
                                return "Order processed successfully!";  
                            } else {  
                                return "Payment failed.";  
                            }  
                        }  
                    } else {  
                        return "Coupon code invalid or not applicable.";  
                    }  
                } else {  
                    return "Coupon code required for premium users.";  
                }  
            } else {  
                return "User type not recognized.";  
            }  
        } else {  
            return "Order cannot be null.";  
        }  
    }  
}
```

Magic Numbers/Strings

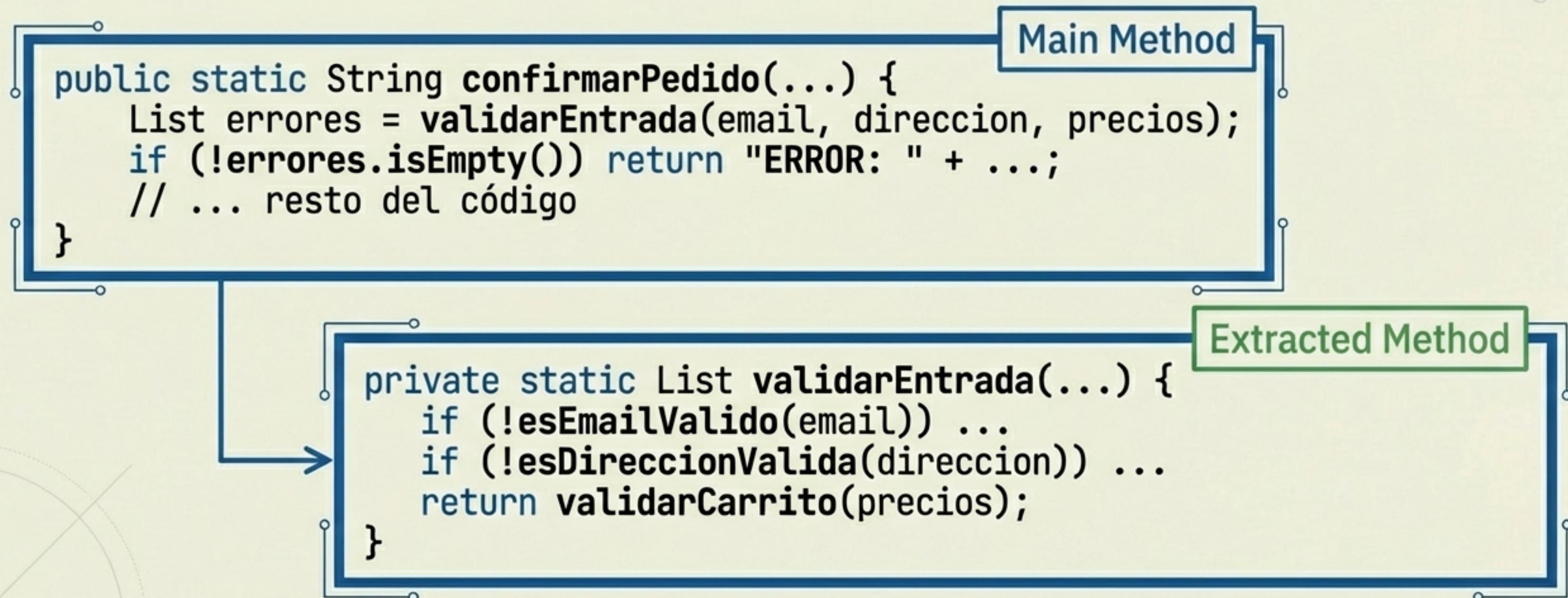
Anidación excesiva
(Arrow Code)

Mezclan lógica
y mensajes

Problemas detectados:
Responsabilidades
mezcladas, difícil de
probar, difícil de
extender.

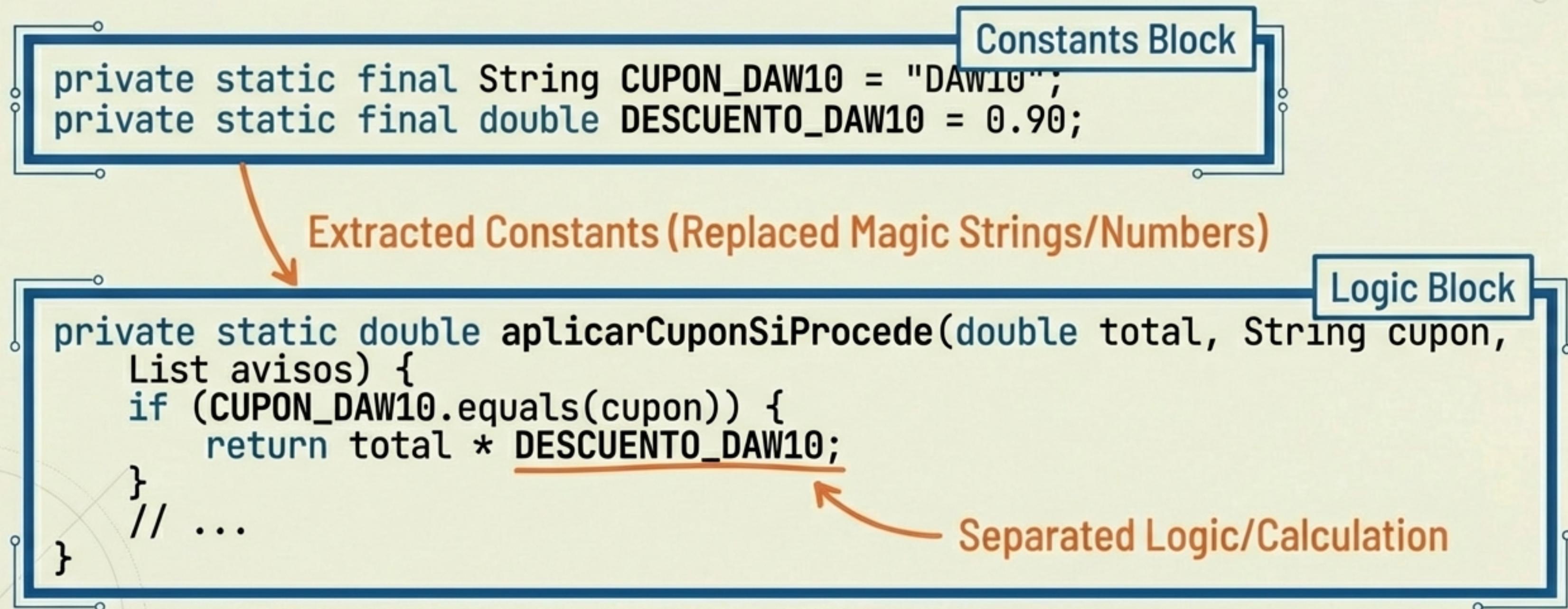
Paso 1: Limpieza y Validación

Aplicar `Extract Method` para aislar validaciones.



Paso 2: Lógica de Negocio y Constantes

Aplicar `Extract Constant` y separar cálculos.



El Resultado Final: Código que Habla

 **Legible:** Flujo lineal.

```
public class CheckoutServiceRefactor {
    private static final String CUPON_DAW10 = "DAW10";
    private static final double DESCUENTO_DAW10 = 0.90;

    public static String confirmarPedido(...) {
        List errores = validarEntrada(email, direccion, precios);
        if (!errores.isEmpty()) return "ERROR: " + ...;
        double total = calcularTotalConCupon(precios, cupon);
        procesarPago(total);
        return "Pedido confirmado";
    }

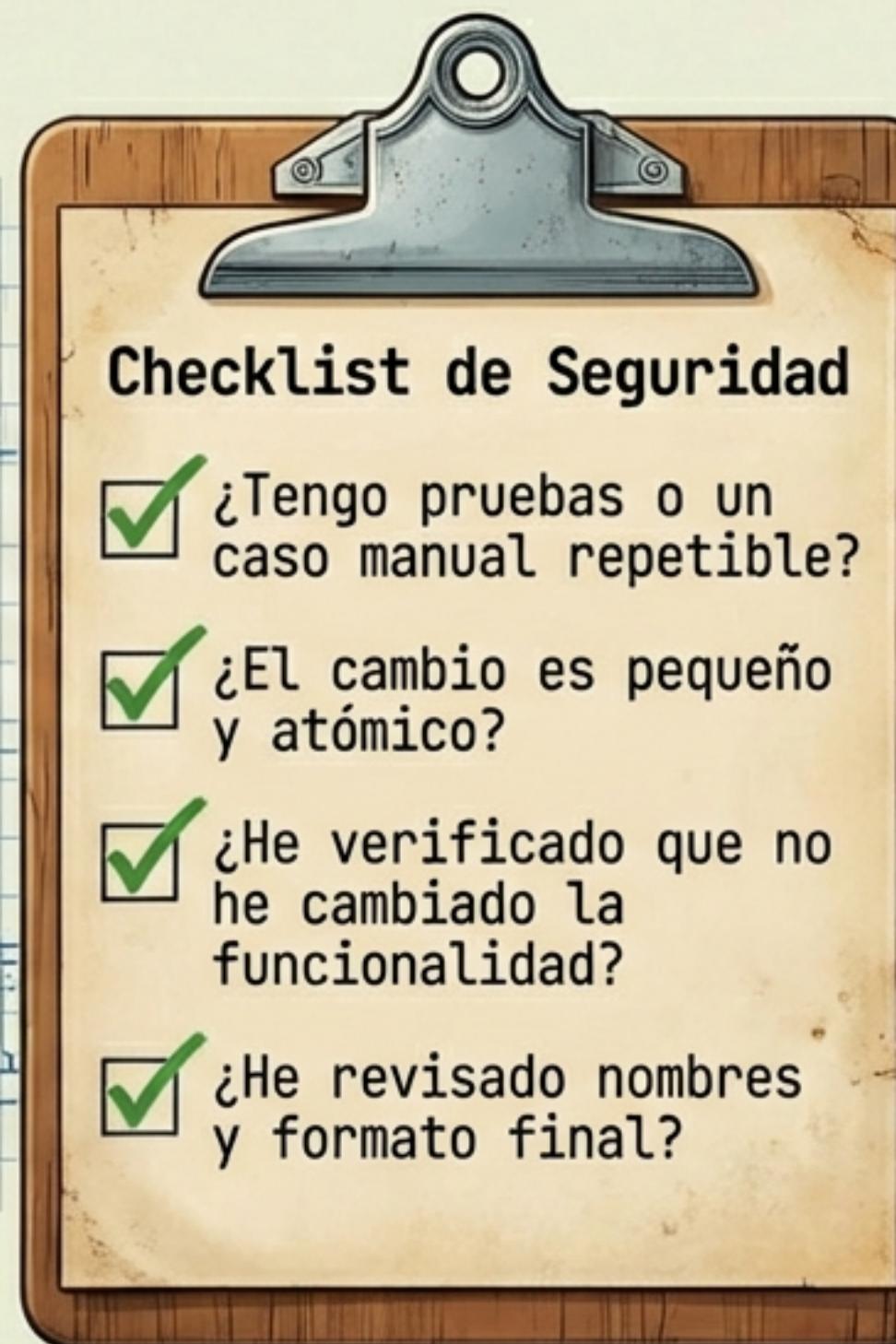
    private static List validarEntrada(...) { ... }
    private static boolean esEmailValido(...) { ... }
    private static boolean esDireccionValida(...) { ... }
    private static List validarCarrito(...) { ... }
    private static double calcularTotalConCupon(double total, String cupon) {
        if (CUPON_DAW10.equals(cupon)) {
            return total * DESCUENTO_DAW10;
        }
        return total;
    }
    private static void procesarPago(double total) { ... }
}
```

 **Seguro:** Uso de Constantes.

 **Responsabilidad Única:** Métodos separados.

 **Extensible:** Fácil añadir nuevos cupones.

Tu Turno: Checklist de Refactorización Segura



Práctica Rápida

Abre tu último proyecto.
Busca un “método monstruo”.
Aplica **Rename**, **Extract Method** y **Extract Constant**.

Ejecuta tus tests.

“Deja el código mejor de lo que lo encontraste.”