Ejercicio 1: Validador de entradas de usuario

Contexto:

Queremos asegurarnos de que un sistema solo acepte nombres válidos de usuario (sin números, ni símbolos, ni espacios).

Ejercicio 2: Calculadora de precios con descuentos

Contexto:

Queremos calcular el precio final de un producto aplicando un descuento.

Ejercicio 3: Sistema de autenticación simple

Contexto:

Queremos permitir que un usuario se registre y luego inicie sesión.

Ejercicio 4: Cálculo de impuestos por tramos

Contexto:

El sistema debe calcular impuestos según niveles de ingreso (por ejemplo, sin impuesto hasta $10,000, luego 10% hasta $20,000, etc.).

Ejercicio 5: Contador de palabras

Contexto:

Queremos analizar un texto y contar cuántas veces aparece cada palabra.

Fase	Qué haces	Qué obtienes
🔴 Red	Escribes una prueba que falla	Claridad sobre lo que el sistema debe hacer
🟢 Green	Escribes el código mínimo para pasar la prueba	Funcionalidad básica comprobada
🔵 Refactor	Limpias, mejoras, reorganizas el código	Código mantenible, sin romper nada

Ejercicio 6: Envios

Contexto:

Una aplicación maneja distintos métodos de envío: `EnvioExpress`, `EnvioNormal`, y `EnvioInternacional`.
El sistema actual usa un `if` gigante para calcular el costo del envío según el tipo.

Preguntas:

a) ¿Qué principio GRASP o SOLID aplicarías para evitar usar `if` según el tipo de pago?

Ejercicio 7: Generador de reportes

Contexto:

Un módulo genera reportes en diferentes formatos: `PDF`, `Excel`, `CSV`.
El método actual tiene un `switch(formato)` que llama a distintos bloques de código para cada tipo.

Preguntas:

a) ¿Qué principio aplicarías para mejorar este diseño?

Ejercicio 4: Sistema de facturación

Contexto:
La clase `Factura` calcula el total dependiendo del tipo de cliente (`Minorista`, `Mayorista`, `Exento`).
Hoy en día se hace con varios `if`.

Preguntas:
a) ¿Qué principio usarías para eliminar los `if` y separar responsabilidades?