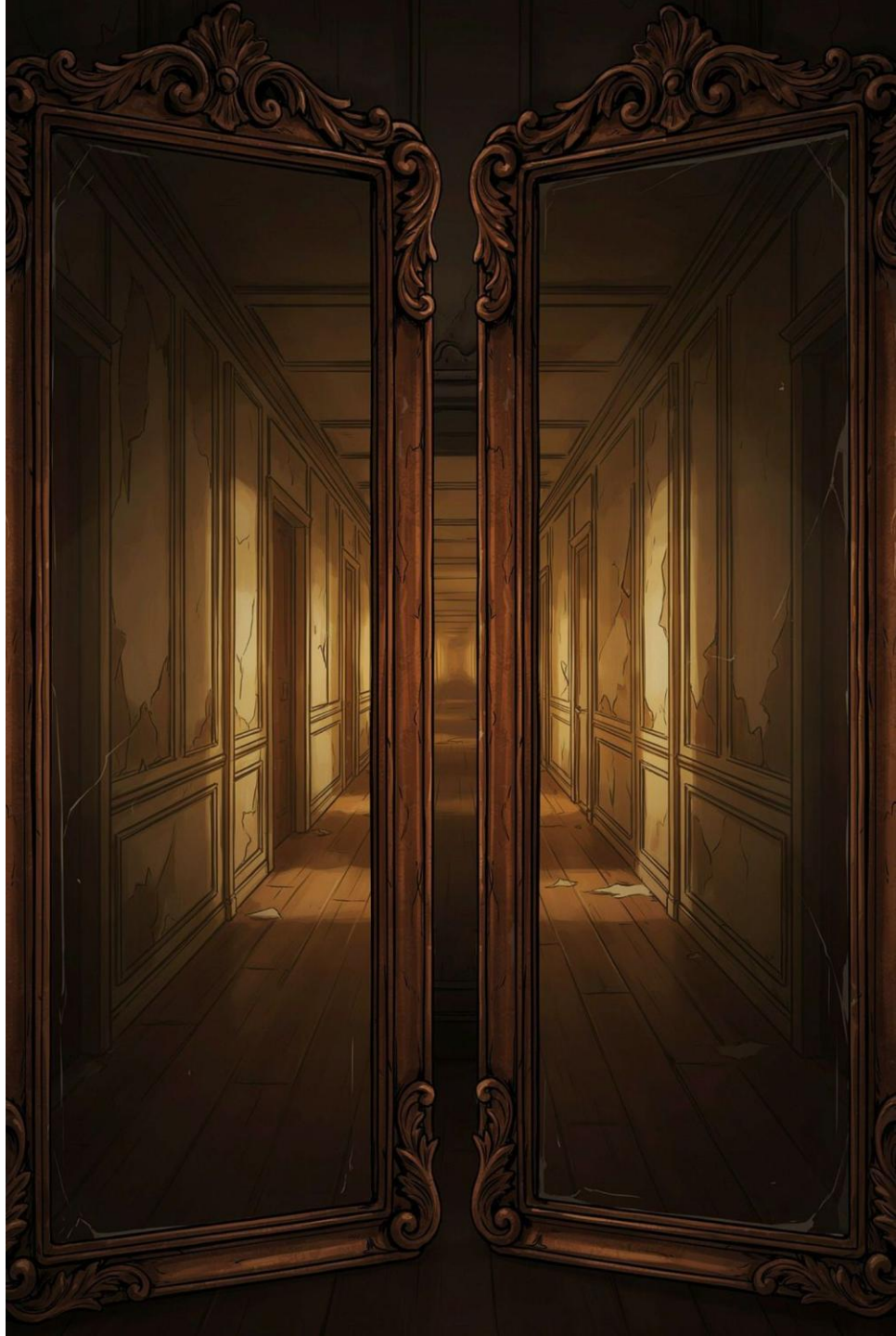
Recursividad y Verificación Orientada a Objetos

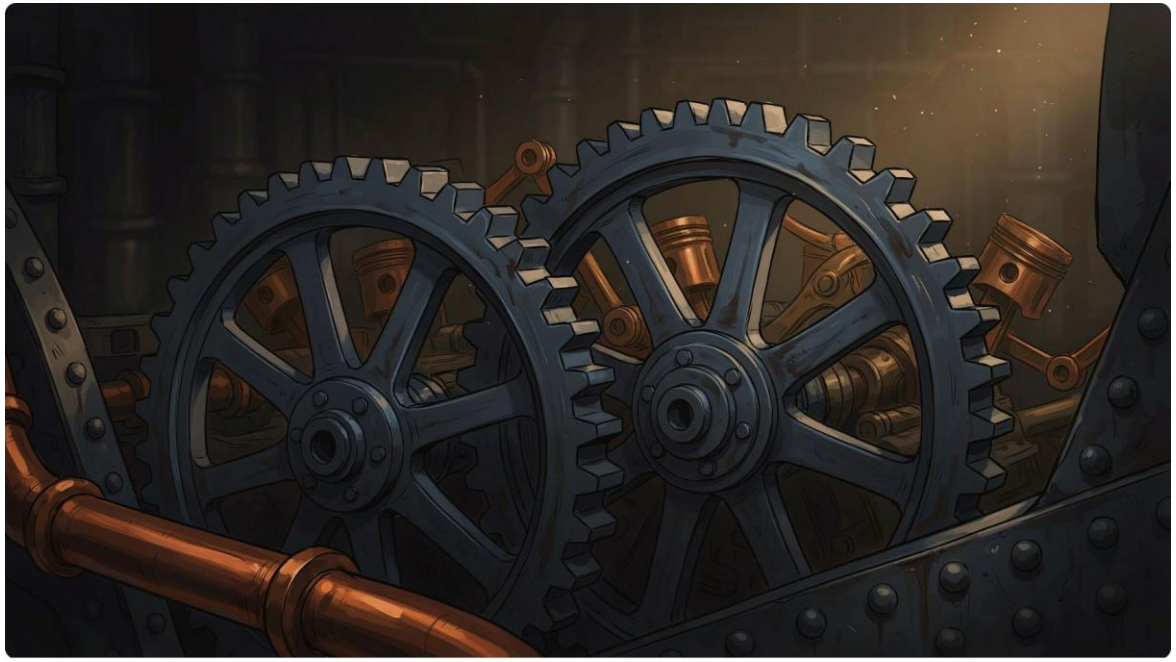
Fundamentos de Verificación

DESARROLLO FORMAL DE SOFTWARE

INGENIERÍA DE SISTEMAS · UNIVERSIDAD DEL PACÍFICO



Dos temas que se conectan



Recursividad

Una función se resuelve **llamándose a sí misma** con una instancia más pequeña del mismo problema.

Verificación OO

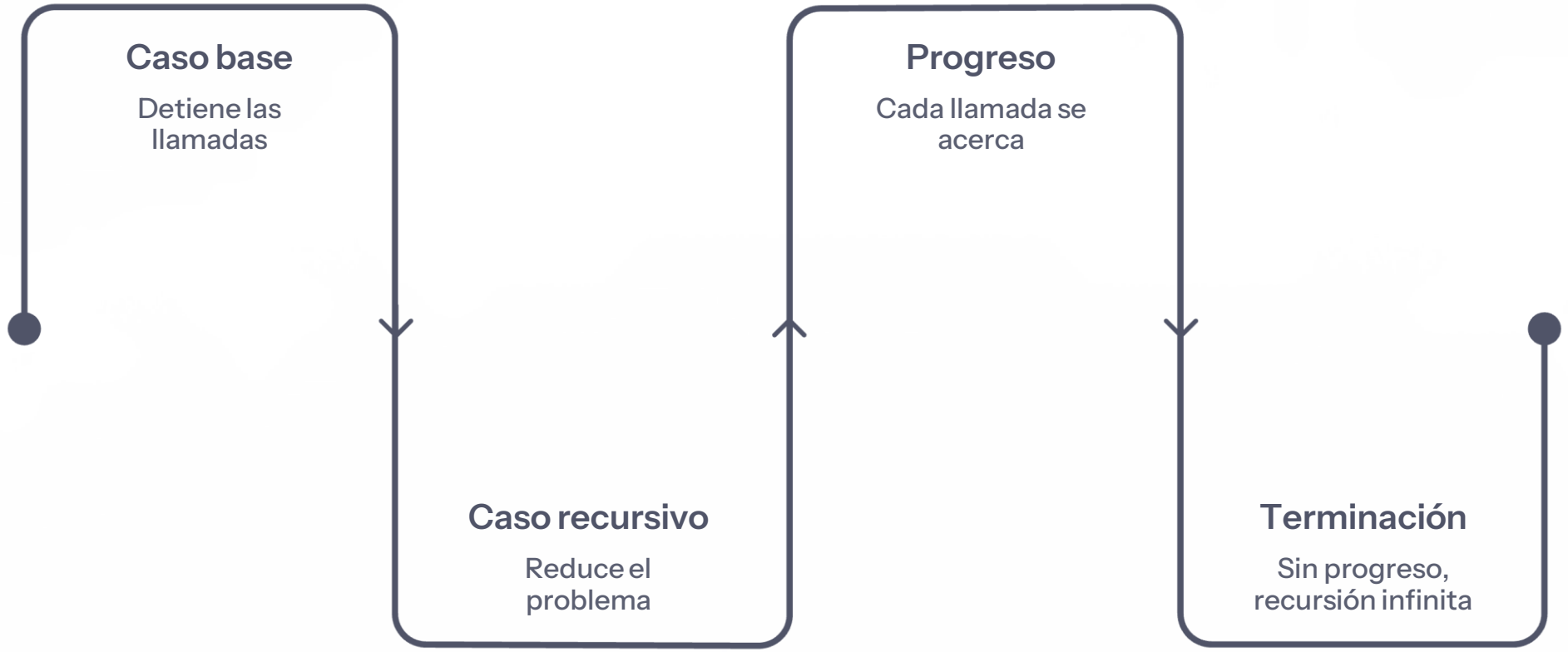
Análisis de **métodos, estado, contratos e invariantes** de una clase para garantizar su corrección.

Punto en común

Ambos requieren **condiciones explícitas** y razonamiento sistemático sobre los casos posibles.

by ykcaicedo

¿Cómo funciona la recursividad?



La clave está en garantizar que cada llamada recursiva acerque el problema al caso base; de lo contrario, la ejecución no termina.

by ykcaicedo

Ejemplo: factorial

Caso base

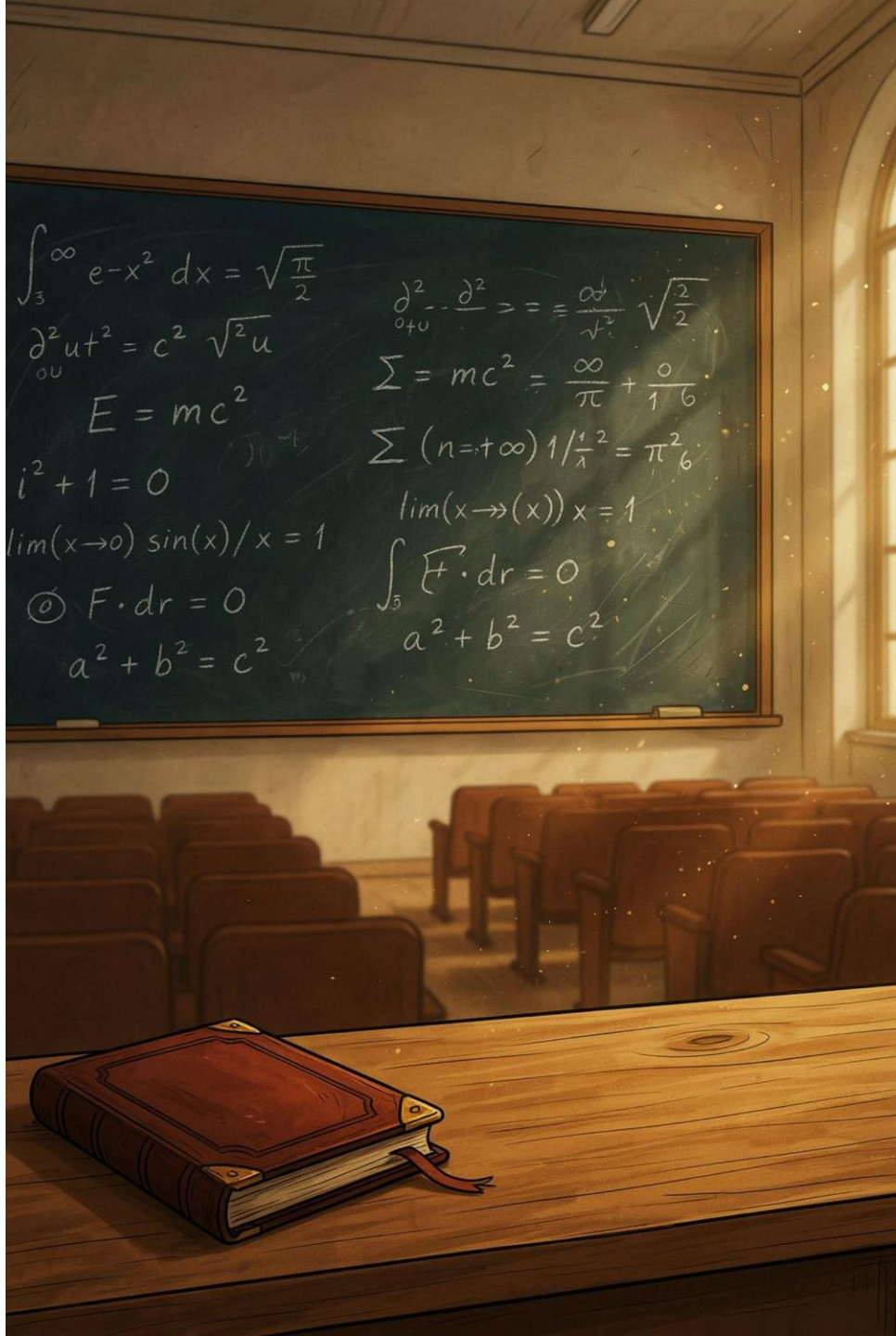
`factorial(0) = 1`

Caso recursivo

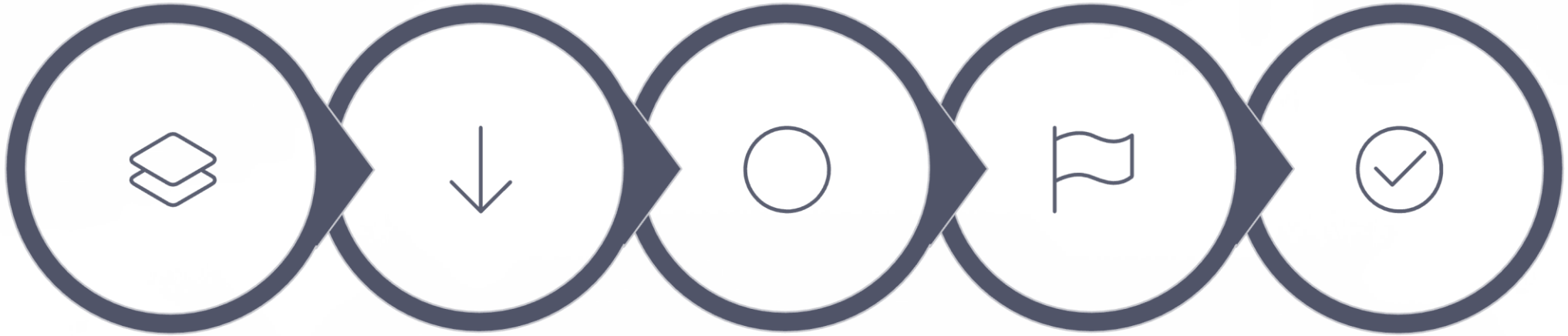
`factorial(n) = n * factorial(n-1)`

Contrato de la función

- **PRE:** `n >= 0`
- El caso base detiene la recursión en `n = 0`
- El caso recursivo reduce `n` en una unidad en cada llamada
- La PRE garantiza que siempre se alcanza el caso base



Seguimiento de `factorial(3)`



factorial(3) **factorial(2)** **factorial(1)** **factorial(0)** **Resultado 6**

Cada llamada apila un marco de ejecución hasta alcanzar el caso base; después, los resultados se combinan en sentido contrario hasta obtener **6**.

by ykcaicedo

Verificar una función recursiva

1

Caso base

El resultado satisface directamente la especificación POST sin llamadas adicionales.

2

Caso recursivo

Se reduce a una instancia menor que cumple la PRE; la combinación satisface la POST.

3

Terminación

El argumento progresa monotónicamente hacia el caso base en cada llamada.

by ykcaicedo

Errores frecuentes en recursividad

⚠ Caso base ausente

Sin caso base correcto, la recursión no termina y provoca *stack overflow*.

⚠ Sin reducción

Si la llamada recursiva no reduce el problema, nunca se alcanza el caso base.

⚠ Entradas fuera de PRE

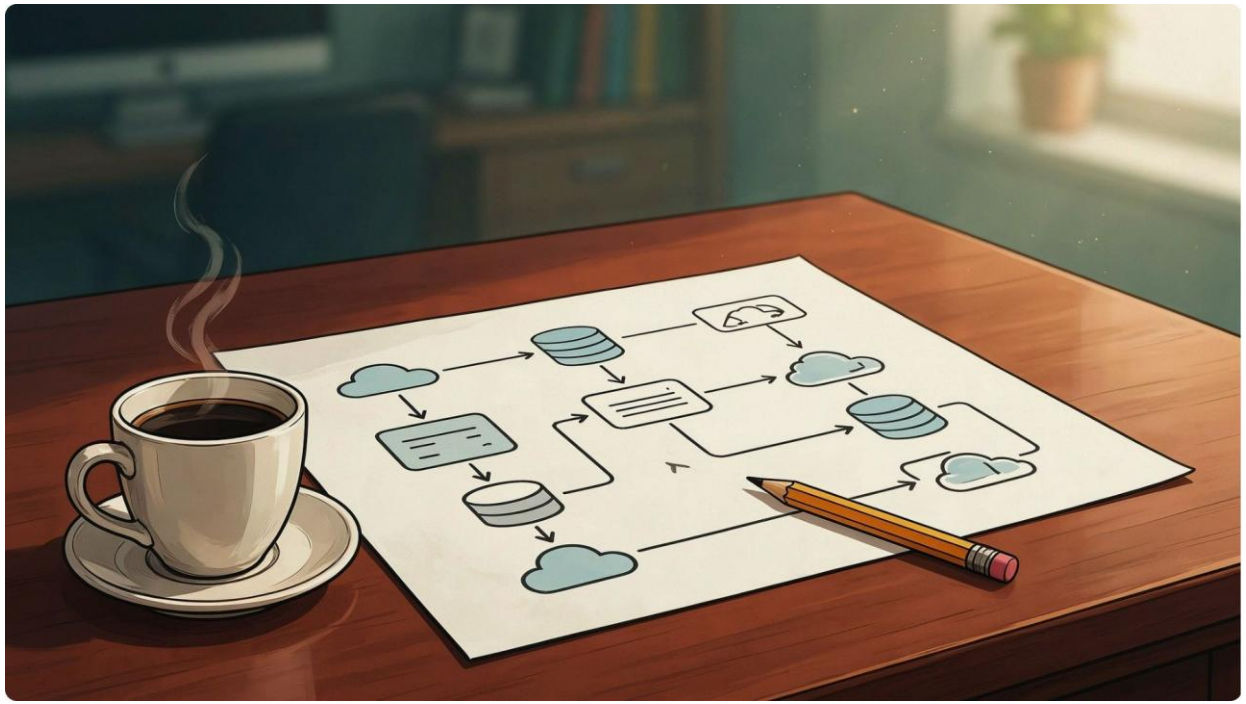
No considerar valores que violan la precondición puede generar comportamiento indefinido.

⚠ Combinación errónea

El resultado parcial de la llamada recursiva no se integra correctamente en el retorno.



Verificación orientada a objetos



Principios clave

Encapsulación

La clase agrupa **estado y operaciones**; el exterior solo accede a través de la interfaz pública.

Contratos PRE / POST

Cada método especifica qué debe cumplirse antes y qué garantiza al finalizar.

Invariante de clase

Propiedad que debe mantenerse en **todos los estados válidos** del objeto.

by ykcaicedo

Ejemplo: clase `Reserva`

Estado e invariante

Atributo: `cuposDisponibles`

Invariante: `cuposDisponibles >= 0`

Esta propiedad debe ser verdadera antes y después de cualquier operación pública.

Método `reservar(cantidad)`

PRE: `cantidad > 0` ∧ `cantidad <= cuposDisponibles`

Postcondición

`cuposDisponiblesnuevo = cuposDisponiblesanterior – cantidad`

Método y estado del objeto

Preservar el invariante

Cada método público debe dejar el objeto en un estado válido tras su ejecución.

Atomicidad de operaciones

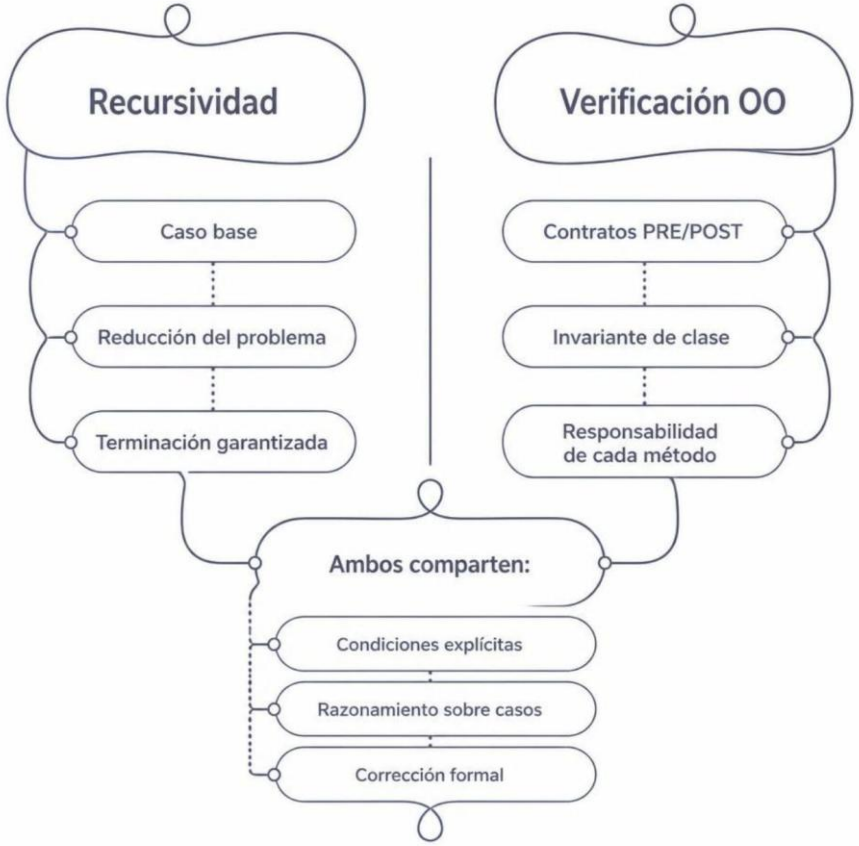
Si una reserva se rechaza, el estado **no debe quedar parcialmente modificado**.

Compatibilidad de contratos

Los contratos de los distintos métodos deben ser coherentes entre sí para que el objeto funcione correctamente.



Conectar recursividad y objetos



Técnicas complementarias

No son intercambiables: la recursividad resuelve problemas que se descomponen naturalmente en subproblemas, mientras que la verificación OO garantiza la consistencia del estado de un objeto a lo largo de su ciclo de vida.

Ambas técnicas comparten la necesidad de **condiciones explícitas** para razonar sobre el comportamiento del software.

Aplicación al proyecto

Recursividad en estructuras

Una operación recursiva resulta natural para procesar estructuras jerárquicas o anidadas si el proyecto las incorpora.

Contratos para reservas

Los contratos PRE/POST e invariantes preservan la validez de `cuposDisponibles` en todo momento.

Justificación del diseño

El diseño debe argumentar tanto la corrección del resultado de cada método como la validez del estado global del objeto.



Reto oral

Pregunta 1

¿Qué ocurre si `factorial(n)` se llama a sí misma con el mismo `n` sin reducirlo? ¿Termina alguna vez?

Pregunta 2

¿Qué propiedad debe mantener siempre `cuposDisponibles`, y cómo se denomina en verificación OO?

Pregunta 3

Si `reservar()` resta los cupos *antes* de validar la PRE, ¿qué riesgo concreto aparece sobre el estado del objeto?

by ykcaicedo

Cierre



> La corrección del software no se supone: se **demuestra** mediante condiciones, contratos e invariantes explícitos.

by ykcaicedo