

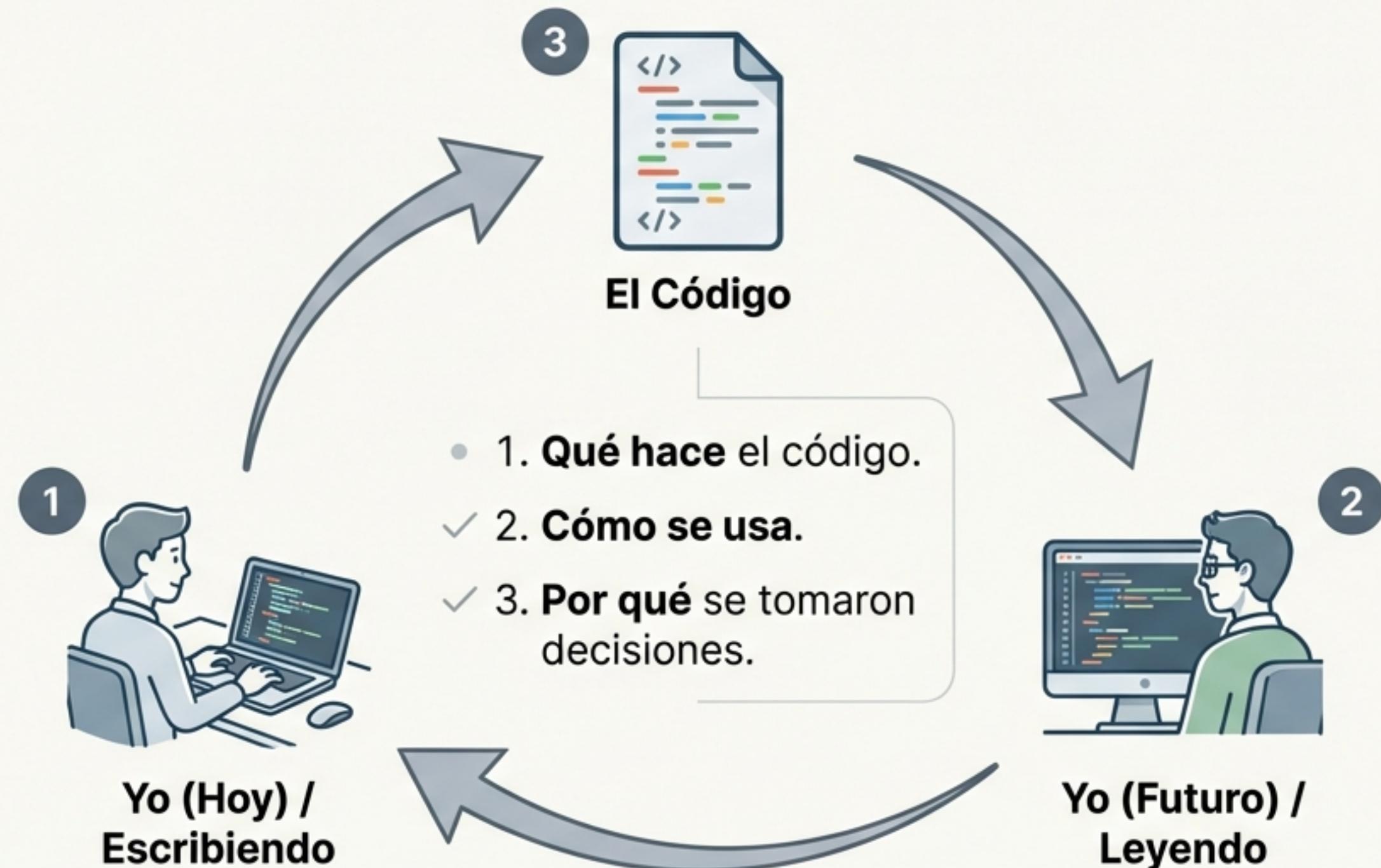
Documentación de Código en Java

Comentarios, Javadoc y Buenas Prácticas

MODULE: SOFTWARE CRAFTSMANSHIP 101

Idea clave: El código es comunicación

La documentación conecta el presente con el futuro.



Herramientas

Comentarios:
Breves y puntuales.

```
//
```

Javadoc:
Estándar para clases y métodos.

```
/**  
 */
```

🧠 ¿Por qué invertir tiempo en documentar?



Trabajo en equipo

Facilita que otros entiendan tu código rápido.



Ahorra tiempo

Menos dudas y menos preguntas de “¿esto por qué está así?”.



Evita errores

Las decisiones y reglas quedan claras, previniendo fallos futuros.



Aprendizaje

Explica reglas de negocio complejas y casos borde (edge cases).

Las dos herramientas: Comentarios vs. Javadoc

Comentarios Internos

```
// Notas internas para el desarrollador  
// Explican detalles de implementación
```

- **Breves y puntuales.**
- Uso de `///`.
- No generan documentación externa.

Javadoc Público

```
/**  
 * El estándar de Java para Clases y Métodos.  
 * Estructura pública (contrato).  
 */
```

- Contrato público.
- Uso de `/** ... */`.
- Permite **generar documentación HTML**.

✍ Comentarios: El arte de explicar el 'Por qué'

Los comentarios deben **aportar valor**, no substituir el código.

✓ Cuándo Sí (Aportan Valor)

Explican el **por qué** (decisión, regla, restricción) o aclaran casos especiales.

```
// Aplicamos IVA reducido por normativa (productos básicos)
double totalConIVA = total * IVA_REDUCIDO;

// Si el usuario está bloqueado, no permitimos login
if (user.isBlocked()) return LoginResult.BLOCKED;
```

✗ Cuándo NO (Sobran)

Dicen lo obvio o repiten el código línea por línea.

```
// Incrementa i
i++;
```

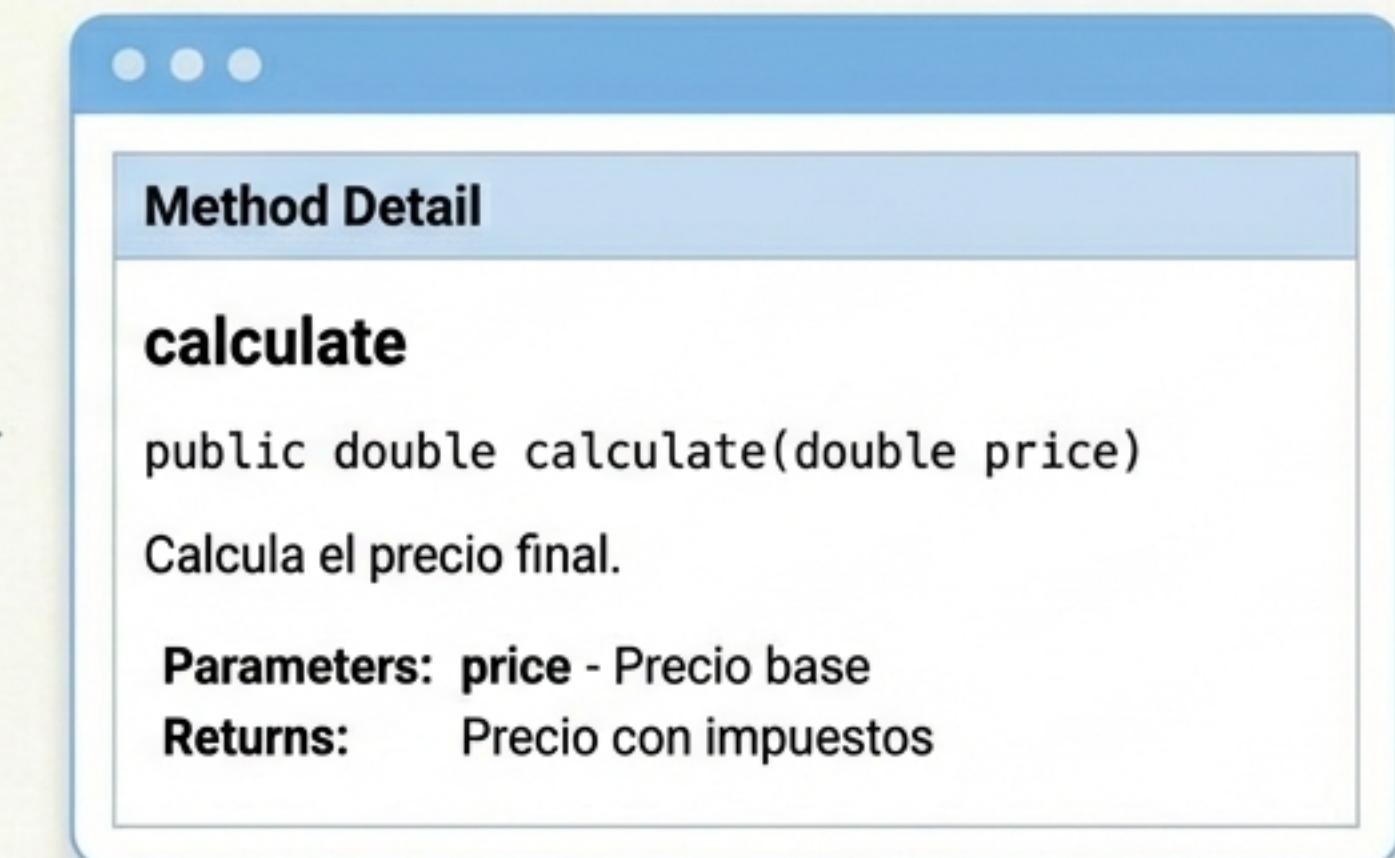
💡 Tip: Si el comentario solo repite el código, mejora el nombre de la variable.

Javadoc: El estándar de la industria

Javadoc es el formato oficial para documentar **clases y métodos**. Combina una descripción con etiquetas para generar HTML.

```
/**  
 * Calcula el precio final.  
 * @param price Precio base  
 * @return Precio con impuestos  
 */  
public double calculate(double price)
```

Generar →



Su gran ventaja es la **generación automática de HTML** para compartir con el equipo.

Etiquetas (Tags) Esenciales

Para Clases

- **@author:** Autor/a o responsable.
- **@author:** Autor/a o responsable.
- **@version:** Versión del componente.
- **@since:** Desde qué versión existe (útil en evolutivos).

Para Métodos

- **@param:** Describe cada parámetro de entrada.
- **@return:** Explica qué devuelve el método.
- **@throws:** Cuándo lanza una excepción.
- **@deprecated:** Si ya no se recomienda su uso.

Tip: No es obligatorio usar todas. Usa solo las que aporten información real.

■ Anatomía de una Clase (Ejemplo)

Descripción:
Define la responsabilidad (qué hace y qué no).

```
/**  
 * Servicio de cálculo de precios para pedidos.  
 * Aplica reglas simples de descuento y validaciones básicas.  
 *  
 * Uso típico: llamar desde el controlador para obtener el total.  
 *  
 * @author Aitor  
 * @version 1.0  
 * @since 2026-02  
 */  
public class PriceService { ... }
```

Contexto:
Explica cómo se usa en el proyecto.

Meta-datos: Autoría y versionado.

Anatomía de un Método (Ejemplo Recomendado)

```
/**  
 * Calcula el precio final de un pedido.  
 *  
 * @param unitPrice precio por unidad (debe ser >= 0)  
 * @param units cantidad de unidades (debe ser >= 0)  
 * @param premium indica si se aplica descuento premium  
 * @return precio final calculado (nunca negativo)  
 * @throws IllegalArgumentException si unitPrice o units son negativos  
 */  
public static double calculateTotal(...) {  
}
```

@param: Indican restricciones (ej: `>= 0`).

@return: Garantías del resultado (ej: "nunca negativo").

@throws: Explicta los errores posibles.

⚠ Cuándo NO usar Javadoc (o minimizarlo)



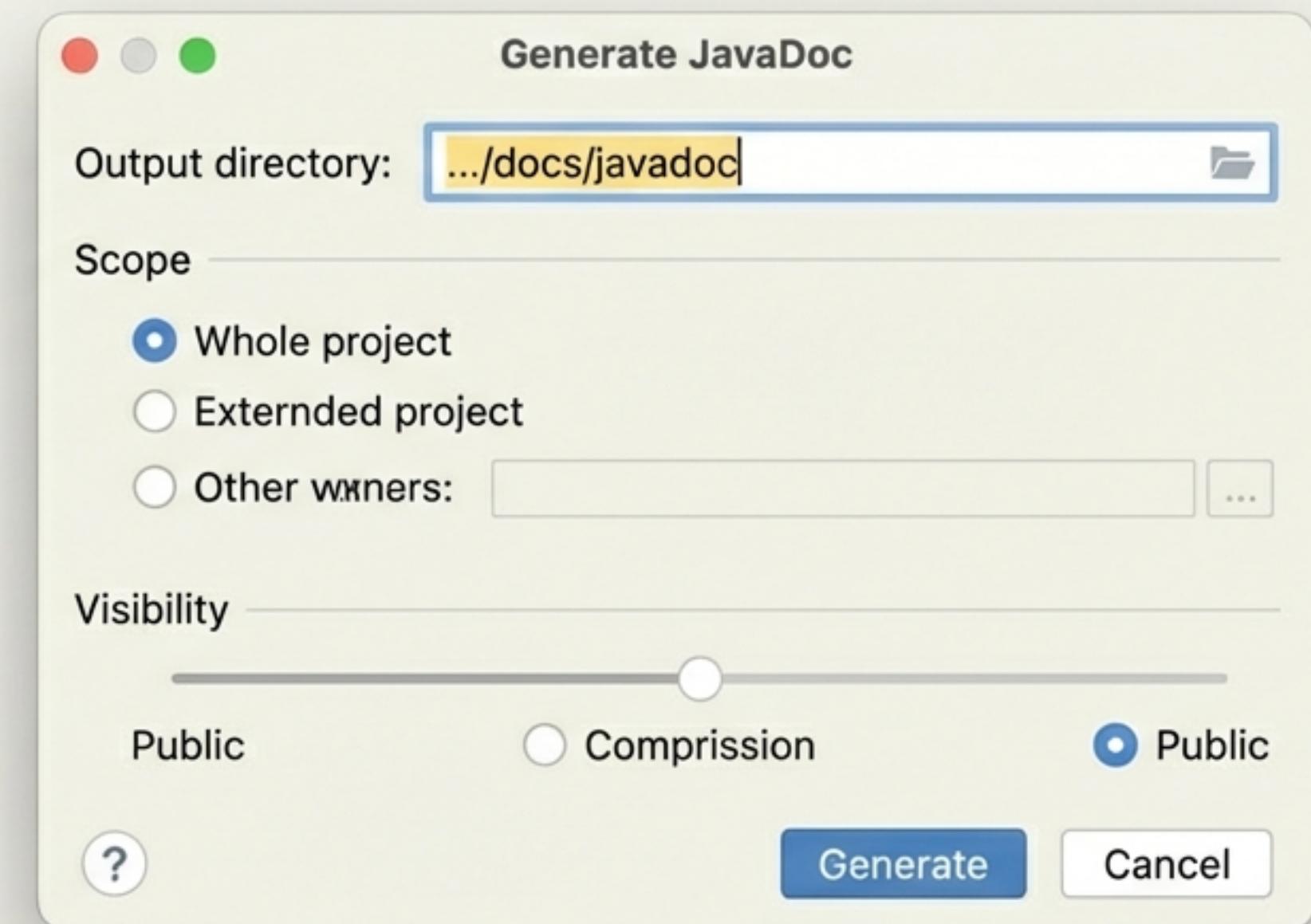
- **Métodos Privados Obvios:** "Si el nombre es claro, el Javadoc es ruido."
- **Código Volátil:** "Si cambia mucho, la documentación se desactualizará rápido."
- **Redundancia:** "Comentarios largos que solo repiten el código línea por línea."

“Mejor un código limpio sin Javadoc, que un Javadoc mentiroso o desactualizado.”

Generar HTML desde IntelliJ IDEA

1. Ir al menú: **Tools → Generate JavaDoc...**
2. Elegir **Output directory**.
3. Seleccionar **Scope** (**Whole Project/Module**).
4. Seleccionar **Visibility: Public** (Recomendado).
5. Pulsar **Generate**.

Obtendrás un **index.html** navegable con toda la estructura.





Automatización: Maven y Gradle

Para equipos profesionales y CI/CD.

Maven

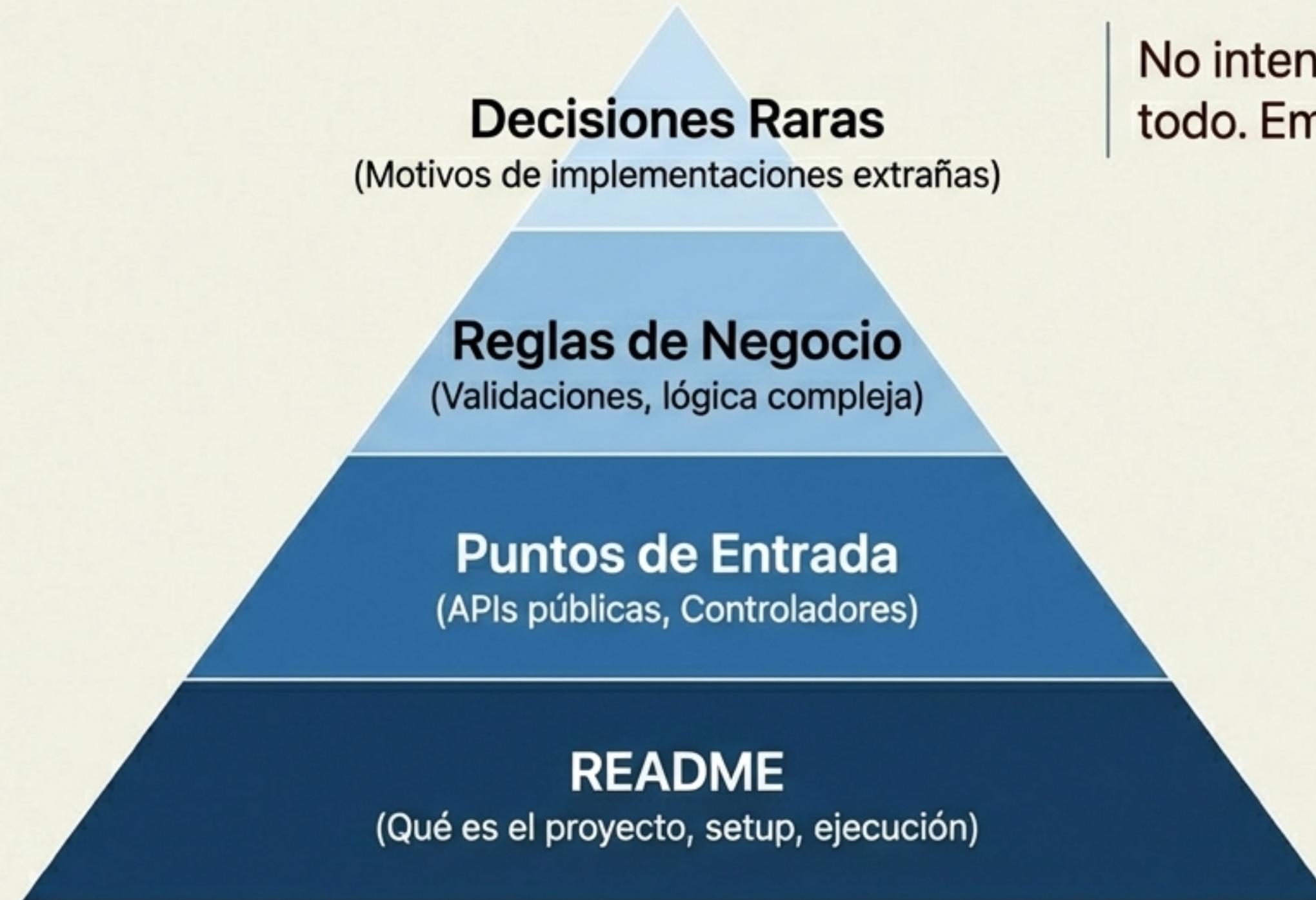
```
$ mvn javadoc:javadoc  
  
[INFO] Generating javadoc...  
[INFO] Output: target/site/apidocs/
```

Gradle

```
$ ./gradlew javadoc  
  
> Task :javadoc  
Output: build/docs/javadoc/
```

Reproducible por todo el equipo, independiente del IDE.

📁 Estrategia: ¿Qué documentar primero?



No intentes documentarlo todo. Empieza por la base.

✓ Checklist de Calidad





Práctica y Conclusión

Tu Misión

1. Elige un método con lógica de negocio.
2. Añade Javadoc definiendo Inputs (`@param`), Output (`@return`) y Excepciones.
3. Añade un comentario interno solo si hay un "por qué" complejo.

**El código te dice cómo se hace.
Los comentarios te dicen por qué.**