

**Diseño de Aplicaciones 2**  
Primer Obligatorio



Cristian Palma 208443 Federico Alonso 182999

**Evidencia de Clean Code y de la aplicación de TDD**

**Grupo N6A**

Repositorio: https://github.com/ORT-DA2/182999\_208443

**Índice**

[1. Clean Code y TDD 3](#_Toc84443560)

[1.1 Descripción de la estrategia de TDD seguida 3](#_Toc84443561)

[1.2 Informe de cobertura para todas las pruebas desarrolladas 3](#_Toc84443562)

[1.3 Evidencia de Clean Code 3](#_Toc84443563)

[1.4 Nombres según Clean Code 3](#_Toc84443564)

[1.4.1 Variables 3](#_Toc84443565)

[1.4.2 Métodos 3](#_Toc84443566)

[1.4.3 Clases 3](#_Toc84443567)

[1.5 Formato según Clean Code 3](#_Toc84443568)

[ // 3](#_Toc84443569)

[1.5.1 Variables y propiedades 4](#_Toc84443570)

[1.5.2 Métodos 4](#_Toc84443571)

[1.5.3 Clases 4](#_Toc84443572)

[1.6 Manejo de errores 4](#_Toc84443573)

[1.7 Ley de demeter 4](#_Toc84443574)

# Clean Code y TDD

## Descripción de la estrategia de TDD seguida

Para el desarrollo del obligatorio se hizo utilizando TDD (Test-Driven Development) siguiendo los siguientes pasos:

1. Escribir las pruebas.
2. Hacer el código mínimo para que la prueba sea fallida.
3. Hacer la implementación mínima para que las pruebas pasen.
4. Refactor para mejorar el código de las etapas anteriores.

Las pruebas antes mencionadas cumplen con el principio FIRST:

* FAST. Las pruebas deben ser rápidas de ejecutarse (aislarlas de factores que las puedan enlentecer como llamadas de red)
* INDEPENDENT. Las pruebas no deben depender una de la otra. Se debe poder correr las pruebas en cualquier orden y estas deberían pasar.
* REPEATABLE. Se deben poder repetir en cualquier ambiente.
* SELF-VALIDATING. Deben tener output booleano, las propias pruebas nos deben decir si pasan o no.
* TIMELY. Las pruebas se deben escribir antes que el código que las haga pasar.

## Informe de cobertura para todas las pruebas desarrolladas

//FOTO DE LA COBERTURA

## Evidencia de Clean Code

//Una clase debe tener una sola razón por la que cambiar SRP

## Nombres según Clean Code

* // Usar nombres que revelen intención
* //Evitar desinformar
* //Usar nombres pronunciables
* //Evitar prefijos
* //siempre nombres claros
* //Elegir sustantivos y no verbos para los nombres de las Clases
* //Para los métodos elegir verbos que indiquen lo que el método hace

### Variables

### Métodos

### Clases

* 1. Formato según Clean Code
* //Leer de arriba hacia abajo
* //Vamos a preferir siempre tener funciones que no tengan ningún argumento, pero en el caso de haber lo “normal” sería tolerar hasta 2
* //NO pasar flags (booleanos) en parámetros
* //El mejor comentario es el que no existe
* //La consistencia es lo más importante
* //Metáfora del diario
* //Dejar que el código respire
* //Indentar consistentemente
* //Agrupar correctamente

### Variables y propiedades

### Métodos

### Clases

## Manejo de errores

// Favorecer el uso de excepciones en lugar de retornar códigos de error, como enteros o enumerados.

## Ley de demeter

//Es una heurística que nos indica que una clase o módulo no debe conocer los secretos de los objetos que manipula.//La ley de demeter limita el número de dependencias que establezco con otros objetos.