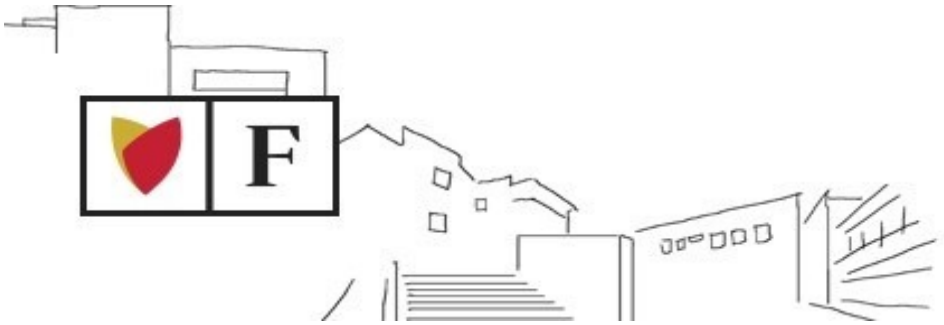

SOLID

2º DAW - Desarrollo Web en Entorno Servidor

Que es SOLID?

- **Conjunto de principios aplicables a la Programación Orientada a Objetos**
  - ayudan a escribir software de calidad
  - mejoran la cohesión disminuyendo el acoplamiento
  - crean código que más fácil de leer, testear y mantener
- **Son buenas prácticas, no leyes ni reglas.**
  - a veces es *imposible cumplir* todos los principios *a la vez*
- **Si no se emplean correctamente**
  - aumentar la complejidad del código
  - Generan código ambiguo y/o confuso.

Principios SOLID

- **Principio de Responsabilidad Única**
  (Single Responsibility Principle)
- **Principio Open/Closed**
  (Open/Closed Principle)
- **Principio de Sustitución de Liskov**
  (Liskov Substitution Principle)
- **Principio de Segregación de Interfaces**
  (Interface Segregation Principle)
- **Principio de Inversión de Dependencias**
  (Dependency Inversion Principle).

Principio de Responsabilidad Única

- **Una clase debe tener una única razón para cambiar = una clase hace una sola ‘’cosa’’**
- **Sospecha de violación del principio**
  - En una misma clase están involucradas dos o más capas de la arquitectura
  - Demasiados métodos públicos
  - Cada método usa sus propios campos
  - Cuesta mucho hacer test unitarios
  - La clase se ve afectada por ‘’otros cambios’’ demasiado a menudo
  - El número de líneas es elevado

Principio Open/Closed

- Una una entidad de software debería estar abierta a extensión pero cerrada a modificación
- se suele resolver utilizando polimorfismo
- Para detectar la violación:
  - Fijarse en las clases que modificamos a menudo
- esta complejidad no siempre compensa y sólo será aplicable si realmente es necesario.
- Un código 100% Open/Closed es prácticamente imposible

Principio Sustitución de Liskov

- **Si usamos una clase, y esta clase es extendida, tenemos que poder utilizar cualquiera de las clases hijas y que el programa siga siendo válido.**
- **Detectar la violación de este principio:**
  - Si un método sobrescrito no hace nada o lanza una excepción.
  - Si los tests de la clase padre no funcionan para la hija
- **Se soluciona mediante:**
  - Correcto uso de la herencia.
  - Composición.

Principio de Segregación de Interfaces

- **Ninguna clase debería depender de métodos que no usa (fat interfaces)**
- **Detectar la violación de este principio:**
  - uno o varios de los métodos no tienen sentido
  - o hace falta dejarlos vacíos o lanzar excepciones
- **Se soluciona mediante:**
  - Divísión en varias interfaces que definan comportamientos más específicos
  - El patrón de diseño *"Adapter"* = permite convertir unas interfaces en otras

Principio de Inversión de Dependencias

- **Definición:**
  a) Las clases de alto nivel no deberían depender de las clases de bajo nivel. Ambas deberían depender de las abstracciones.
  b) Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones
- **Detectar la violación de este principio:**
  - cualquier instanciación de clases complejas
- **Se soluciona mediante:**
  - paso de dependencias por constructor y setters
  - Inyección de dependencias