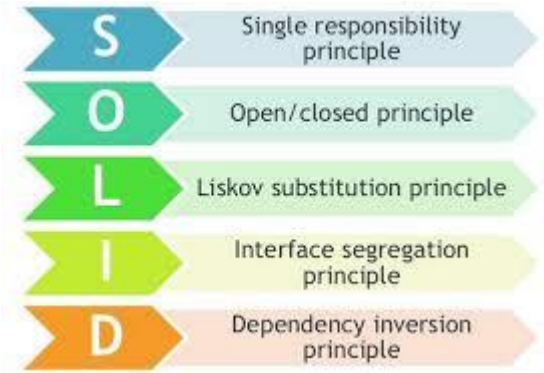
PRINCIPIOS SOLID DE DISEÑO

Ing. Luciano Straccia

UTN.BA
DPTO. INGENIERÍA EN SISTEMAS DE INFORMACIÓN
CÁTEDRA DISEÑO DE SISTEMAS

Principios solid

- Permiten mejorar el diseño
- SOLID =
  - Single Responsability
  - Open Closed
  - Liskov Substitution
  - Interface Segregation
  - Dependency Inversion



PRINCIPIOS SOLID

- Video sugerido
  - 0:00 a 6:39: SOLID
  - 6:40 a 19:15: SRP
  - 19:16 a 28:18: OCP
  - 28:19 a 39:39: LSP
  - 39:40 a 44:52: ISP
  - 44:53 al final: DIP

- Hacer click en la imagen o buscar en el canal de YouTube de la materia

Single responsability

- Single Responsability Principle (SRP)
- Principio de responsabilidad única
- Una clase debe tener una única responsabilidad
- Se entiende por responsabilidad a un motivo de cambio

Single responsability

Single responsability

- Ejemplo

Nuestra casa posee tres tipos de aves diferentes: gorriones, loros y pinguinos. Se debe permitir setear la altitud de vuelo para el caso de aves que vuelan)

El canto de los loros es grabado habitualmente por sus dueños en distintos formatos: MP3 y WMA.

Se espera llevar registro del tiempo total que las aves que vuelan han permanecido volando

- Si el AVE y el VUELO son incluidos en la misma clase, en lugar de dos clases distintas, se estaría infringiendo este principio

Single responsability

- Ejemplo

Pensemos en un módulo que compila e imprime un reporte.

Este módulo puede cambiarse por dos razones: primero, puede cambiar el contenido del reporte, y segundo, puede cambiar el formato del reporte

Estos son cambios muy distintos: uno de contenido y otro de formato

Estos aspectos son dos problemas distintos y deben existir en clases separadas

SINGLE RESPONSABILITY

- Ejemplo Proyecto EP.SOLID
- Violaciones:
  - 1 - Validación de email y envío de email
  - 2 - Validación de CPF
  - 3 - Conexión a base de datos
  - 4 - Adicionar cliente (sus acciones) junto con el tratamiento de sus atributos
- Clases nuevas:
  - 1 - Clase con responsabilidad de validación y envío de email (EmailServices)
  - 2 - Clase con responsabilidad de validación de CPF (CPFServices)
  - 3 - Clase con responsabilidad de persistir en base de datos (ClienteRepository)
  - 4 - Por un lado el tratamiento de sus atributos (Cliente) y por el otro su comportamiento (ClienteService)

Visual Studio

GitHub

Open/closed

- Open Closed Principle (OCP)
- Las entidades de software deben estar abiertas para ser extendidas, cerradas para ser modificadas

Open/closed

- Ejemplo

Nuestra casa posee tres tipos de aves diferentes: gorriones, loros y pinguinos. Se debe permitir setear la altitud de vuelo para el caso de aves que vuelan)

El canto de los loros es grabado habitualmente por sus dueños en distintos formatos: MP3 y WMA.

Se espera llevar registro del tiempo total que las aves que vuelan han permanecido volando

- Si hacemos una clase Audio que tiene los métodos “record_mp3” y “record_wma”, cada nuevo formato requiere modificar la clase. Mejor hacer una clase con cada formato (clase MP3, clase WMA, ambas subclases de Audio)

Open/closed

- Ejemplo Proyecto EP.SOLID
- Violaciones:
  - Unica clase con la implementación del método Debitar() independientemente del tipo de cuenta
- Clases nuevas:
  - Una clase por tipo de cuenta

Visual Studio

GitHub

Liskov substitution

- Liskov Substitution Principle (LSP)
- Si S es un subtipo de T, entonces los objetos de tipo T pueden ser reemplazados por objetos de tipo S sin alterar ninguna de las propiedades deseables del programa.

Liskov substitution

- Si nada como un pato, vuela como un pato, pero necesita batería probablemente haya un problema de abstracción

Liskov substitution

- Ejemplo

Nuestra casa posee tres tipos de aves diferentes: gorriones, loros y pinguinos. Se debe permitir setear la altitud de vuelo para el caso de aves que vuelan)

El canto de los loros es grabado habitualmente por sus dueños en distintos formatos: MP3 y WMA.

Se espera llevar registro del tiempo total que las aves que vuelan han permanecido volando

- si tenemos una clase BIRD con un método setAltitude y tenemos una clase PENGUIN que hereda, pero ¡el pingüino no vuela! No es conveniente este tipo de cosas

Liskov substitution

- Ejemplo Proyecto EP.SOLID

- Violaciones:
  - El cuadrado si bien tiene características comunes a los rectángulos, podría tener características o comportamientos diferentes (por ejemplo, la forma de calcular algún atributo)

- Solución:
  - Clase paralelogramo y sus subclases rectángulo y cuadrado

Visual Studio

GitHub

Interface segregation

- Interface Segregation Principle (ISP)
- Los clientes no deben ser forzados a depender de métodos que no usan
- El principio separa las interfaces que son muy grandes en interfaces más pequeñas y específicas, de manera que los clientes sólo tengan que conocer los métodos que les interesan.

Interface segregation

- Ejemplo Proyecto EP.SOLID

- Violaciones:
  - Suponiendo que el cadastro de cliente no necesite enviar email está obligado a implementar el método de envío de email (aunque sin funcionalidad, poniendo por ejemplo comentarios)

- Solución:
  - Armar diversas interfaces para cada cadastro

Visual Studio

GitHub

Dependency inversion

- Dependency Inversion Principle (DIP)
- Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones.
- Las abstracciones no deben depender de detalles. Los detalles deben depender de abstracciones.
- El objetivo es desacoplar a los componentes de alto nivel de los componentes de bajo nivel, de manera que sea posible la reutilización de los componentes de alto nivel al usar componentes de bajo nivel distintos

Dependency inversion

- Ejemplo Proyecto EP.SOLID
- Violaciones:
  - Hay alto acoplamiento.
  - Por ejemplo en clase Cliente utilizo una instancia de EmailServices y de CPFServices
  - Por ejemplo en clase ClienteServices construyo una instancia de ClienteRepository
- Solución:
  - Uso de Interfaces (como en este ejemplo)
  - Uso de parámetros (http://www.michael-pratt.com/blog/13/Patrones-de-Diseno-Inyeccion-de-Dependencias/)

Visual Studio

GitHub

FIN