Franco Barlocco – Nicolas Lorenzo - LucasBenitez

UNIDAD TEMÁTICA 5 – Patrones de diseño– Trabajo de Aplicación 1

Para cada uno de los siguientes ejercicios, en equipo:

1. Determine que principio SOLID se está violando, agregando una justificación.
2. Agregue las clases, interfaces, métodos que considere necesarios para remediar la situación.

Ejercicio 1)

El principio que no se respeta en el ejemplo 1 es el de Liskov, dado que algunos objetos heredan funciones que no son posibles ejecutar

Para solucionar este problema deberíamos tener:

-Una clase animal con la función comer()
-Una clase animalNoVuela que hereda de animal
-Una clase animalVuela que hereda de animal y se extiende implementando el método volar().

Ejercicio 2)

El prinicpio que se esta rompiendo es Single-Responsibility principle debido a que la clase impresora tiene mas de una responsabilidad, en este caso, escanear e imprimir.

Para solucionar esto debemos tener:

-Una clase impresora que tenga como responsabilidad imprimir()
-Una clase escanearDocumento que tenga la responsabilidad de escanear mediante un método escanear()

Ejercicio 3)

El principio que se esta rompiendo es Single-Responsibility principle debido a que la clase Base de datos tiene mas de una responsabilidad, como ser Guardar() y enviarCorreo()

Para solucionar esto deberíamos tener:
-La clase base de datos con el metodo guardar()
-Una clase EnviarCorreo con el método enviar()

Ejercicio 4)

El principio que se esta rompiendo es Single-Responsibility principle debido a que la clase Robot tiene mas de una razón para cambiar, como ser si cambia la forma de cocinar o la forma de limpiar, la clase va a cambiar.
Para esto deberíamos refactorizarlo de la siguiente manera:
-Por cada funcionalidad del robot que sea diferente, crear una nueva clase e insertar todos los métodos relacionados a esa clase. De esta forma cada clase quedaría con una única responsabilidad.

Ejercicio 5)

El principio que se esta rompiendo es Single-Responsibility principle, la clase cliente no debe tener la responsabilidad de crear un pedido, sino que solamente debe representar a un usuario.
Para solucionar esto debemos crear una clase crearPedido() que se encargue de crear un pedido y la clase cliente solo debe tener los métodos relacionados al cliente, como su creación.

Ejercico 6)

En este caso no se respeta el principio de Liskov debido a que se hereda el método volar en un objeto que no puede volar, para solucionar esto, debemos
-Crear una clase pato con el metodo nadar() y graznar()
-Una clase patoVuela que hereda de pato, y tiene el método volar()
-Una clase patoNoVuela que hereda de pato

Ejercicio 7)

En este caso se rompe el principio de Interface Segregation Principle ya que en base a si definición los clientes no deben ser forzados a depender de tipos que no usan, por ejemplo, la clase ReadDatabase implementa la interfaz IDataBase y no debe tener la responsabilidad de implementar el método WriteData()
Para solucionar esto debemos crear dos interfaces, una que contenga el método write data y otra que tenga los métodos Connect() y Disconnect().
Luego desde la clase Database debemos implementar ambas interfaces y desde la clase ReadDatabase debemos implementar solo la interfaz que contiene los métodos Connect() y Disconnect().

Ejercicio 9)

En este ejercicio se rompe el principio SRP ya que la clase User no debe tener la responsabilidad de tener el método canEditPost(), para solucionarlo debemos

-Crear una clase User con el método isAdmin()
-Crear una clase post con el método autor()
-Crear una clase permisos con el método canEditPost()

Ejercicio 10)

El prinicpio que se rompe en este caso es el de OCP dado que la clase musicPlayer contiene la logia para reproducir en distintos reproductores, esto no permite que sea abierto a la extensión y cerrado a la modificación.
Para solucionar esto debemos crear una interfaz ImusicPlayer que tenga la firma play(nombreArchivo)
Luego esta interfaz se implementa en cada clase que contenga un reproductor diferente, por ejemplo reproductorMp3 : ImusicPlayer.