****

**UNIVERSIDAD PRIVADA DE TACNA**

**FACULTAD DE INGENIERIA**

**Escuela Profesional de Ingeniería de Sistemas**

**INFORME DE LABORATORIO**

**“Principios SOLID”**

Curso: Patrones de Software

Docente: Mag. Patrick Cuadros Quiroga

**Gallegos Laucata, Alvaro Jheol (2017057608)**

**Limache Durand, Rodrigo Jeral (2017059278)**

**Vilca Condori, Erlag Fernando (2019064024)**

**Tacna – Perú**

**2023**

INDICE

[**I.** **INFORMACIÓN GENERAL** 3](#_Toc133072742)

[- **Objetivos:** 3](#_Toc133072743)

[- **Equipos, materiales, programas y recursos utilizados:** 3](#_Toc133072744)

[**II.** **MARCO TEORICO** 3](#_Toc133072745)

[**III.** **PROCEDIMIENTO** 4](#_Toc133072746)

[CONCLUSIONES 4](#_Toc133072747)

[RECOMENDACIONES 4](#_Toc133072748)

**INFORME DE LABORATORIO**

**Principios SOLID**

1. **INFORMACIÓN GENERAL**

* **Objetivos:**

El objetivo principal de SOLID es mejorar la calidad del software mediante la creación de código limpio, fácil de entender, fácil de mantener y extensible. Los cinco principios de SOLID, que son:

* Principio de Responsabilidad Única (SRP)
* Principio de Abierto/Cerrado (OCP)
* Principio de Sustitución de Liskov (LSP)
* Principio de Segregación de Interfaces (ISP)
* Principio de Inversión de Dependencia (DIP)

cada uno tiene un objetivo específico:

* SRP: Asegurar que cada clase o método tenga una única responsabilidad, lo que facilita la comprensión, la prueba y el mantenimiento del código.
* OCP: Permitir que el software sea extensible sin modificar el código fuente original, lo que reduce el riesgo de errores y acelera el proceso de desarrollo.
* LSP: Garantizar que las subclases se puedan utilizar en lugar de sus superclases sin afectar el comportamiento del programa, lo que permite una mayor flexibilidad en la creación de software.
* ISP: Dividir las interfaces en interfaces más pequeñas y específicas para que los clientes solo implementen lo que necesiten, lo que reduce la complejidad y el acoplamiento del software.
* DIP: Invertir la dependencia entre módulos de software, de tal manera que los módulos de alto nivel no dependan de los módulos de bajo nivel. En cambio, ambos dependen de abstracciones, lo que permite una mayor flexibilidad y facilidad en la creación de software.

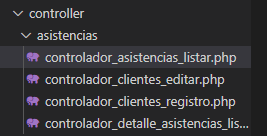
1. **MARCO TEORICO**

El marco teórico de SOLID se compone de cinco principios de diseño de software: SRP, OCP, LSP, ISP y DIP. Estos principios buscan crear un código más mantenible, escalable y fácil de extender.

* SRP se enfoca en que cada clase o método tenga una única responsabilidad para facilitar la comprensión y mantenimiento del código.
* OCP busca que el software sea extensible sin modificar el código fuente original.
* LSP garantiza que las subclases se puedan utilizar en lugar de sus superclases sin afectar el comportamiento del programa.
* ISP busca dividir las interfaces en interfaces más pequeñas y específicas para reducir la complejidad y el acoplamiento del software.
* DIP busca invertir la dependencia entre módulos de software para permitir una mayor flexibilidad y facilidad en la creación de software.

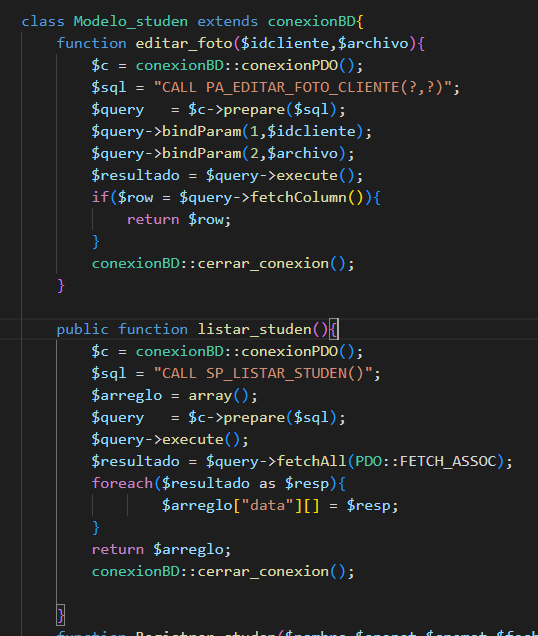
1. **PROCEDIMIENTO**
2. **Responsabilidad Única (SRP)**

En un controlador de Asistencias, se puede aplicar el principio de Responsabilidad Única (SRP) dividiendo las responsabilidades del controlador en diferentes métodos o clases, cada uno con una única responsabilidad.



1. **Principio Abierto/Cerrado (OCP):**

Sí, el código presentado cumple con el principio Abierto-Cerrado. En este ejemplo, la clase Modelo\_studen extiende la clase conexionBD, que encapsula la lógica de conexión a la base de datos. Además, cada uno de los métodos de la clase Modelo\_studen es responsable de una tarea específica y bien definida, y se puede extender la funcionalidad de la clase sin modificar su código original.



# CONCLUSIONES

En conclusión, el marco teórico de SOLID ofrece una guía para el diseño de software más mantenible, escalable y fácil de extender. Cada uno de los cinco principios tiene su propio objetivo específico, pero en conjunto, se busca minimizar la complejidad y el acoplamiento del software, permitiendo una mayor flexibilidad y facilitando la creación de software de alta calidad

La aplicación de los principios SOLID puede ayudar a los desarrolladores de software a crear aplicaciones más eficientes y fáciles de mantener. Además, el cumplimiento de estos principios puede mejorar la calidad del código y permitir su reutilización en proyectos futuros, lo que a su vez puede ahorrar tiempo y recursos en el desarrollo de software. En resumen, SOLID es un conjunto valioso de principios para cualquier desarrollador de software que busque crear código de alta calidad y fácil de mantener.

# RECOMENDACIONES

El estudiante podrá sugerir todas las recomendaciones y observaciones referidas al desarrollo de la práctica. Debiendo explicar las causas, si fuera el caso, de resultados obtenidos distintos a los esperados.