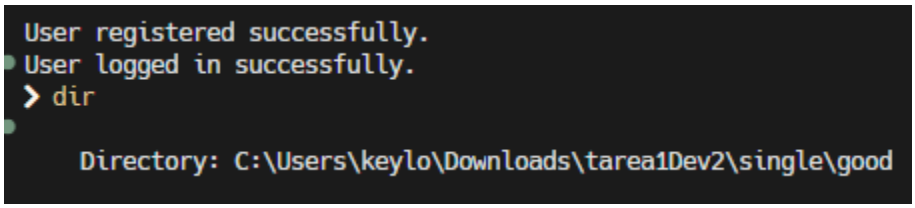
1. Principio de Responsabilidad Única (SRP)

- Código original: Había una clase que hacía muchas cosas al mismo tiempo.
- Problema identificado: Esto viola el SRP porque la clase tiene más de una responsabilidad.
- Refactorización aplicada: Se separaron las tareas en distintas clases.
- Justificación técnica: Así el código es más ordenado y fácil de mantener.

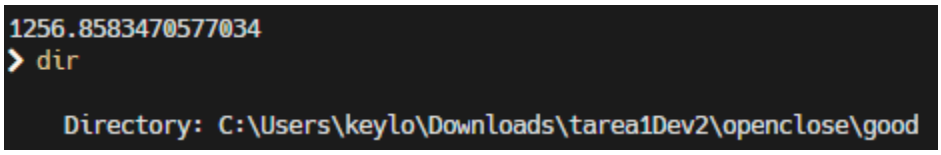
Evidencia:

- 

2. Principio Abierto/Cerrado (OCP)

- Código original: Se usaban condicionales para agregar nuevos comportamientos.
- Problema identificado: Cada cambio obligaba a modificar el código existente.
- Refactorización aplicada: Se usaron interfaces y herencia.
- Justificación técnica: Ahora se pueden agregar nuevas funcionalidades sin modificar el código.

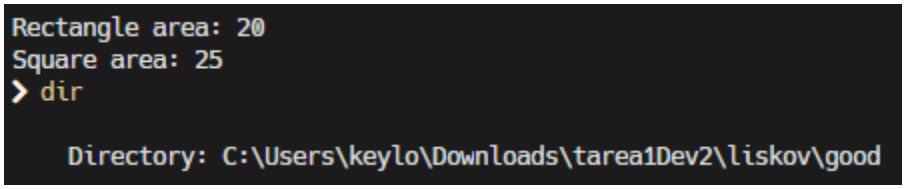
Evidencia:

- 

3. Principio de Sustitución de Liskov (LSP)

- Código original: Square heredaba de Rectangle y causaba comportamientos incorrectos.
- Problema identificado: No se podía usar Square donde se esperaba un Rectangle.
- Refactorización aplicada: Se creó una clase base llamada Shape.
- Justificación técnica: Todas las figuras se pueden usar de la misma forma.

Evidencia:

- 

4. Principio de Segregación de Interfaces (ISP)

- Código original: Una interfaz tenía métodos que no todas las clases necesitaban.
- Problema identificado: Algunas clases implementaban métodos que no usaban.
- Refactorización aplicada: Se dividió la interfaz en varias más pequeñas.
- Justificación técnica: Cada clase implementa solo lo que necesita.

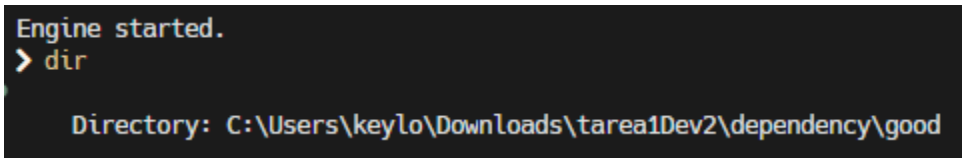
Evidencia:

- 

5. Principio de Inversión de Dependencias (DIP)

- Código original: La clase Car dependía directamente de Engine.
- Problema identificado: Esto crea un fuerte acoplamiento.
- Refactorización aplicada: Se creó una interfaz Engine.
- Justificación técnica: Ahora es más fácil cambiar el tipo de motor.

Evidencia:

- 

Repositorio

https://github.com/keyloor/IF0004_SOLID_Tarea-1