# Actividad #2

## Dependency Inversion:

Se aplica para mejorar la mantenibilidad del codigo.

Con la interaz IDatabaseOperations define los métodos esenciales para las operaciones de bases de datos, con esto se permite desacoplar las clases MySql y MongoDB de la clase que las usa. Ahora la clase Database depende de la interfaz y no de implementaciones concretas.

Con la implementación de IDatabaseOperations en MySql y MongoDB garantizamos que ambas cumplan con lo que se definió en la interfaz para las operaciones.

Anteriormente, la clase Database tenía una dependencia directa con la implementación concreta de MySql ahora la clase Database depende de la abstracción IDatabaseOperations.

Con esto se facilita cambiar de base datos, si en el futuro se añade una nueva base de datos (por ejemplo, PostgreSQL), solo se debe implementar la interfaz IDatabaseOperations en la nueva clase sin modificar la clase Database.

## Segregation interfaces

Inicialmente la interfaz ICamera definía una serie de métodos que no todas las cámaras implementaban, obligando a la clase CheapCamera a implementar métodos que no le correspondían (como detectMovement, termalDetection, faceRecognition), lo cual obligaba a lanzar excepciones.

Aplicando la segregación de interfaces se divide las responsabilidades en interfaces específicas y se mejora la coherencia entre lo que se necesita.

Se definen interfaces con la definición necesaria.

- IPhotoCamera: Define la funcionalidad relacionada únicamente con la toma de fotos.
- IVideoCamera: Define la funcionalidad para grabar videos.
- IAdvancedCamera: Hereda de IPhotoCamera y IVideoCamera y agrega un método que solo algunas cámaras necesitan implementar.

- CheapCamera: Ahora solo implementa IPhotoCamera e IVideoCamera, que son las funcionalidades que realmente le corresponden.
- SmartCamera: Implementa la interfaz IAdvancedCamera, que agrupa todas las funcionalidades que una cámara avanzada requiere, como reconocimiento facial, detección de movimiento y detección térmica.

## Liskov

Se definio la clase Employee como una clase abstracta con esto se obliga a que cada subclase implemente su propio metodo calculatePayment haciendo que ese comportamiento sea especifico en cada subclase.

Tanto Programmer como Tester deben implementar su propia versión de calculatePayment(), sin sobrescribir un método concreto de la clase base.

Si en el futuro se añaden nuevos tipos de empleados con diferentes métodos de pago, simplemente se crea una nueva subclase que implemente su propia versión de calculatePayment

## Open/Close

Se introducen adapters para cada instrumento (DrumAdapter, GuitarAdapter, PianoAdapter) que implementan la nueva interfaz Instrument con el método playNotes, con esto garantizamos que si en el futuro se quiere agregar nuevos comportamientos se podrá realizar mediante los adaptadores correspondientes sin necesidad de alterar las clases originales.

## Single responsibility

Originalmente la clase User tenía incluido muchas responsabilidades entre ellas: la gestión de usuario, autenticación, gestión de dirección y correo electrónico.

Con esto cualquier cambio en el comportamiento relacionado con la autenticación, la gestión de direcciones o el envío de correos impactaría directamente en la clase User.

Se separa entonces cada responsabilidad en una clase independiente.

- Usuario: Encargada de la gestión del usuario
- Address: Encargada de gestionar y modificar la información de dirección

- Email: Encargada del envio de correos electrónicos.
- Password: Encargada de la autenticación del usuario.

Con esta reestructuración se garantiza que cualquier cambio sobre un tema específico (usuario, address, email, password) afecte a otros que no están relacionados.