**1. ¿Cumple el principio de Inversión de Dependencia?**

No, la implementación actual **no cumple** con el **Principio de Inversión de Dependencia (DIP)**. El **DIP** establece que:

* Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones.
* Las abstracciones no deben depender de los detalles. Los detalles deben depender de las abstracciones.

En el caso de la clase Factura, esta está dependiendo directamente de clases concretas como Deduccion y Iva. Esto significa que si alguna de estas clases cambia, también tendríamos que modificar la clase Factura, lo cual rompe con el principio de inversión de dependencias.

**Problemas en el diseño actual:**

* **Dependencia directa en clases concretas**: La clase Factura crea instancias de las clases Deduccion e Iva dentro del método calcularTotal, lo que introduce un acoplamiento fuerte entre Factura y esas clases.
* **Difícil de extender y probar**: No es fácil reemplazar o modificar el comportamiento de las clases Deduccion o Iva sin cambiar la clase Factura.

**Conclusión**

Con esta refactorización, la clase Factura ahora **cumple con el Principio de Inversión de Dependencia (DIP)** porque:

* **Factura** depende de interfaces (Deduccion e Iva), no de implementaciones concretas.
* Esto permite que las implementaciones de Deduccion e Iva puedan cambiar sin afectar la clase Factura.
* El diseño es más extensible y fácil de probar, ya que se pueden sustituir las implementaciones concretas por otras nuevas o incluso por mocks para pruebas.