

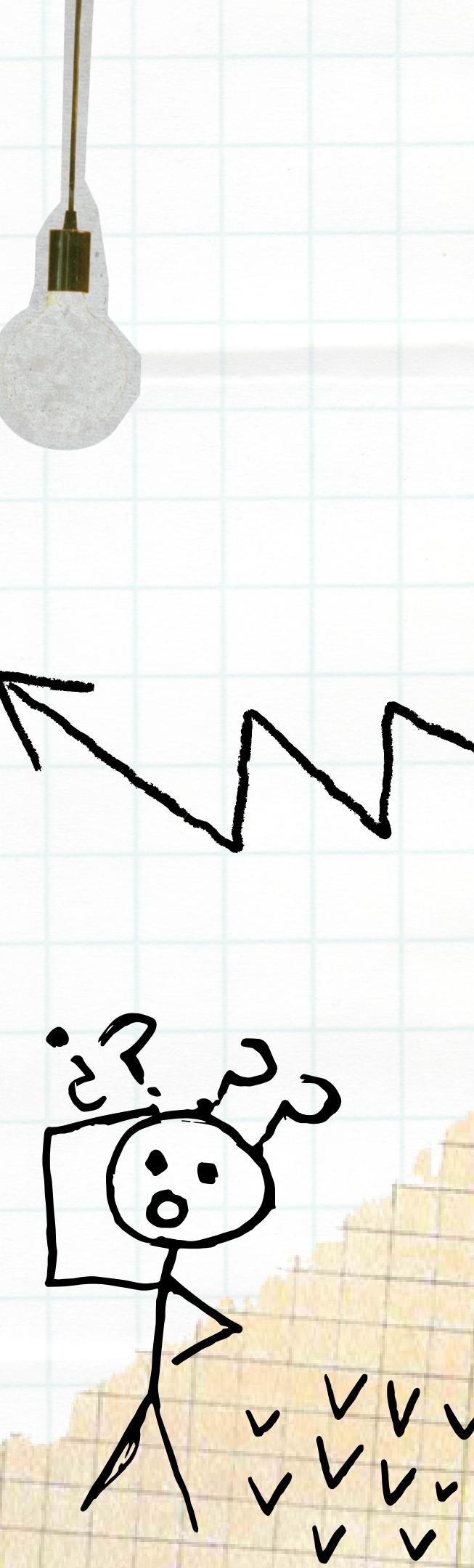
# INYECCION DE DEPENDENCIAS

---

Tecnología Diseño de aplicaciones 2

# Cuando se da una dependencia?

Cuando dos piezas, componentes, librerías, módulos, clases, funciones requieren de la otra para funcionar.



# A nivel de clases

significa que una cierta 'Clase A' tiene algún tipo de relación con una 'Clase B', delegándole el flujo de ejecución a la misma en cierta lógica.

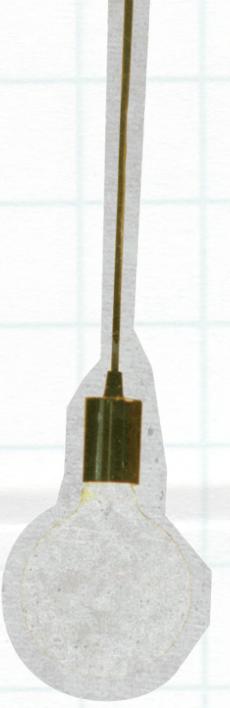
# Ejemplo

```
public class UserLogic : IUserLogic  
{  
    public IRepositoryUser users;  
  
    public UserLogic()  
    {  
        users = new UserRepository();  
    }  
}
```

**¿Que problema  
pueden causar  
estas dependencia?**

El problema reside en que ambas piezas de código tiene la responsabilidad de la instanciación de sus dependencias.

**¿Por que esto es  
malo?**

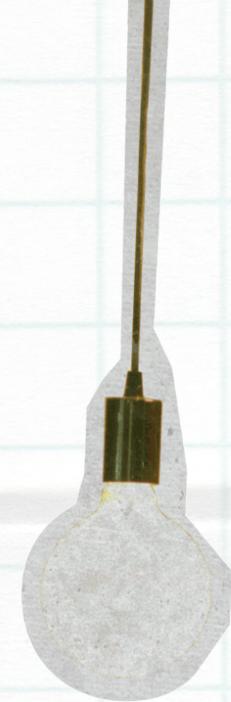


- Si queremos reemplazar por una implementación diferente
- Si la UserLogic tiene sus propias dependencias, debemos configurarlas dentro del controller.
- Es muy difícil de testear, ya que las dependencias 'están hardcodeadas'

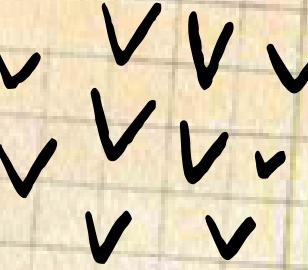
**¿Cómo resolvemos  
esto?**

Vamos a injectar la dependencia de la lógica de negocio en nuestro controller, y vamos a injectar la dependencia del repositorio de datos en nuestra lógica de negocio.

# Aplicando ID al ejemplo anterior



```
public class UserLogic : IUserLogic  
{  
    public IRepositoryUser users;  
  
    public UserLogic(IRepositoryUser users)  
    {  
        this.users = users;  
    }  
}
```



# Luego. hago en la clase startup da inyeccion

```
services.AddDbContext<DbInterfaz, DbContext>(  
    o => o.UseSqlServer(Configuration.GetConnectionString("REFERENCIA_AL_ARCHIVO_D  
E_CONFIGURACIÓN"))  
);
```

```
services.AddScoped< IRepositoryUser, UserRepository>();
```

# Ventajas de usar ID

- 1- Código más limpio.
- 2- Más fácil de Testear.
- 3- Más fácil de modificar.
- 4. Permite NO Violar SRP.
- 5- Permite NO Violar OCP.