

# Inyección de dependencias

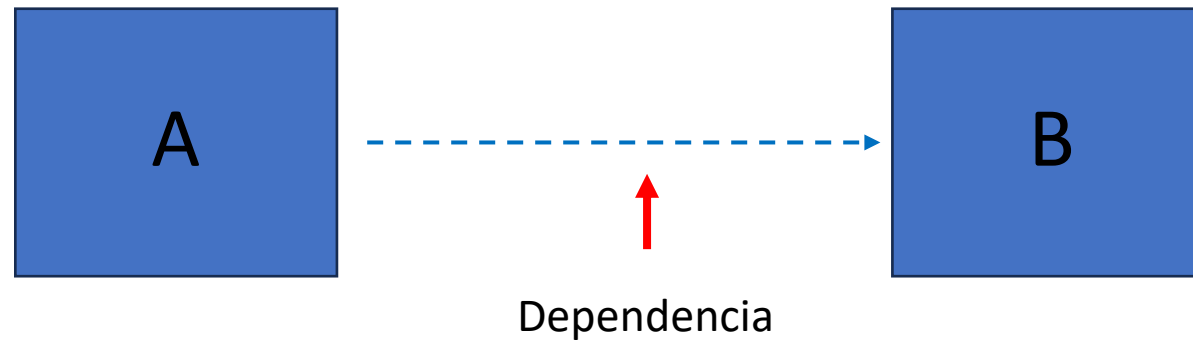
- Dependencias
- ¿Cuál es el propósito?
- Dónde?
- Práctica



# Dependencias

**En términos generales:** Cuando un módulo A depende de otro módulo B, significa que el módulo A utiliza el módulo B. O dicho de otra forma, que el módulo A necesita el módulo B para funcionar.

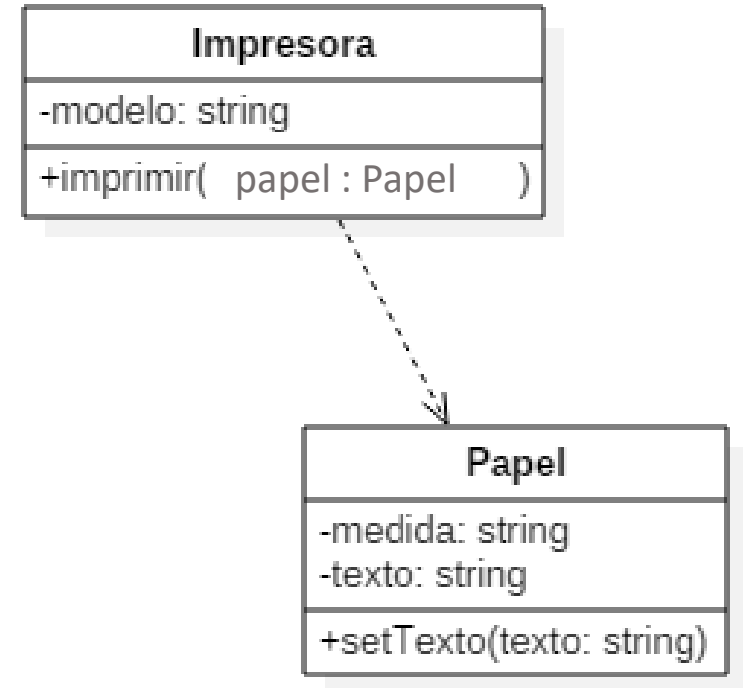
**En termino de POO:** La dependencia entre dos clases declara que una de ellas necesita conocer acerca de la clase a la que utilizará.



# Dependencias

Supongamos el siguiente caso:

Una *Impresora* **usa** un *Papel* para imprimir.  
Por lo tanto, una clase *Impresora* depende de una clase *Papel* para poder imprimir ya que este tiene información, tamaño, texto, etc.



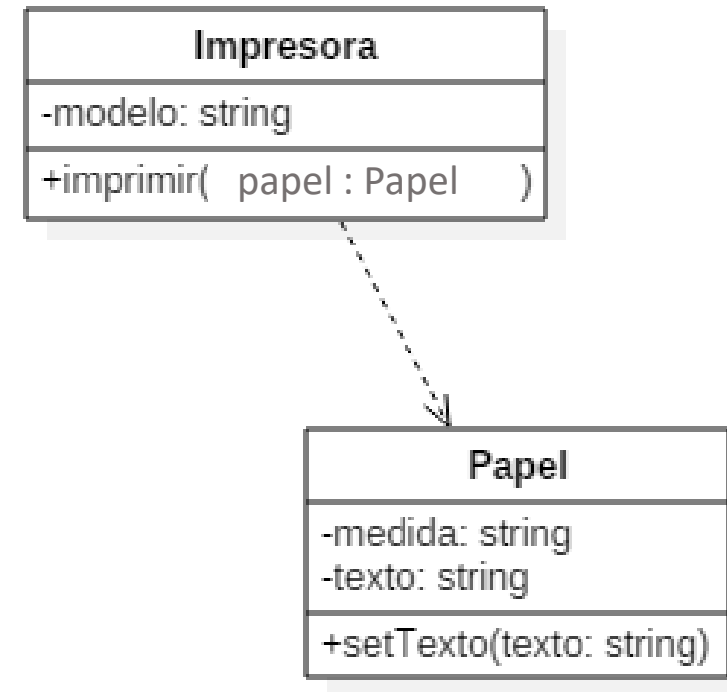
# Dependencias

Supongamos el siguiente caso:

```
public class Impresora
{
    private string modelo;

    public Impresora(string modelo)
    {
        this.modelo = modelo;
    }

    public void Imprimir(Papel papel)
    {
        System.SetTamañoPapel(papel.medida);
        System.Print(papel.GetText());
    }
}
```



# Inyección de dependencias

La Inyección de Dependencias consiste en proporcionar las dependencias que un objeto necesita desde el exterior, en lugar de que el objeto las cree internamente.

```
public class ClaseA
{
    private ClaseB _objetoB = new ClaseB();
}
```

Aplicando  
Inyección de  
dependencias



```
public class ClaseA
{
    private ClaseB _objetoB;

    public ClaseA(ClaseB objetoB)
    {
        _objetoB = objetoB;
    }
}
```

# Inyección de dependencias

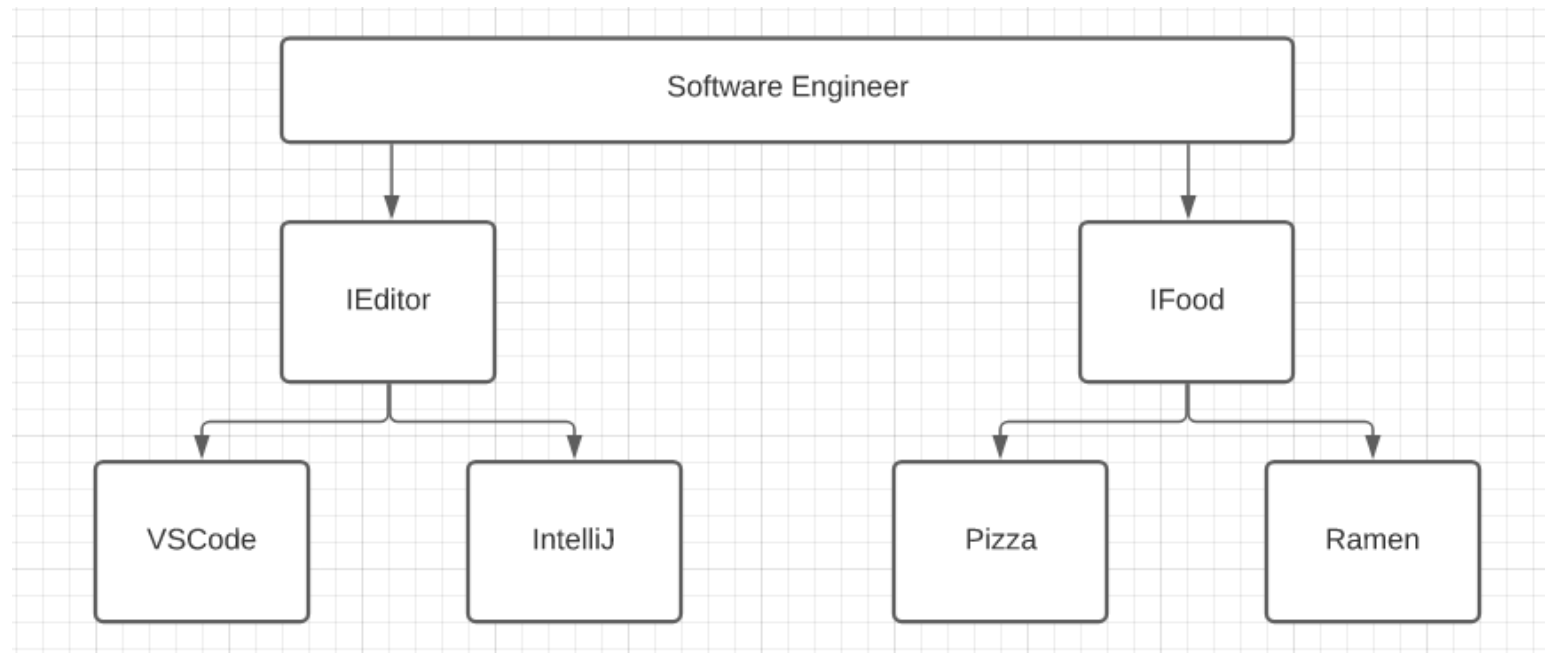
El concepto de Inyección de Dependencia surge de un principio de diseño de software llamado Principio de Inversión de Dependencia . Este es uno de los principios SOLID , y consta de dos partes:

- Los módulos de alto nivel no deberían importar nada de los módulos de bajo nivel. Ambos deben depender de abstracciones (por ejemplo, interfaces).
- Las abstracciones no deben depender de los detalles. Los detalles (implementaciones concretas) deberían depender de abstracciones.

# Inyección de dependencias

¿Cuál es el propósito?

Para mejorar la separación de responsabilidades entre las clases y permitir que diferentes partes del código puedan cambiar independientemente.



# Inyección de dependencias

## Beneficios

- **Desacoplamiento:** Un componente no necesita saber cómo se crean sus dependencias; simplemente se le proporcionan.
- **Reusabilidad:** Puedes cambiar las implementaciones de las dependencias sin afectar el componente principal.
- **Mantenibilidad:** Cambiar o agregar nuevas dependencias se convierte en una tarea más sencilla y menos propensa a errores.
- **Flexibilidad:** Permite configurar y cambiar comportamientos en tiempo de ejecución mediante la inyección de diferentes implementaciones de dependencias.



# Inyección de dependencias

## Un ejemplo

```
public class MovieRepository()
```

```
{  
    private readonly string _connectionString;
```

Ahora la variable privada tiene el valor que recibió por constructor.

```
    public MovieRepository (string connectionString)
```

Inyecto la cadena de conexión

```
{  
    _connectionString = connectionString;  
}
```

Asigno la cadena de conexión a la variable privada

```
    public List<Movie> GetAll()
```

```
{  
    using(IDbConnection conn = new DbConnection(_connectionString))  
    {  
        //Returns a list of all movies in our database  
    }  
}
```

# Inyección de dependencias

## ¿Que inyectamos?

```
public interface IMovieRepository
{
    List<Movie> GetAll();
    Movie GetByID(int id);
}
```

```
public class MovieRepository : IMovieRepository
{
    public List<Movie> GetAll()
    {
        //Implementation
    }
    public Movie GetByID(int id)
    {
        //Implementation
    }
}
```

# Inyección de dependencias

## ¿Como inyectar en .net?

En lugar de que un componente controle la creación y gestión de sus dependencias, la inversión de control delega esta responsabilidad a un contenedor o framework externo.

### **Pasos para la inyección de dependencias**


- 1 . Registrar Dependencias
- 2 . Inyección en Controladores

# Inyección de dependencias


## Pasos para inyectar dependencias en .net

**1. Registrar Dependencias:** La implementación de .NET 6 WebApplicationBuilder expone el objeto Services, lo que nos permite agregar servicios al contenedor .NET, que luego .NET inyectará en las clases dependientes.


**builder.Services.AddScoped<IMovieRepository, MovieRepository>();**




Objeto  
constructor  
de la webApp




Contenedor  
de Servicios  
de la  
webApp



Tiempo de vida  
del objeto  
inyectado



Interfaz que  
implementa  
el objeto a inyectar



Clase que implementa la  
interfaz antes expuesta

# Inyección de dependencias

## Pasos para inyectar dependencias en .net

**2. Inyección en Controladores:** En los controladores, las dependencias se inyectan mediante el constructor.

```
public class MiControlador : Controller
{
    private readonly IMovieRepository _movieRepository;

    public MiControlador( IMovieRepository movieRepository,)
    {
        _movieRepository = servicioTransient;
    }

    // Resto del código del controlador
}
```

# Inyección de dependencias

## ¿Qué tipos de inyecciones existen?

En .NET, existen tres tipos principales de inyección de dependencias que se utilizan comúnmente:

- Transient
- Scoped
- Singleton

# Inyección de dependencias

## ¿Qué tipos de inyecciones existen?

**1. Transient:** Un nuevo objeto **es creado para cada solicitud de servicio**. Es decir, cada vez que alguien solicita el servicio, se crea una nueva instancia.

### Uso Típico:

- Para servicios ligeros y efímeros que no almacenan estado.
- Cuando se desea una nueva instancia cada vez que se resuelve el servicio.
- Servicios que realizan transformaciones o cálculos efímeros.
- donde necesitas generar instancias de objetos de manera dinámica y cada instancia generada no tiene dependencias comunes
- Cuando la creación de instancias del servicio no implica una sobrecarga significativa

```
builder.Services.AddTransient<IMiServicioTransient, MiServicioTransient>();
```

# Inyección de dependencias

## ¿Qué tipos de inyecciones existen?

**2. Scoped:** Un objeto único es creado por cada **solicitud HTTP** en el ámbito de la aplicación. En otras palabras, se utiliza la misma instancia para toda la duración de una solicitud HTTP.

### Uso Típico:

- Para servicios que contienen estado que debe ser compartido dentro del mismo ámbito de la solicitud.
- Es útil para la gestión de transacciones en una solicitud HTTP.
- Un caso típico son los repositorios

```
builder.Services.AddScoped<IRepositorio, Repositorio>();
```



# Inyección de dependencias

**3. Singleton:** Un único objeto es creado para toda la duración de la aplicación. La misma instancia se utiliza para todas las solicitudes y para todo el tiempo de vida de la aplicación.

## Uso Típico:

- Para servicios que son costosos de instanciar y que deben compartirse entre todas las solicitudes.
- Para servicios que mantienen un estado global.
- Servicios que gestionan configuraciones globales o parámetros de la aplicación
- Mecanismo de caché o almacenamiento de datos global que persista durante toda la duración de la aplicación

**`builder.Services.AddScoped<ICustomLogger, CustomLogger>()`**

# Bibliografía

<https://exceptionnotfound.net/dependency-injection-in-dotnet-6-service-lifetimes/>

<https://exceptionnotfound.net/dependency-injection-in-dotnet-6-adding-and-injecting-dependencies/>

<https://exceptionnotfound.net/dependency-injection-in-dotnet-6-intro-and-background/>

<https://learn.microsoft.com/en-us/previous-versions/msp-n->

[p/ff649690\(v=pandp.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff649690(v=pandp.10)?redirectedfrom=MSDN)

<https://learn.microsoft.com/es-es/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application>

# Inyección de dependencias

## En .net

¿Qué puede ser una dependencia?

Cualquier cosa con una abstracción. Por lo general, se trata de clases u objetos pequeños y fáciles de modificar.

¿Dónde existen estas dependencias?

Agregamos esos servicios al "contenedor" para un proyecto

¿Cómo se inyectan estas dependencias en las clases que las necesitan?

.NET hará esto automáticamente para los servicios en el contenedor.

# Inyección de dependencias

En .net

¿Cómo se comportan estas dependencias?

Podemos asignar "tiempos de vida" a las dependencias para especificar si se crean una vez, en cada solicitud o cada vez que se necesitan.

¿A qué tipo de problemas podemos llegar con DI?

La principal son las "dependencias circulares", pero hay otras.