**Resumen – Capa de Servicio e Inyección de Dependencias en ASP.NET Core**

**1. Capa de Servicio**

* **Objetivo:** Separar las reglas de negocio de los controladores para evitar código duplicado y facilitar mantenimiento.
* **Controlador:**
  + Recibe solicitudes (request).
  + Llama a la capa de servicio para aplicar reglas.
  + Devuelve la respuesta.
* **Servicio:**
  + Contiene validaciones, cálculos, algoritmos, lógica de negocio.
  + Se implementa a través de **interfaces** (IPeopleService) para mantener un contrato de lo que debe hacer, sin definir cómo.
  + **Convención:** interfaces comienzan con I mayúscula.

**2. Creación de la Capa de Servicio**

* Carpeta Services con:
  1. **Interface** (IPeopleService) → define métodos (ej. bool Validate(People p)).
  2. **Clase de implementación** (PeopleService) → contiene la lógica real.
* Ventaja: Cambios en la validación se hacen solo en un lugar, y afectan a todos los controladores que usan el servicio.

**3. Inyección de Dependencias (DI)**

* **Concepto:** El controlador no crea directamente el servicio (new), lo recibe ya instanciado por el framework.
* **Principio SOLID relacionado:** *Dependency Inversion* → depender de **abstracciones** (interfaces) y no de implementaciones concretas.
* **Registro en Program.cs:**

builder.Services.AddSingleton<IPeopleService, PeopleService>();

* **Uso en el controlador:**

public PeopleController(IPeopleService service) {

\_peopleService = service;

}

* Cambiar la implementación (ej. People2Service) requiere solo modificar el registro en Program.cs.

**4. Inyección de Dependencias por Clave (Key) – .NET 8**

* Permite registrar múltiples implementaciones de la misma interfaz y seleccionarlas con un **key**.
* **Registro:**

builder.Services.AddKeyedSingleton<IPeopleService, PeopleService>("PeopleService");

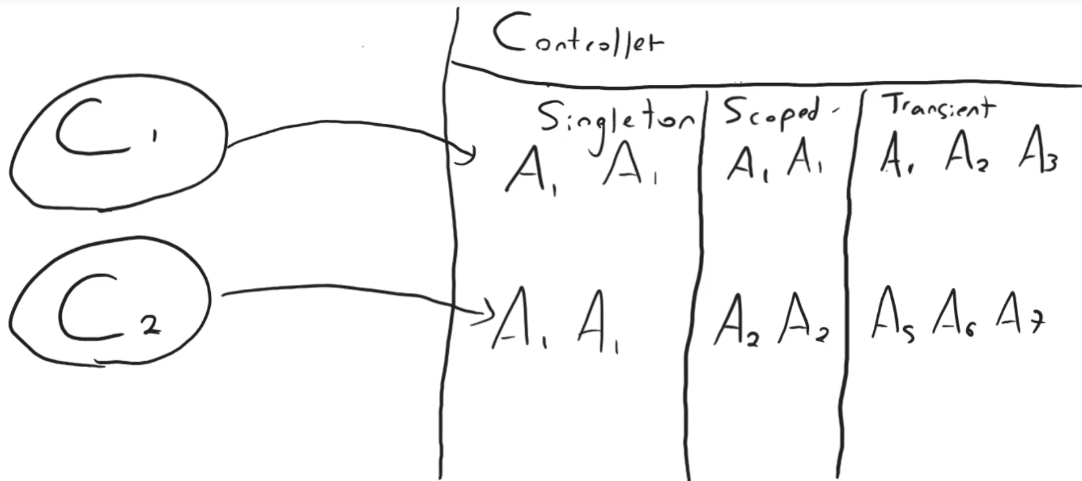
builder.Services.AddKeyedSingleton<IPeopleService, People2Service>("People2Service");

* **Uso:**

public PeopleController([FromKeyedServices("PeopleService")] IPeopleService service) { ... }

**5. Tipos de Ciclo de Vida en la DI**

1. **Singleton:**
   * Un solo objeto compartido por toda la aplicación y todos los clientes.
2. **Scoped:**
   * Un nuevo objeto por cada solicitud HTTP, pero el mismo dentro de esa solicitud.
3. **Transient:**
   * Un nuevo objeto cada vez que se inyecta, incluso dentro de la misma solicitud.



**6. Ejemplo práctico de los Ciclos de Vida**

* Servicio RandomService genera un número aleatorio en el constructor.
* Registrado como:
* AddKeyedSingleton<IRandomService, RandomService>("Singleton")
* AddKeyedScoped<IRandomService, RandomService>("Scoped")
* AddKeyedTransient<IRandomService, RandomService>("Transient")
* **Comportamiento:**
  + **Singleton:** Mismo número para todos los clientes y solicitudes.
  + **Scoped:** Número distinto por cada solicitud, pero se repite dentro de la misma.
  + **Transient:** Número distinto en cada inyección, incluso dentro de la misma solicitud.