Service Life Time

a **service** refers to any class or component that provides specific functionality or a particular service that other classes (known as **clients**) depend on

**1. Transient: New Instance Every Time**

**Analogy:** Think of a disposable coffee cup. Every time you want a coffee, you use a new cup.

**Real-Life Example:** A service that creates a random number every time it's needed.

**Code Example:**

public interface IRandomNumberService

{

int GenerateRandomNumber();

}

public class RandomNumberService : IRandomNumberService

{

private static readonly Random \_random = new Random();

public int GenerateRandomNumber()

{

return \_random.Next();

}

}

// Register in Startup.cs

services.AddTransient<IRandomNumberService, RandomNumberService>();

// Use in a controller

public class RandomController : ControllerBase

{

private readonly IRandomNumberService \_randomNumberService;

public RandomController(IRandomNumberService randomNumberService)

{

\_randomNumberService = randomNumberService;

}

[HttpGet("random-number")]

public ActionResult<int> GetRandomNumber()

{

return \_randomNumberService.GenerateRandomNumber();

}

}

Here, every time GetRandomNumber is called, a new RandomNumberService instance is used, just like getting a new coffee cup for each coffee.

### 2. Scoped: One Instance Per Request

**Analogy:** Think of a reusable shopping bag you use for one shopping trip. For the duration of your shopping trip, you use the same bag, but you get a new one each time you go shopping.

**Real-Life Example:** A service that keeps track of items in a shopping cart during a single shopping session.

**Code Example:**

public interface IShoppingCartService

{

void AddItem(string item);

List<string> GetItems();

}

public class ShoppingCartService : IShoppingCartService

{

private readonly List<string> \_items = new List<string>();

public void AddItem(string item)

{

\_items.Add(item);

}

public List<string> GetItems()

{

return \_items;

}

}

// Register in Startup.cs

services.AddScoped<IShoppingCartService, ShoppingCartService>();

// Use in a controller

public class CartController : ControllerBase

{

private readonly IShoppingCartService \_shoppingCartService;

public CartController(IShoppingCartService shoppingCartService)

{

\_shoppingCartService = shoppingCartService;

}

[HttpPost("add-item")]

public IActionResult AddItem(string item)

{

\_shoppingCartService.AddItem(item);

return Ok();

}

[HttpGet("items")]

public ActionResult<List<string>> GetItems()

{

return \_shoppingCartService.GetItems();

}

}

Here, ShoppingCartService keeps the same list of items for the duration of the user’s session, similar to using the same shopping bag for one trip.

### 3. Singleton: One Instance for the Whole Application

**Analogy:** Think of a single water bottle dispenser in an office. Everyone uses the same dispenser, and it stays the same throughout its lifetime.

**Real-Life Example:** A service that provides application-wide settings that don’t change, like the app's name or version.

**Code Example:**

public interface IAppSettingsService

{

string GetAppName();

}

public class AppSettingsService : IAppSettingsService

{

private readonly string \_appName = "MyCoolApp";

public string GetAppName()

{

return \_appName;

}

}

// Register in Startup.cs

services.AddSingleton<IAppSettingsService, AppSettingsService>();

// Use in a controller

public class SettingsController : ControllerBase

{

private readonly IAppSettingsService \_appSettingsService;

public SettingsController(IAppSettingsService appSettingsService)

{

\_appSettingsService = appSettingsService;

}

[HttpGet("app-name")]

public ActionResult<string> GetAppName()

{

return \_appSettingsService.GetAppName();

}

}

Here, AppSettingsService provides the same application name to everyone, like a single water dispenser used by everyone in the office.

**Summary**

* **Transient:** New instance every time (disposable coffee cup).
* **Scoped:** One instance per request or session (reusable shopping bag for one shopping trip).
* **Singleton:** One instance for the entire application (single water bottle dispenser in an office).

These analogies should help make the concept of service lifetimes in C# easier to understand.

The main purpose of a DTO is to reduce the number of method calls and network overhead by bundling multiple parameters into a single call.