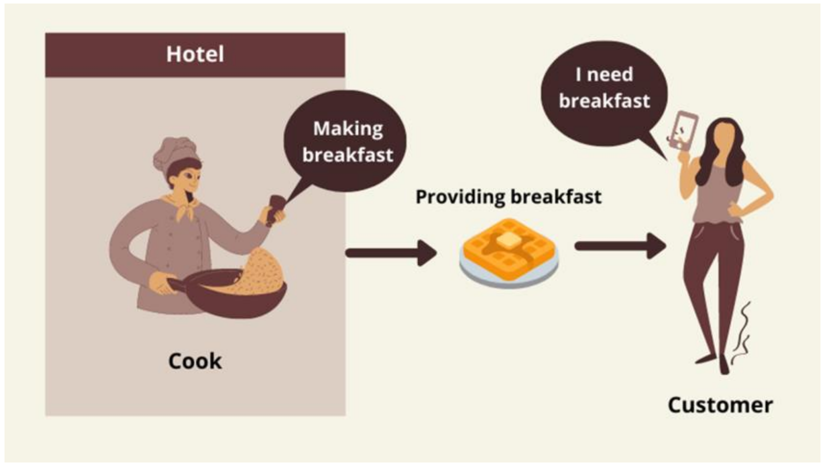
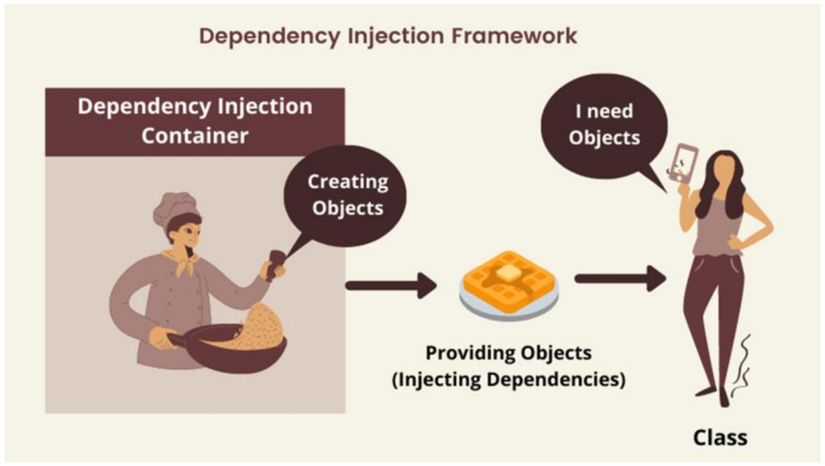
What is Dependency Injection

- “In software engineering, dependency injection is a technique in which an object receives other objects that it depends on. These other objects are called dependencies.”
- Dependency injection is a programming technique that makes a class independent of its dependencies.
- We don’t create objects of other classes (dependencies) in our code directly but ask someone else (DI Container) to create objects for us and once these objects are created, they are provided (injected) in our classes so that we can use them in the code.

What is Dependency Injection




Dependency Injection Framework
Dependency Injection Container
Creating Objects
I need Objects
Providing Objects (Injecting Dependencies)
Class

Benefits of Dependency injection

- **Flexible and Maintainable Code:** it makes our code more flexible and maintainable because we can change the implementation of the class without changing the code or business logic in the application. Code is also easy to read because the initialization or creation of dependencies is done outside the code.
- **Easy Unit Testing:** We can easily test different implementations by injecting one class or another.
- **Loose Coupling:** Dependency Injection facilitates loose coupling of software components.
- **Easy to Scale Applications:** Dependency Injection makes it easy to extend our application as we can not only introduce new components more easily but we can also implement better versions of existing components and inject them into our applications easily.

Dependency Injection in ASP.NET Core

- The ASP.NET Core Framework is designed from scratch to support inbuilt support for Dependency Injection. The ASP.NET Core Framework injects objects of dependency classes through constructor or method by using a built-in IoC (Inversion of Control) container.
- The built-in container is represented by IServiceProvider implementation that supports constructor injection by default. The types (classes) managed by built-in IoC containers are called services.

Different Types of Dependency Injection in C#?

- The injector class injects the dependency object to a class in three different ways. They are as follows.
- **Constructor Injection:** When the Injector injects the dependency object (i.e. service) through the client class constructor, then it is called as Constructor Injection.
- **Property Injection:** When the Injector injects the dependency object (i.e. service) through the public property of the client class, then it is called as Property Injection. This is also called as the Setter Injection.
- **Method Injection:** When the Injector injects the dependency object (i.e. service) through a public method of the client class, then it is called as Method Injection. In this case, the client class implements an interface that declares the method(s) to supply the dependency object and the injector uses this interface to supply the dependency object (i.e. service) to the client class.

Types of Services in ASP.NET Core

- There are two types of services in ASP.NET Core. They are as follows:
- 1. Framework Services: Services that are a part of the ASP.NET Core framework such as IApplicationBuilder, IWebHostEnvironment, ILoggerFactory, etc.
- 2. Application Services: The services (custom types or classes) which you as a programmer create for your application.
- In order to let the IoC container automatically inject our application services, we first need to register them with the IoC container.

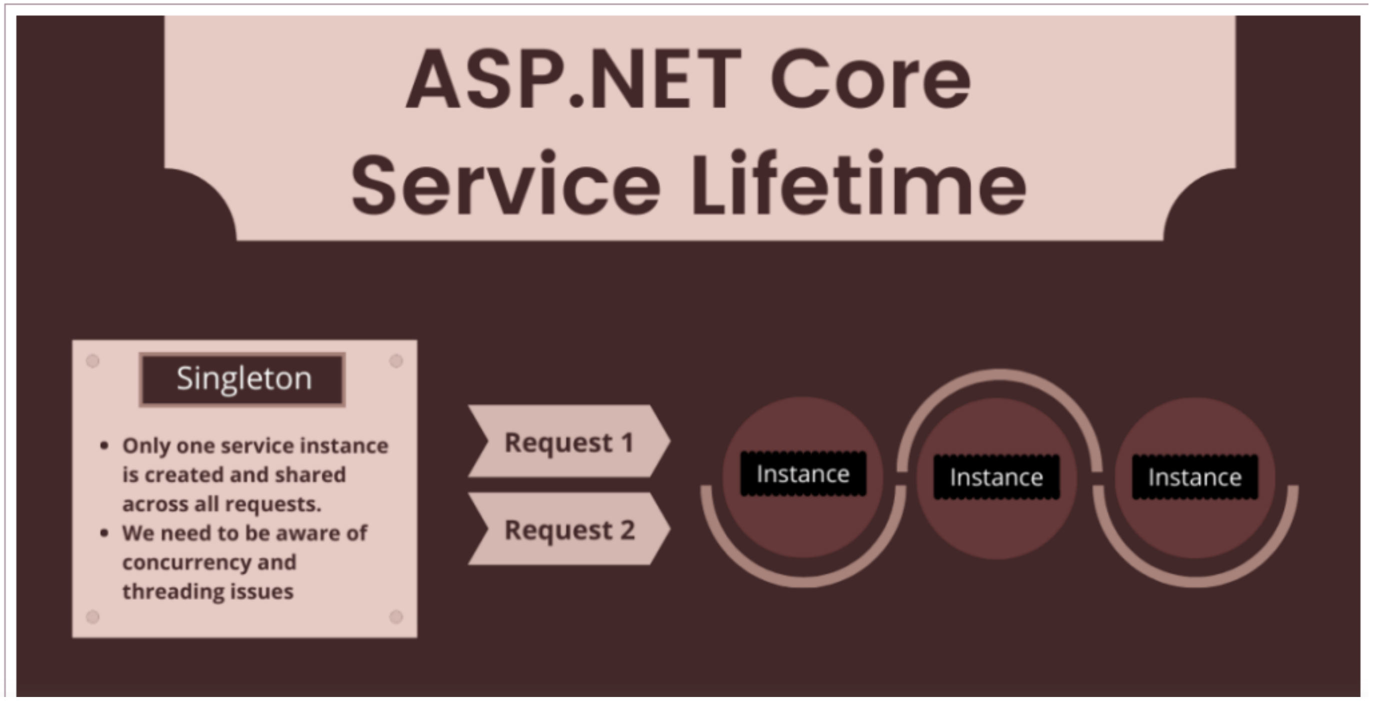
Register a Service with ASP.NET Core Dependency Injection Container?

- to register a service with ASP.NET Core Dependency Injection Container within the ConfigureServices() method of the Startup class up to 5.0 version.
- to register a service with ASP.NET Core Dependency Injection Container within the Program class from 6.0 version.
- **ASP.NET Core Provides different methods to register a service with Dependency Injection Contains**
- The ASP.NET core provides 3 methods to register a service with the ASP.NET Core Dependency Injection container as follows. The method that we use to register a service will determine the lifetime of that service.
- **Singleton**
- **Transient**
- **Scoped**

Singleton

- When we use the AddSingleton() method to register a service, then it will create a singleton service. It means a single instance of that service is created and that singleton instance is shared among all the components of the application that require it. That singleton service is created when we requested for the first time.

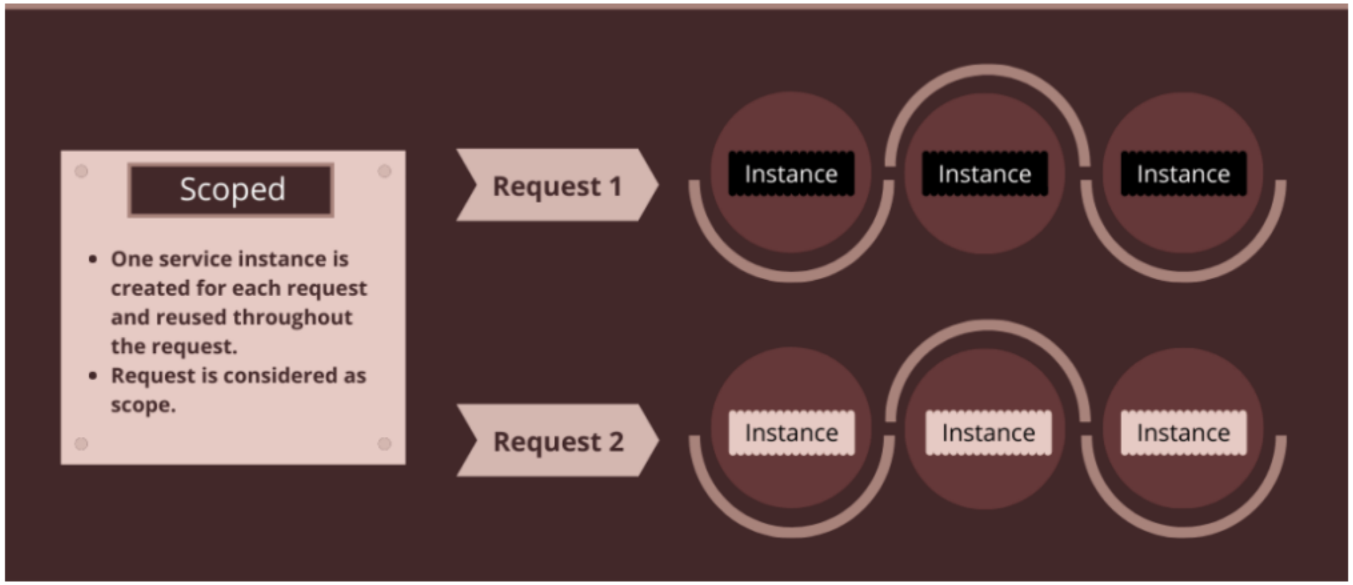
```
services.AddSingleton<IProductService, ProductService>();
```



- Only one service instance is created and shared across all requests.
- We need to be aware of concurrency and threading issues

Scoped

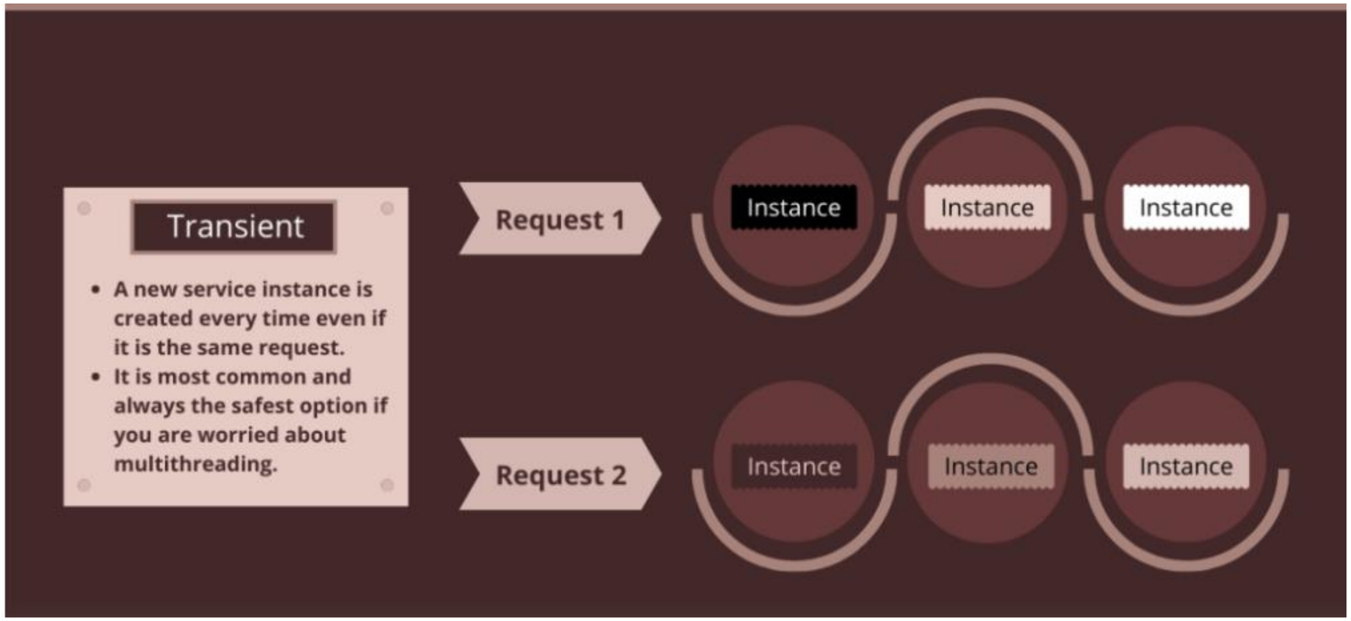
- Scoped
- Scoped means instance per request. When we use the AddScoped() method to register a service, then it will create a Scoped service. It means, an instance of the service is created once per each HTTP request and uses that instance in other calls of the same request.



- One service instance is created for each request and reused throughout the request.
- Request is considered as scope.

Transient

- Transient
- When we use the AddTransient() method to register a service, then it will create a Transient service. It means a new instance of the specified service is created each time when it is requested and they are never shared.


Transient
• A new service instance is created every time even if it is the same request.
• It is most common and always the safest option if you are worried about multithreading.
Request 1
Instance
Instance
Instance
Request 2
Instance
Instance
Instance