Lifetime of the service:

1. **Transient**: creates a new instance of the service, every time you requestit.
2. **Scoped**: creates a new instance for every scope. (Each request is a Scope). Within the scope, it reuses the existing service.
3. **Singleton**: Creates a new Service only once during the application lifetime, and uses it everywhere

public interface ITransientService

{

    Guid GetID();

}

public interface IScopedService

{

    Guid GetID();

}

public interface ISingletonService

{

    Guid GetID();

}

public class SomeService : ITransientService, IScopedService, ISingletonService

{

    Guid id;

    public SomeService()

    {

        id = Guid.NewGuid();

    }

    public Guid GetID()

    {

        return id;

    }

}

The service generates a unique id, whenever we instantiate it. The GetID method returns that id.

**Transient**

The Transient services always create a new instance, every time we request for it

**Register the Transient Service**

Now, under ConfigureServices method of the startup class register the SomeServive via ITransientService interface as shown below

|  |  |
| --- | --- |
| 1  2  3 | services.AddTransient<ITransientService, SomeService>(); |

**Inject it into Controller**

Open the HomeController and inject the two instance of the SomeService as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public class HomeController : Controller  {      ITransientService \_transientService1;      ITransientService \_transientService2;        public HomeController(ITransientService transientService1,                            ITransientService transientService2)      {          \_transientService1 = transientService1;          \_transientService2 = transientService2;      }        public IActionResult Index()      {            ViewBag.message1 ="First Instance " + \_transientService1.GetID().ToString();          ViewBag.message2 ="Second Instance "+ \_transientService2.GetID().ToString();            return View();      }  } |

First, We inject the two instance of the service via the interface ITransientService in the constructor of HomeController.

|  |  |
| --- | --- |
| 1  2  3 | public HomeController(ITransientService transientService1,ITransientService transientService2) |

Next, we query the GetID method in each instance and pass it to the view using [ViewBag](https://www.tektutorialshub.com/asp-net-core/asp-net-core-viewbag-viewdata/).

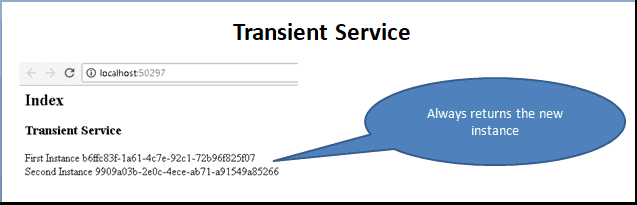
|  |  |
| --- | --- |
| 1  2  3  4 | ViewBag.message1 ="First Instance " + \_transientService1.GetID().ToString();  ViewBag.message2 ="Second Instance "+ \_transientService2.GetID().ToString(); |

**View**

Open the view and add the following code

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <h3>Transient Service</h3>  @ViewBag.message1  </br>  @ViewBag.message2 |

Run the application and you should see two different Guid are displayed on the screen. It is evident that we have received two new instances of the Transient service.



**Scoped**

The Services with scoped lifetime are created only once per each request (scope). I.e. It creates a new instance per request and reuses that instance within that request.

Now, let us see it with an example

**Register the Scoped Service**

Under the ConfigureServices method register the SomeService using the AddScoped method & using the IScopedService interface as shown below.

|  |  |
| --- | --- |
| 1  2  3 | services.AddScoped<IScopedService, SomeService>(); |

**Inject scoped service into Controller**

Next, inject the service into the controller. We already have transient service injected into the controller. Let us not change that. The HomeController is now as below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | public class HomeController : Controller  {        ITransientService \_transientService1;      ITransientService \_transientService2;        IScopedService \_scopedService1;      IScopedService \_scopedService2;          public HomeController(ITransientService transientService1,                            ITransientService transientService2,                            IScopedService scopedService1,                            IScopedService scopedService2)      {          \_transientService1 = transientService1;          \_transientService2 = transientService2;            \_scopedService1 = scopedService1;          \_scopedService2 = scopedService2;      }        public IActionResult Index()      {            ViewBag.message1 ="First Instance " + \_transientService1.GetID().ToString();          ViewBag.message2 ="Second Instance "+ \_transientService2.GetID().ToString();            ViewBag.message3 = "First Instance " + \_scopedService1.GetID().ToString();          ViewBag.message4 = "Second Instance " + \_scopedService2.GetID().ToString();            return View();       }  } |

**View**

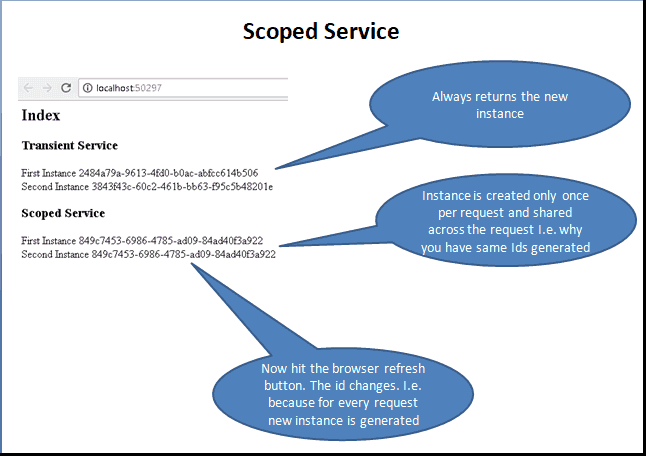
In the view add these lines

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <h3>Scoped Service</h3>  @ViewBag.message3  </br>  @ViewBag.message4 |

Run the application

The instance created only once per request, i.e. why you have the same IDs generated.

Now hit the browser refresh button. The id changes i.e. because DI creates the new instance of the service and reuses it



**Singleton**

The Singleton scope creates a single instance of the service when the request for it comes for the first time. After that for every subsequent request, it will use the same instance. The new request does not create the new instance of the service but reuses the existing instance.

**Register the Singleton Service**

Singleton services are registered using the AddSingleton method.

|  |  |
| --- | --- |
| 1  2  3 | services.AddSingleton<ISingletonService, SomeService>(); |

**Inject Singleton service into Controller**

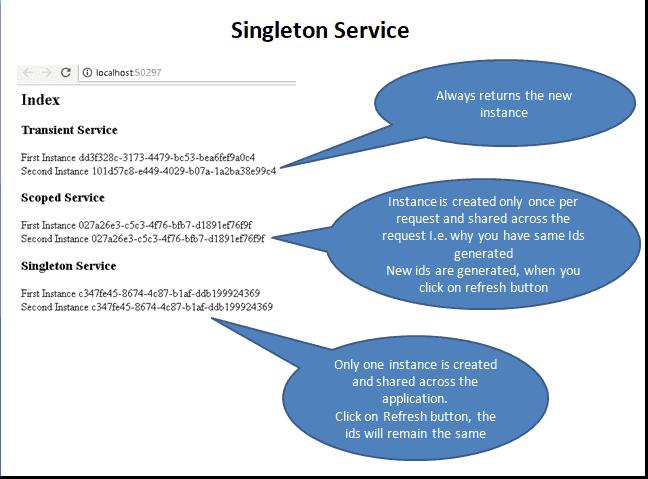
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | public class HomeController : Controller  {      ITransientService \_transientService1;      ITransientService \_transientService2;        IScopedService \_scopedService1;      IScopedService \_scopedService2;        ISingletonService \_singletonService1;      ISingletonService \_singletonService2;        public HomeController(ITransientService transientService1,                        ITransientService transientService2,                        IScopedService scopedService1,                        IScopedService scopedService2,                        ISingletonService singletonService1,                        ISingletonService singletonService2)      {            \_transientService1 = transientService1;          \_transientService2 = transientService2;            \_scopedService1 = scopedService1;          \_scopedService2 = scopedService2;            \_singletonService1 = singletonService1;          \_singletonService2 = singletonService2;         }         public IActionResult Index()       {             ViewBag.message1 ="First Instance " + \_transientService1.GetID().ToString();           ViewBag.message2 ="Second Instance "+ \_transientService2.GetID().ToString();               ViewBag.message3 = "First Instance " + \_scopedService1.GetID().ToString();           ViewBag.message4 = "Second Instance " + \_scopedService2.GetID().ToString();             ViewBag.message5 = "First Instance " + \_singletonService1.GetID().ToString();           ViewBag.message6 = "Second Instance " + \_singletonService2.GetID().ToString();             return View();       }  } |

First, we are injecting 6 instances the SomeService. Two instances per each interface. This is done in the constructor of the Controller.

### View

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | @{      ViewData["Title"] = "Index";  }    <h2>Index</h2>    <h3>Transient Service</h3>  @ViewBag.message1  </br>  @ViewBag.message2    <h3>Scoped Service</h3>  @ViewBag.message3  </br>  @ViewBag.message4    <h3>Singleton Service</h3>  @ViewBag.message5  </br>  @ViewBag.message6 |

Run the application. The ids generated from Singleton services are same and will not change even if you refresh the application. You can see it from the image below



## Which one to use

Transient services are safest to create, as you always get the new instance. But, since the dependency injection system creates them every time they will use more memory & Resources and can have a negative impact on performance if you too many of them.

Use Transient lifetime for the lightweight service with little or no state.

Scoped services service is the better option when you want to maintain state within a request.

Singletons are created only once and not destroyed until the end of the Application. Any memory leaks in these services will build up over time. Hence watch out for the memory leaks. Singletons are also memory efficient as they are created once reused everywhere.

Use Singletons where you need to maintain application wide state. Application configuration or parameters, Logging Service, caching of data is some of the examples where you can use singletons.

## Injecting service with different lifetimes into another

Be careful, while injecting service into another service with a different lifetime

Consider the example of Singleton Service, which depends on another Service which is registered with say the transient lifetime.

When the request comes for the first time a new instance of the singleton is created. It also creates a new instance of the transient object and injects into the Singleton service.

When the second request arrives the instance of the singleton is reused. The singleton already contains the instance of the transient service Hence it is not created again. This effectively converts the transient service into the singleton.

The services with the lower lifetime injected into service with a higher lifetime would change the lower lifetime service to a higher lifetime. This will make debugging the application very difficult and should be avoided at all costs.

Hence, remember the following rules

1. Never inject Scoped & Transient services into Singleton service.
2. Never inject Transient services into scoped service