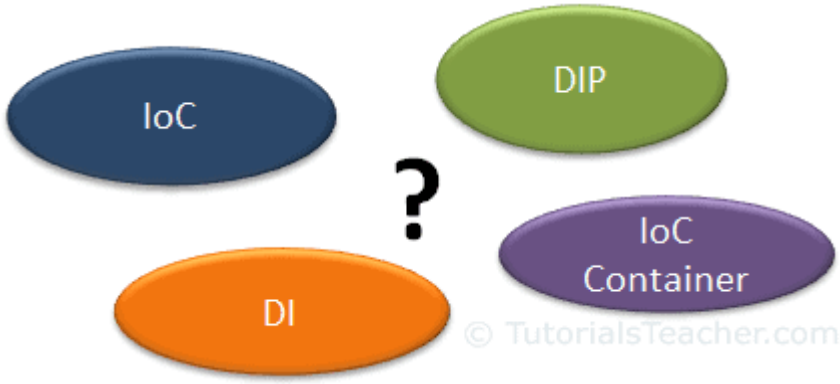
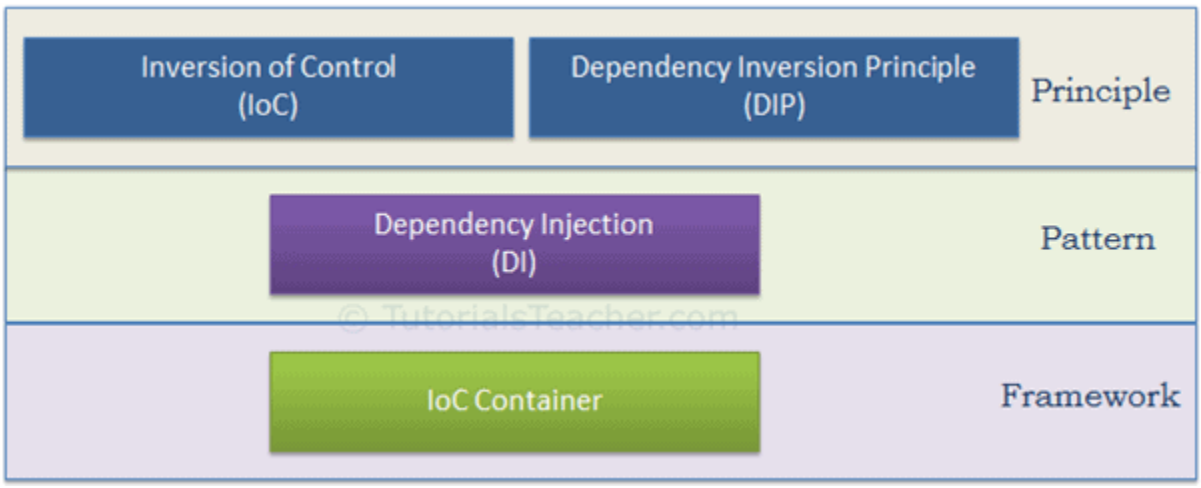
Inversion of Control – Deel 2

Terminologie: IoC, DIP, DI, Ioc Container

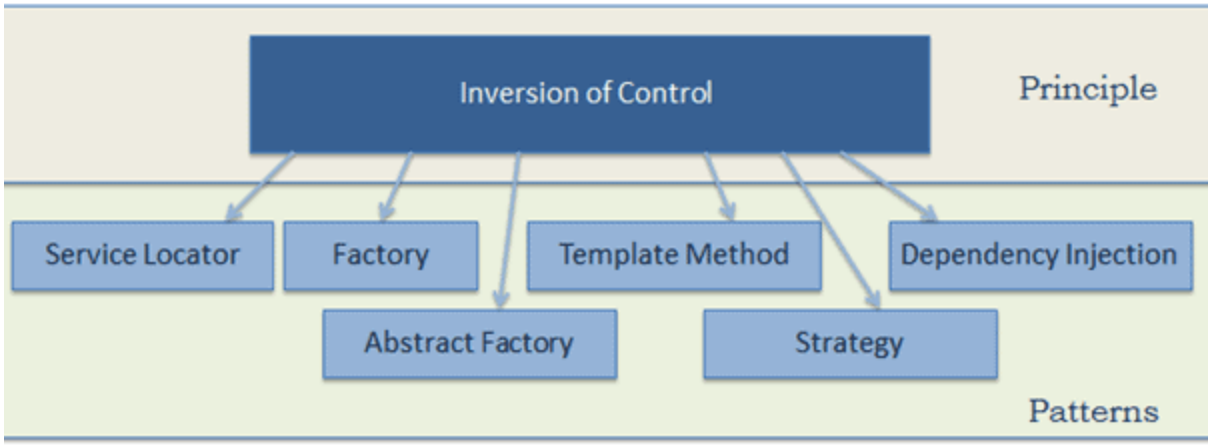


IoC en DIP zijn hoog-niveau design principes . Dependency Injection (DI) is a pattern en IoC container is een framework.



IoC is a principe, geen pattern. Het geeft enkel op hoog niveau richtlijnen, maar geen implementatie details.

De volgende **design patterns** o.a. **implementeren het IoC** (Inversion of Control) principe:



Dependency Injection in ASP.NET Core

ASP.NET Core ontworpen met built-in Dependency Injectio.

ASP.NET Core injecteert objecten van dependency-klassen dmv constructor of methode met behulp van een built-in IoC container.

Built-in IoC Container

ASP.NET Core bevat een eenvoudige out-of-the-box IoC container.

De built-in ASP.Net Core IoC Container ondersteunt standaard constructor injection door de `IServiceProvider` implementatie.De Types (classes) die beheerd worden door de built-in IoC container worden **services** genoemd.

Er zijn 2 types van services in ASP.NET Core:

1. **Framework Services**: Services die deel uitmaken van het ASP.NET Core framework, bv IApplicationBuilder, IHostingEnvironment, ILoggerFactory ,….
2. **Applicatie Services**: Services (custom types of klassen) die je als programmeur zelf creëert voor je applicatie.

Voordat de IoC container de applicatie services kan injecteren, moeten deze eerst worden geregistreerd met de IoC container.

Registreren van een Applicatie Service

Als voorbeeld hebben we bv een ILog interface en een class MyConsoleLogger die deze interface implementeert. We willen deze registreren met de built-in IoC container en de MyConsoleLogger kunnen aanspreken in onze applicatie.

```
public interface ILog
{
   void info(string str);
}

class MyConsoleLogger : ILog
{
   public void info(string str)
   {
      Console.WriteLine(str);
   }
}
```

ASP.NET Core laat ons toe om onze applicatie services met de IoC container te registreren in de **ConfigureServices** methode van de Startup class.
De **ConfigureServices** methode neemt een parameter van het type **IServiceCollection** die gebruikt wordt om de services te registreren.

Voorbeeld: Registreren van Service MyConsoleLogger **met de built-in IoC container van ASP.NET Core:**

```
public class Startup
{
   public void ConfigureServices(IServiceCollection services)
   {
      services.Add(new ServiceDescriptor(typeof(ILog), new MyConsoleLogger()));
   }

   //...
}
```

Via de **Add()** methode van de instantie van het type **IServiceCollection** kan een service geregistreerd worden met de IoC container. De ServiceDescriptor wordt gebruikt om een service type en zijn instantie. In het voorbeeld is ILog het service type en **MyConsoleLogger** zijn instantie. Deze zal bij default ILog service als een singleton

registreren. De ASP.NET Core IoC container zal een singleton object van **MyConsoleLogger** class creëren en het injecteren via de **constructors** of methoden van de klassen in de applicatie waarin een parameter van het type **ILog** is gespecifieerd.

Levensduur van een Service

De ASP.NET Core built-in IoC container beheert de levensduur van een geregistreerde service.

De ASP.NET Core built-in IoC container ondersteunt 3 verschillende levensduren:

1. **Singleton:** de IoC container zal één enkele instantie van de service aanmaken voor de hele levensduur van de applicatie en deze ene instantie wordt overal geïnjecteerd wanneer ernaar gevraagd wordt.
2. **Transient:** de IoC container creëert een nieuwe instantie van de service telkens deze wordt gevraagd.
3. **Scoped:** de IoC container creëert een nieuwe instantie van de service type per request en wordt gedeeld in één request.

Voorbeeld: Registreren van een Service met specificatie met verschillende levensduur (via Extensie methoden `AddSingleton, AddTransient, AddScoped`**)**

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<ILog, MyConsoleLogger>();
    services.AddTransient< ILog, MyConsoleLogger>();
    services.AddScoped<ILog, MyConsoleLogger>();

}
```

Constructor Injection

De IoC container levert automatische constructor injection voor de geregistreerde service. De injectie van de instantie van de service gebeurt bij alle klassen die in hun constructor het type van een geregistreerde service hebben.

Bijvoorbeeld de `ILog` service die geregisteerd is, kan gevraagd worden in om het eender welke Web API of MVC controller.

Voorbeeld: gebruik van met IoC container geregistreerde Service

```
public class HomeController : Controller
{
   private ILog _log;

   public HomeController(ILog log)
   {
      _log = log;
   }
   public IActionResult Index()
   {
      _log.info("Uitvoeren van  /home/index");

      return Ok(1);
   }
}
```

De IoC container zal automatisch een instantie van **MyConsoleLogger** doorgeven aan de constructor van **HomeController**. De IoC container zal de levensduur van de instantie regelen en deze creëren en create en weggooien op basis van de geregistreerde levensduur.

Action Methode Injectie

Indien een service type bv slechts in één action methode nodig is, is injectie van de service instantie in de methode eveneens mogelijk via het attribuut [FromServices] met het service type als parameter in de methode.

Voorbeeld: Action Methode Injectie

```
using Microsoft.AspNetCore.Mvc;

public class HomeController : Controller
{
   public HomeController()
   {
   }

   public IActionResult Index([FromServices] ILog log)
   {
      log.info("Index methode");

      return Ok(1);
   }
}
```

Property Injectie

De Built-in ASP.NET Core IoC container ondersteunt **geen** property injection. Er bestaan wel 3rd pary IoC containers die dit wel ondersteunen .

Manueel Service instantie opvragen

Via de `RequestServices` property van het `HttpContext` object is het eveneens mogelijk om een instantie van een geregistreerde service in de built-in IoC container op de vragen.

Voorbeeld: Service Instantie manueel opvragen

```
public class HomeController : Controller
{
   public HomeController()
   {
   }
   public IActionResult Index()
   {
      var services = this.HttpContext.RequestServices;
      var log = (ILog)services.GetService(typeof(ILog));

      log.info("Index methode");

      return Ok();
   }
}
```

Referenties

https://www.tutorialsteacher.com/ioc

https://jakeydocs.readthedocs.io/en/latest/fundamentals/dependency-injection.html

https://dotnettutorials.net/lesson/introduction-to-inversion-of-control/

https://dotnettutorials.net/lesson/dependency-injection-design-pattern-csharp/

https://www.yogihosting.com/aspnet-core-configurations/#project

https://dotnettutorials.net/lesson/dependency-inversion-principle/

https://www.yogihosting.com/aspnet-core-dependency-injection/#views

https://blog.davidwu.xyz/csharp-asp.net%20core/2020/05/12/understanding-built-in-ioc-container/