# Vkládaní závislostí

## Co je to vkládání závislostí a co nám umožňuje?

Při psaní kódu je důležité dbát na to, aby byl kód robustní, přehledný a lehce upravitelný. Je tedy důležité si projekt chytře rozdělit a vytvořit jednotlivé komponenty (objekty), které se pak využívají v různých místech programu.

Co ale dělat, když se rozhodneme jednu z těchto komponent nahradit? Obvykle to pro programátora znamená najít všechna místa, kde se daná komponenta využívá a ručně kód upravit. Takto zdánlivě jednoduchá úprava je pak u velkých projektů často velmi časově náročná.

Co kdyby tedy existoval způsob, jak psát více modulární kód, jehož jednotlivé části by šly snadněji nahradit? Nebudu vás dlouho napínat, tento způsob existuje a říká se mu vkládání závislostí, což je český překlad pro anglický termín *„dependency injection“*.

## Dependency Injection v ASP .NET Core

### Teorie

Framework ASP .NET Core podporuje vkládání závislostí již v základu. O vkládání závislostí se stará takzvaný IoC Container (Inversion Of Control Container), kterému Microsoft dala název Service Provider.

Service provider má k sobě zaregistrované služby (objekty), které si od něj ostatní objekty mohou vyžádat. Tyto služby se do Service Provideru přidávají ve Startup třídě projektu v metodě ConfigureServices.

**Služby se dají zaregistrovat 3 různými způsobu.**

* Singleton
  + během chodu aplikace se vytvoří pouze jedna instance služby
  + nejméně náročná na výkon
  + hodí se pouze pro případy, kdy uživatel aplikace nepotřebuje vlastní instanci služby
* Scoped
  + nová instance služby pro každého uživatele
  + středně náročná na výkon
* Transient
  + nová instance služby pro každý dotaz na server
  + ze všech 3 nejvíce náročná na výkon

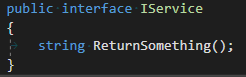
### Názorná ukázka

Nyní je čas si ukázat vkládání závislostí v praxi. Postup bude následovný.

* vytvoření rozhraní služby (interface)
* vytvoření služby implementující rozhraní
* registrace služby do Service Provideru
* vyžádání služby

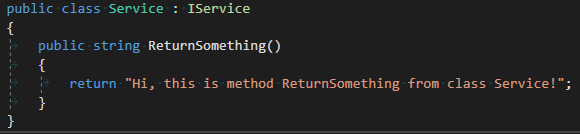
#### Rozhraní služby

Vytvoříme si jednoduché rozhraní služby.



#### Služba implementující rozhraní

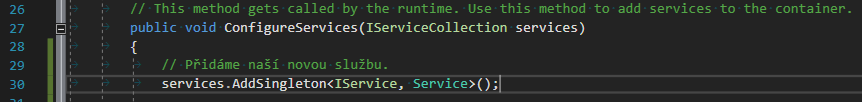
Vytvoříme si jednoduchou službu implementující rozhraní



#### Registrace služby do Service Provideru

Službu zaregistrujeme do ServiceProvideru jako Singleton. Jak můžete na obrázku vidět, funkce pro přidání služby vyžaduje 2 generické parametry. První z nich je rozhraní služby a druhý je implementace tohoto rozhraní, tedy konkrétní třída.

Důvod, proč se služby přidávají jakožto rozhraní je prostý. Všude v kódu, kde budeme danou službu vyžadovat, vlastně budeme vyžadovat pouze její rozhraní, což znamená, že můžeme implementaci zaměnit, kdykoliv budeme chtít a všechno bude fungovat tak, jak má.



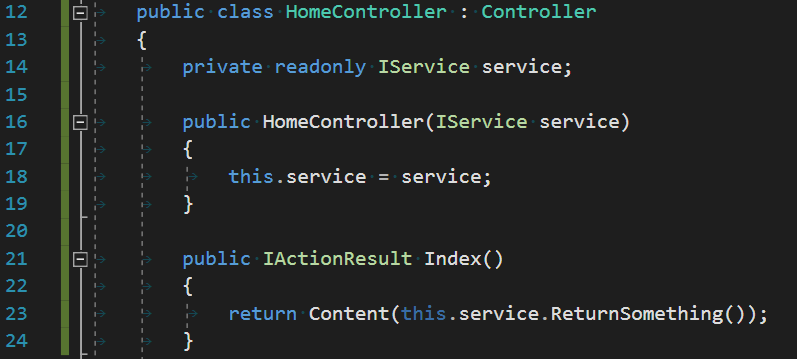
#### Vyžádání služby

Na řadu teď přichází vyžádání této služby. Služba se dá vyžádat hned několika způsoby, z nichž každý se hodí pro jiné využití.

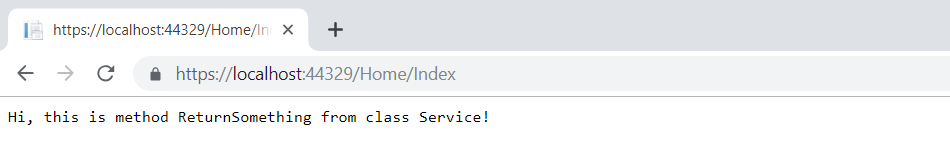
Způsoby vyžádání služby

* předání služby do konstruktoru třídy
* předání celého ServiceProvideru do konstruktoru třídy
* vyžádání služby v Razor šabloně pomocí příkazu @inject

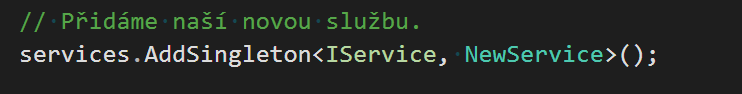
Na řadu teď přichází vyžádání této služby. Toho docílíme tím, že v konstruktoru třídy budeme vyžadovat tuto službu jako parametr. Service Provider ji nám při vytvoření instance třídy pak do konstruktoru dodá.

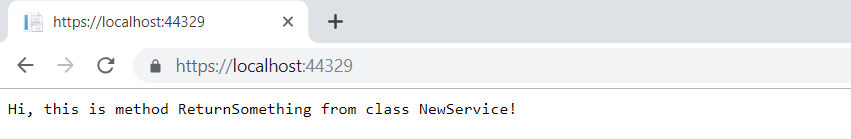


Aplikace by se nyní měla chovat následovně. Pokud načteme stránku /Home/Index, měl by se nám zobrazit text „Hi, this is method ReturnSomething from class Service!“.



Nyní si můžeme zkusit zaměnit implementaci.





Jak vidíme, záměna implementace služby byla opravdu velmi jednoduchá. Představte si, že by se jednalo o službu, kterou byste využívali ve stovkách jiných tříd. Service Provider nám opravdu velmi ulehčuje práci a umožňuje psát mnohem modulárnější kód.