# Dependency Injection – Inversion of Control (C#)

We vertrekken vanuit een praktisch voorbeeld om deze 2 belangrijke begrippen uit te leggen:

*In een Cafe is er een Ober die gevraagd wordt om een Biertje en Koffie te brengen.*

Naar objecten toe wordt er een Cafe (hoofdprogramma), een Ober, een Biertje en een Koffie met onderstaande code:

class Program

{

static void Main(string[] args)

{

Ober ober = new Ober();

ober.DienOp();

}

}

public class Ober

{

private Bier bier = new Bier();

private Koffie koffie = new Koffie();

public void DienOp()

{

Console.WriteLine("U heeft "+bier.DrankGeven()+ " gevraagd");

Console.WriteLine("U heeft " + koffie.DrankGeven() + " gevraagd");

Console.ReadLine();

}

}

public class Bier

{

public string DrankGeven()

{

return "een biertje";

}

}

public class Koffie

{

public string DrankGeven()

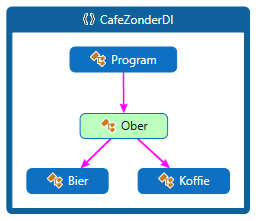
{

return "een koffie";

}

}

In dit voorbeeld instantiëren we een *Ober* die op zijn beurt een *Bier* en een *Koffie* gaat instantiëren om deze dan te kunnen opdienen. Op deze manier is de *Koffie* en het *Bier*tje ingebakken bij de *Ober* en als hij iets anders moet opdienen zoals bv. een *Limonade* dan moet die in de klasse van de *Ober* geïnstantieerd worden om deze te kunnen opdienen.



## Inversion Of Control

Beter zou het zijn dat de *Ober* alles kan opdienen wat hem aangereikt wordt vanaf het *Cafe*. Dit principe wordt **Inversion Of Control** genoemd : de afhankelijkheid van andere objecten (in dit geval de dranken) wordt van buitenaf toebedeeld in plaats van dat hij zelf voor die dranken moet zorgen.

Om elke willekeurige drank aan de *Ober* te kunnen laten opdienen, gaan we werken met een *Interface IDrank*. Deze *Interface* definieert de signatuur van een drank zodat deze door de *Ober* altijd op dezelfde manier benaderd kan worden.

public interface IDrank

{

string DrankGeven();

}

De dranken implementeren deze interface:

public class Bier : IDrank

{

public string DrankGeven()

{

return "een biertje";

}

}

public class Koffie : IDrank

{

public string DrankGeven()

{

return "een koffie";

}

}

## Dependency Injection via Constructor

Om nu de *Ober* te voorzien van wat hij moet opdienen, wordt in het *Cafe* de nodige drank geïnstantieerd, en wordt dit doorgegeven aan de *Ober*. Concreet betekent dit dat in de constructor van de *Ober* een parameter van het interface-type *IDrank* wordt meegegeven (**geïnjecteerd**).

public class Ober

{

private IDrank drank;

**public Ober(IDrank deDrank)**

{

drank = deDrank;

}

public void DienOp()

{

Console.WriteLine("U heeft "+drank.DrankGeven()+ " gevraagd");

}

}

Dit noemt met **Dependency Injection*.*** Het *Cafe* beslist welke drank door de *Ober* wordt opgediend, en wordt niet meer de *Ober* zelf. Hierdoor kan de *Ober* veel flexibeler worden ingezet, en wordt de *Ober* als het ware herbruikbaarder.

static void Main(string[] args)

{

IDrank biertje = new Bier();

IDrank koffie = new Koffie();

Ober james = new Ober(biertje);

Ober jack = new Ober(koffie);

james.DienOp();

jack.DienOp();

Console.ReadLine();

}

Bv. een nieuwe drank *Limonade* kan worden gecreëerd, die op zich de interface *IDrank* implementeert. Het *Cafe* kan deze drank instantiëren en zondermeer door de *Ober* laten opdienen.

public class Limonade : IDrank

{

public string DrankGeven()

{

return "een limonade";

}

}

static void Main(string[] args)

{

IDrank biertje = new Bier();

IDrank koffie = new Koffie();

IDrank limonade = new Limonade();

Ober james = new Ober(biertje);

Ober jack = new Ober(koffie);

Ober joan = new Ober(limonade);

james.DienOp();

jack.DienOp();

joan.DienOp();

Console.ReadLine();

}

## Dependency Injection via Property

Doordat *Idrank* met de constructor wordt meegegeven, wordt dit **Dependency Injection via Constructor** genoemd. Nadeel is dat je wel kan kiezen wat de *Ober* opdient, maar de drank niet meer kan veranderen.

Beter zou het zijn als je de *Ober* om het even wat kan laten opdienen. Dit kan je doen door te werken met **Dependency Injection via Property**. Op deze wijze kan het programma de *Ober* eerst de drank geven om daarna op te dienen.

class Program

{

static void Main(string[] args)

{

IDrank biertje = new Bier();

IDrank koffie = new Koffie();

IDrank limonade = new Limonade();

Ober james = new Ober();

james.deDrank = biertje;

james.DienOp();

james.deDrank = koffie;

james.DienOp();

james.deDrank = limonade;

james.DienOp();

Console.ReadLine();

}

}

public class Ober

{

public IDrank deDrank { get; set; }

public void DienOp()

{

Console.WriteLine("U heeft "+deDrank.DrankGeven()+ " gevraagd");

}

}

Op deze wijze moet natuurlijk eerst de *IDrank* worden toegekend alvorens hem te kunnen opdienen.

## Dependency Injection via Method

Het meest algemeen zou je kunnen stellen, is dat het programma met één instructie de *Ober* kan laten opdienen wat hij wil. Dit kan je doen door te werken met **Dependency Injection via Method**.

class Program

{

static void Main(string[] args)

{

IDrank biertje = new Bier();

IDrank koffie = new Koffie();

IDrank limonade = new Limonade();

Ober james = new Ober();

james.DienOp(biertje);

james.DienOp(koffie);

james.DienOp(limonade);

Console.ReadLine();

}

}

public class Ober

{

public void DienOp(IDrank deDrank)

{

Console.WriteLine("U heeft "+ deDrank.DrankGeven() + " gevraagd");

}

}