C# OO Programmeren

LES 4

JOHAN DONNÉ

Overzicht

- Interfaces – techniek
- Polymorfisme via interfaces
- Interfaces als abstractie in lagenmodel

- ‘loose coupling’
- Loose coupling via interfaces & Dependency Injection (DI)

Interfaces

Interface

= verzameling van methode-signaturen die samen horen.

Moeten steeds als één geheel geïmplementeerd worden door een klasse

Vormen soort ‘contract’ waar een klasse zich toe kan verbinden

```
public interface IDrawable
{
    Color DrawColor { get; set; }
    bool IsVisible { get; set; }
    void Draw();
}
```

```
public class Line : IDrawable
{
    private Point start, end;

    public Line(Point start, Point end)
    {
        this.start = start;
        this.end = end;
    }
    public Color DrawColor { get; set; }
    public bool IsVisible { get; set; }

    public void Draw()
    {
        if (IsVisible) DrawLine(start, end);
    }

    ...
}
```

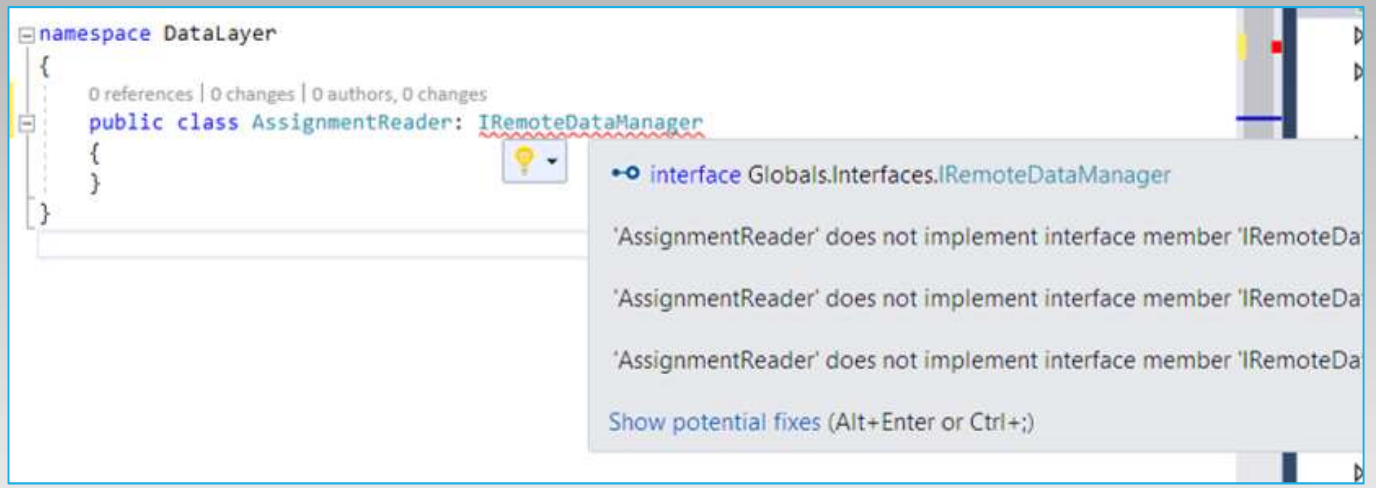
Interface

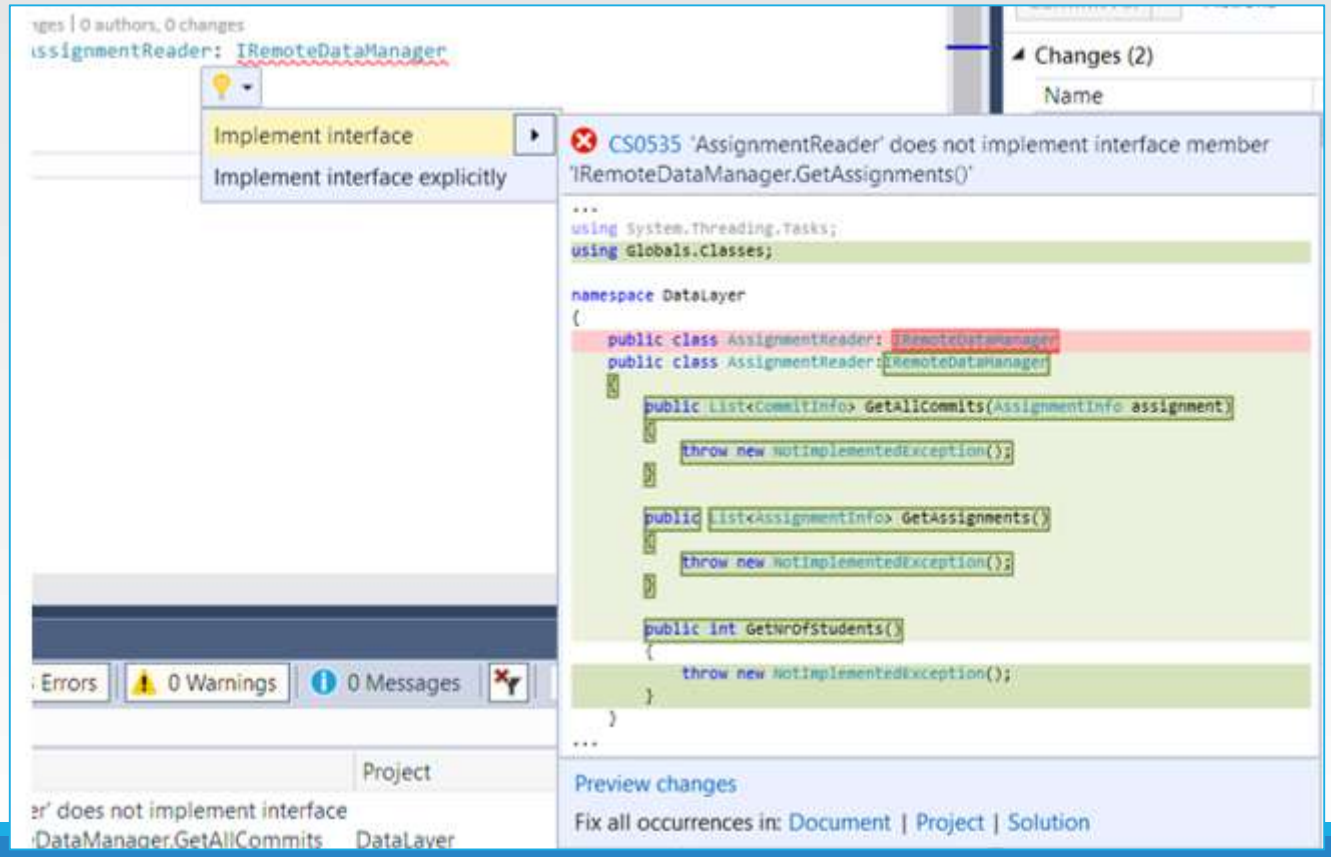
```
public interface IDrawable
{
    Color DrawColor { get; set; }
    bool IsVisible { get; set; }
    void Draw();
}
```

- Conventioneel start de naam van een interface met ‘I’
- Alle members van een interface zijn impliciet ‘public’
- Dus: geen access modifiers toegelaten
- Members kunnen ook niet static zijn.
- Geen velden, constructors
- Kan gebruikt worden als een type:

```
IDrawable something = new Line(p1, p2);
something.Draw();
```

Interface


namespace DataLayer
{
0 references | 0 changes | 0 authors, 0 changes
public class AssignmentReader: IRemoteDataManager
{
}
}
interface Globals.Interfaces.IRemoteDataManager
'AssignmentReader' does not implement interface member 'IRemoteDa
'AssignmentReader' does not implement interface member 'IRemoteDa
'AssignmentReader' does not implement interface member 'IRemoteDa
Show potential fixes (Alt+Enter or Ctrl+;)


ges | 0 authors, 0 changes
ssignmentReader: IRemoteDataManager
Implement interface
Implement interface explicitly
Changes (2)
Name
CS0535 'AssignmentReader' does not implement interface member 'IRemoteDataManager.GetAssignments()'
using System.Threading.Tasks;
using Globals.Classes;
namespace DataLayer
{
public class AssignmentReader: IRemoteDataManager
public class AssignmentReader:IRemoteDataManager
{
public List<CommitInfo> GetAllCommits(AssignmentInfo assignment)
{
throw new NotImplementedException();
}
public List<AssignmentInfo> GetAssignments()
{
throw new NotImplementedException();
}
public int GetNrOfStudents()
{
throw new NotImplementedException();
}
}
Errors
0 Warnings
0 Messages
Project
er' does not implement interface
DataManager.GetAllCommits
DataLayer
Preview changes
Fix all occurrences in: Document | Project | Solution

Polymorfisme via interfaces

Interface kan als type gebruikt worden:

```
IDrawable something = new Line(p1, p2);
something.Draw();
```

Verschillende klassen kunnen zelfde interface implementeren

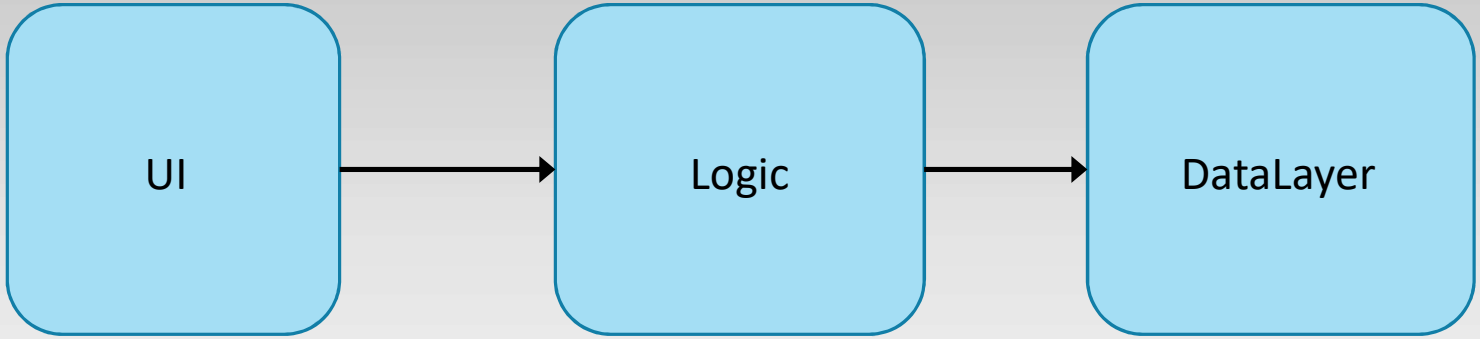
```
var drawList = new List<IDrawable>
            {
                new Line(new Point(0,0),new Point (10,10)),
                new Circle(new Point(5,5),5),
                new Rectangle(new Point (1,1), new Point(4,4))
            };

            foreach (var shape in drawList)
            {
                shape.Draw();
            }
```

⇨ Polymorfisme !

Interfaces als abstractie in lagenmodel

Structuur zonder interfaces:




```
public partial class Form1:Form
{
   private Logic logic = new Logic();
   ...
}
```

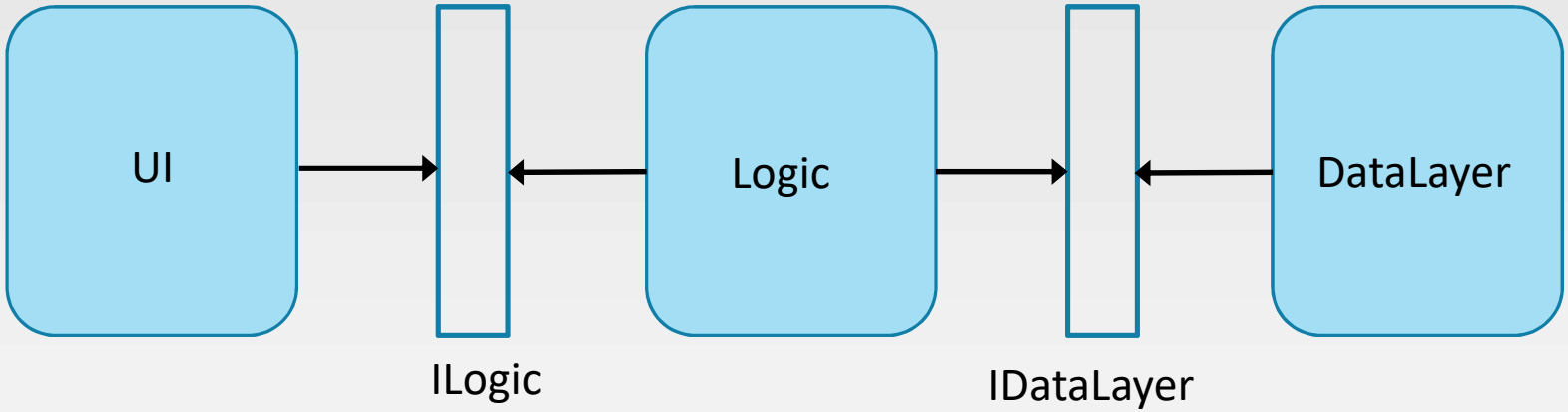
```
public class Logic
{
   private Datalayer datalayer = new Datalayer();
   ...
}
```

```
public class Datalayer
{
   ...
}
```

Gevolgen van rechtstreekse koppeling

- ‘Form’ gebruikt altijd de ‘Logic’ implementatie en de ‘Datalayer’ implementatie.
- Geïsoleerd testen van bv. de logische laag niet mogelijk.
- Verleidelijk om ‘en parcours’ het gedrag van bv. datalaag aan te passen aan noden van logische laag.
  Gevaar voor vertroebelen van logische opsplitsing in modules
- Wijziging in bv. datalaag > ook hercompilatie van ‘Logic’ en ‘Form’ (niet mogelijk om dynamisch één implementatie van datalayer te vervangen door een andere zonder hercompilatie van Logic)

Structuur met interfaces:



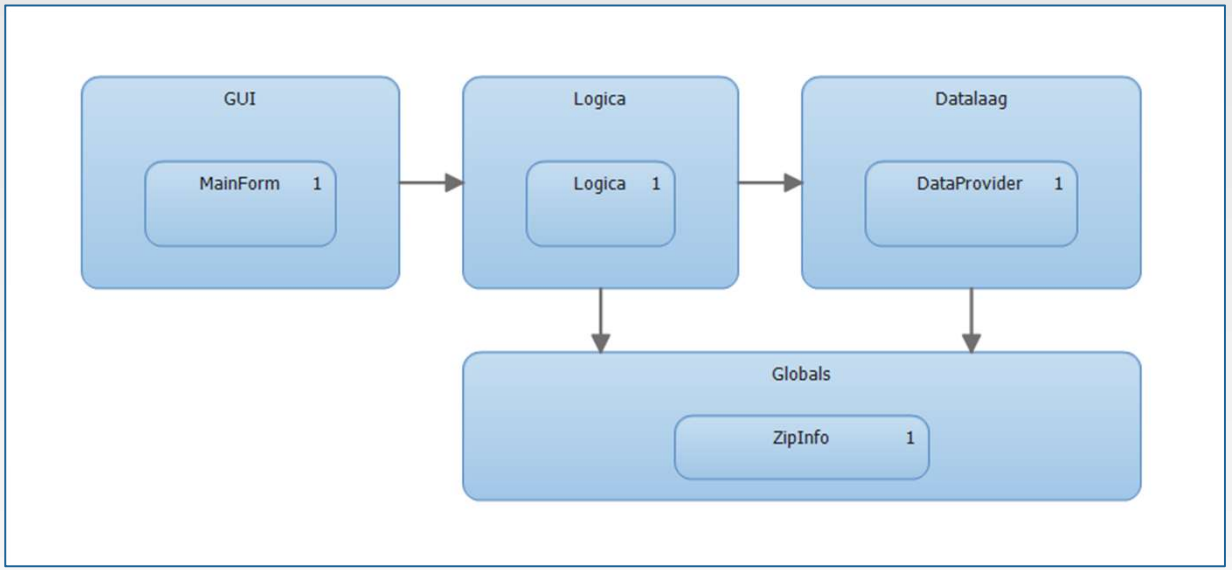
Zin van interfaces tussen de lagen

- Techniek om een grote toepassing beter op te splitsen in verschillende losse modules
- Enige koppeling tussen twee modules is de ‘Interface’

‘Interface’ :

- ‘contract’ tussen twee modules
- abstractie van gedrag van een module (laag)
- Dus op voorhand de verschillende modules afbakenen samen met hoe ze interageren.

Voorbeeld gemeentezoeker



Implementatie voor GUI

```
using System;
using System.Windows.Forms;
using GemeenteZoeker.Logica;

namespace GemeenteZoeker
{
    3 references | 0 changes | 0 authors, 0 changes
    public partial class MainForm : Form
    {
        private readonly PostcodeLogica logica;

        1 reference | 0 changes | 0 authors, 0 changes
        public MainForm()
        {
            InitializeComponent();
            logica = new PostcodeLogica();
        }

        1 reference | 0 changes | 0 authors, 0 changes
        private void ZoekTextChanged(object sender, EventArgs e)
        {
            resultBox.Lines = logica.GetMatchingResults(zoekBox.Text).ToArray();
        }
    }
}
```

Interactie tussen lagen:

Gui

```
public List<string> GetMatchingResults(string query)
```

Logica

```
public List<ZipInfo> GemeenteLijst { get; }
```

Datalaag

⇨ Mooi afgelijnd. Ziet er goed uit.

Scenario: overschakeling naar Web API

Eigen bestand/database met informatie onderhouden is niet optimaal.

==> https://opzoeken-postcode.be/

Bv.: https://www.opzoeken-postcode.be/9031.json

Eigen software aanpassen naar gebruik van online API…

Maar wat aanpassingen nodig…

overschakeling naar Web API – probleem 1

Datalaag:

```
public List<ZipInfo> GemeenteLijst { get; }
```

Kan nu niet meer: geen toegang tot volledige databank achter de Web API

En: logische laag gaat uit van implementatiedetail van de datalaag
(volledige lijst beschikbaar) ==> geen goede scheiding

oorzaak: vroegere logica-laag bevatte eigenlijk aspecten die in datalaag thuishoorden

Bijkomende bedenking:

De gevraagde informatie komt onder de vorm van een ‘List<string>’.
Handig voor de UI, niet voor gebruik door andere modules
(zoals ondersteuning bij invullen Forms, automatische mailing…).

overschakeling naar Web API – probleem 2

Beschikbare informatie niet identiek:

```
9300;AALST;Oost-Vlaanderen
9880;AALTER;Oost-Vlaanderen
3200;AARSCHOT;Vlaams-Brabant
8700;Aarsele;West-Vlaanderen
```

```
[{
"Postcode":{
  "postcode_hoofdgemeente":"9000",
  "naam_hoofdgemeente":"Gent",
  "postcode_deelgemeente":"9031",
  "naam_deelgemeente":"Drongen",
  "taal":"N",
  "region":"VL",
  "longitude":"3.6626560000",
  "latitude":"51.0503100000"}
}]
```

Onze ZipInfo structuur is specifiek gericht op de originele info:

```
public struct ZipInfo :
{
    public int Zipcode { get; }
    public string Gemeente { get; }
    public string Provincie { get; }
}
```

overschakeling naar Web API – Stappen:

1. ‘ZipInfo’ entity algemener maken (en class ipv struct):

```
public class ZipInfo :
{
    public int Zipcode { get; }
    public string Gemeente { get; }
    public string ExtraInfo { get; }
}
```

2. UI krijgt ‘List<ZipInfo>’ ipv ‘List<string>’

3. Logicalaag & Datalaag samenvoegen (met nieuwe implementatie)

Gevolg: Aanpassingen in Globals, Dataprovider, Gui en consoletoepassing!

Resultaat: zie demo.

Verdere gevraagde aanpassingen:

1. ‘Dure’ en trage API: lokale caching van gegevens
2. Logging van API-calls (controle facturatie provider)
3. Authenticatie van gebruikers voor gebruik dataprovider

…

==> telkens aanpassingen op verschillende plaatsen

+ ‘harde’ koppeling tussen modules => moeilijk afzonderlijk te testen.

Fundamentele aanpak: losse koppeling

1. Voor elke laag/module een interface declareren
2. Elke laag: bijhorende interface implementeren
3. Geen ‘new’ meer voor aanmaak van modules die nodig zijn, maar ‘dependency injection’ via het interfacetype.

Opmerking: koppeling van modules (via DI) gebeurt vanuit het opstartpunt van de toepassing: ‘Program.cs’.

Losse koppeling: concreet

1. IDataProvider interface declareren:

```
namespace Globals
{
    7 references | 0 changes | 0 authors, 0 changes
    public interface IDataProvider
    {
        3 references | 0 changes | 0 authors, 0 changes
        List<ZipInfo> GetMatchingResults(string query);
    }
}
```

```
namespace Datalaag
{
    2 references | 0 changes | 0 authors, 0 changes
    public class DataProvider : IDataProvider
    {
        private const string baseUri = "https://www.opzoeken-postcode.be/";

        3 references | 0 changes | 0 authors, 0 changes
        public List<ZipInfo> GetMatchingResults(string query)
        {
            using (var httpClient = new HttpClient { BaseAddress = new Uri($"{
            {
```

Losse koppeling: concreet

2. Constructor ‘dependency injection’ via het interfacetype.

```
using Globals;
using System;
using System.Windows.Forms;

namespace GemeenteZoeker
{
    3 references | 0 changes | 0 authors, 0 changes
    public partial class MainForm : Form
    {
        private readonly IDataProvider dataProvider;

        1 reference | 0 changes | 0 authors, 0 changes
        public MainForm(IDataProvider dataProvider)
        {
            InitializeComponent();
            this.dataProvider = dataProvider;
        }
```

Losse koppeling: concreet

3. Modules koppelen:

```
[STAThread]
0 references | 0 changes | 0 authors, 0 changes
static void Main()
{
    IDataProvider dataProvider = new DataProvider();

    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new MainForm(dataProvider));
}
```

⇨ ‘MainForm’ heeft niet meer zelf controle over backend, maar krijgt die ‘geïnjecteerd’ van buiten af.

⇨ ‘Dependency Injection’ (‘Inversion of Control’, IOC)

Losse koppeling: resultaat

Zie demo

(met toevoeging van aanpassingen)

Wat hebben we vandaag geleerd?

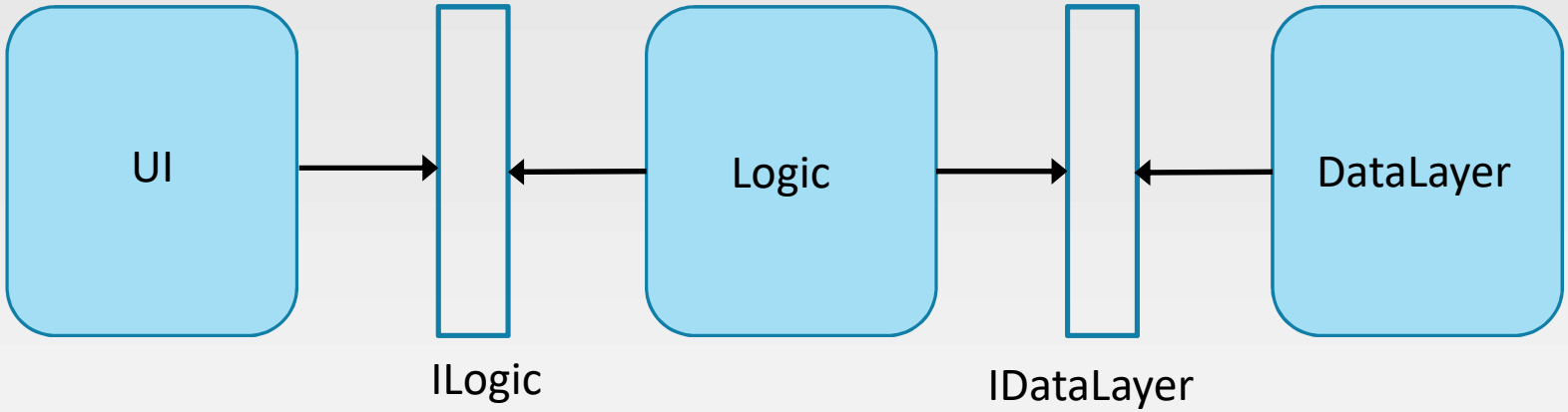
Zin van interfaces met DI tussen de lagen

Gevolg van Interface tussen de lagen:

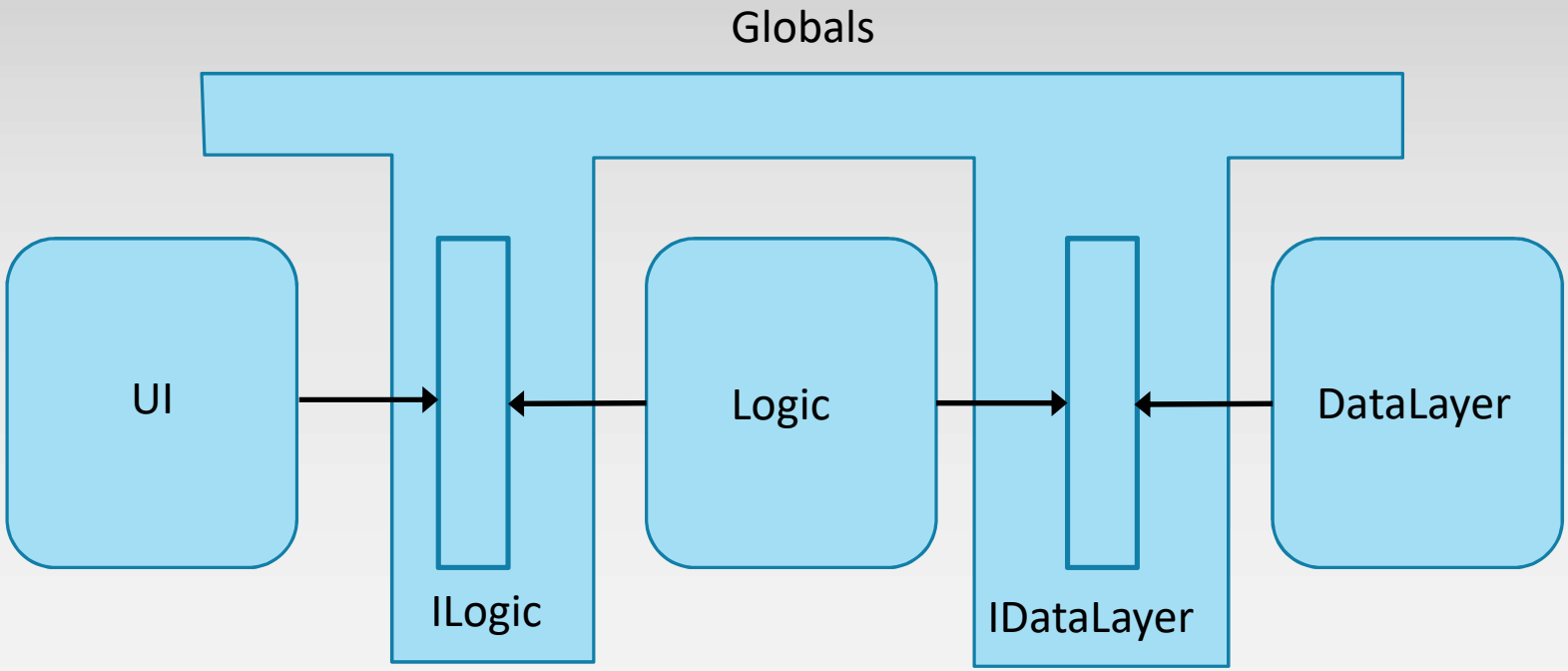
- Gedwongen op voorhand na te denken over gedrag van elke module: beter ontwerp, meer onafhankelijke modules (onderhoudbaarheid, foutzoeken, ontwikkeling in teams)
- Geïsoleerd testen van modules mogelijk (door zelf 'dummy' dependencies te injecteren)
- Meer flexibiliteit mogelijk (bv. 'at runtime' koppelen van andere implementatie van datalaag)
- Beperkte hercompilatie bij beperkte wijzigingen

Voordeel pas duidelijk bij grotere projecten met meer functionaliteit, complexiteit.

Basistemplate voor architectuur:



Basistemplate voor architectuur:



Basistemplate voor architectuur: